# Access Control from an Intrusion Detection Perspective[1]

Virginia Nunes Leal Franqueira
Centre for Telematics and Information Technology, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
franqueirav@ewi.utwente.nl

February 28, 2006

[1]This research is supported by the research program Sentinels (www.sentinels.nl). Sentinels is being financed by Technology Foundation STW, the Netherlands Organization for Scientific Research (NWO), and the Dutch Ministry of Economic Affairs.

Technical Report TR-CTIT-06-10

# Contents


**1 Motivation** **1**
- 1.1 From access control to intrusion detection . . . 1
- 1.2 From intrusion detection to access control . . . 2
- 1.3 Narrowing the gap between access control and intrusion detection 4
- 1.4 Organization of the report . . . 6

**2 Access Control Policy Models** **7**
- 2.1 Discretionary Access Control . . . 7
- 2.2 Mandatory Access Control . . . 8
- 2.3 Role-Based Access Control . . . 9
- 2.4 Policy Models Comparison . . . 9

**3 Comparison Framework** **11**
- 3.1 Types of Access Control Policy . . . 12
- 3.2 Elements . . . 13
- 3.3 Constraints . . . 14
- 3.4 Comparison Structure . . . 15

**4 Access control Languages** **17**
- 4.1 Ponder . . . 18
  - 4.1.1 Ponder access control elements . . . 18
  - 4.1.2 Ponder permissions . . . 19
  - 4.1.3 Ponder extensions . . . 20
  - 4.1.4 Ponder access control models . . . 21
  - 4.1.5 Ponder comparison summary . . . 21
- 4.2 Law-governed Interaction - LGI . . . 22
  - 4.2.1 LGI complementary concepts . . . 23
  - 4.2.2 LGI access control elements . . . 23
  - 4.2.3 Roles in LGI . . . 26
  - 4.2.4 LGI permissions . . . 26
  - 4.2.5 LGI access control models . . . 28
  - 4.2.6 LGI comparison summary . . . 29
- 4.3 Security Policy Language - SPL . . . 30
  - 4.3.1 SPL access control elements . . . 30

4.3.2 Grouping in SPL . . . . . 30
4.3.3 SPL permissions . . . . . 31
4.3.4 SPL access control models . . . . . 33
4.3.5 SPL comparison summary . . . . . 34
4.4 Policy Description Language - PDL . . . . . 34
4.4.1 PDL comparison summary . . . . . 35

**5 Discussion** **40**

**6 Conclusions** **43**

**Abstract**

Access control and intrusion detection are essential components for securing an organization's information assets. In practice, these components are used in isolation, while their fusion would contribute to increase the range and accuracy of both. One approach to accomplish this fusion is the combination of their security policies. This report pursues this approach by defining a comparison framework for policy specification languages and using this to survey the languages Ponder, LGI, SPL and PDL from the perspective of intrusion detection. We identified that, even if an access control language has the necessary *ingredients* for merging policies, it might not be appropriate due to mismatches in overlapping concepts.

# Chapter 1

# Motivation

*Is there a fuzzy area between intrusion detection and access control?* We hypothesize that the answer is *yes* and it motivates this report. We will discuss this fuzzy area looking in two directions: from access control to intrusion detection and from intrusion detection to access control.

## 1.1 From access control to intrusion detection

The three basic objectives of security, which are confidentiality, integrity and availability, aim to guarantee that information is safeguarded from misuse, modification and denial of access, respectively. In order to achieve these objectives, access control is not sufficient on its own and must be supported by two kinds of mechanisms: authentication and auditing [Dam02, Sch03]. First, we have the authentication process which is concerned with verifying the identity of a user, by checking passwords, credit card numbers, PIN numbers and so on. Second, we have access control itself which "is concerned with limiting the activity of legitimate users who have been successfully authenticated", according to Damianou et al. [DDLS00]. Finally, we have the auditing process which consists of registering access data (user requests and activities) in an *audit trail* or *audit log* for later analysis. However, this analysis can be performed automatically by an Intrusion Detection System (IDS), not only in a passive way, by analyzing audit records off-line, but also in an active way, by analyzing suspected security violations in real-time.

Sandhu, in 1997 [SS97], has already made the connection between access control and intrusion detection. Figure 1.1 is an adapted combination of Sandhu's figures 1 and 10 [SS97]. Thus, it shows the sequence of four mutually supporting mechanisms[1]: subject's authentication, control over subject's access to an object (according to authorization policies), logging of access data into an audit log database and, finally, auditing i.e. log analysis performed by an IDS. As

[1]The mechanisms and databases are illustrated separately just to demonstrate their different functionalities. In practice, it might not occur as shown.

a result of this analysis, the IDS issue alerts for administrators about security policy violations[2].

In summary, from the access control point of view, access control and intrusion detection complement each other by tracking users' misbehavior, misuse of users' privileges and security flaws which can be an indication of:

- Trojan Horse attacks, which occur when a malicious but apparently harmless program is installed in a host machine;
- host penetration[3], used by intruders as a step towards a more sophisticated attack;
- inside intruders[4] attacks, such as password cracking, information theft and falsification, system configuration changes used by intruders to set backdoors[5];
- intruder tracks covering (also called masquerading), achieved by altering log entries and disguising the steps used to gain access.

## 1.2 From intrusion detection to access control

An IDS is a software sensor which detects malicious activity based on signatures, i.e. patterns of known attacks, or based on unusual or abnormal user behavior. The former is referred to as a signature-based IDS and the latter as an anomaly-based IDS. Thus, we are talking here about *malicious activity*, *unusual behavior*, *abnormal behavior* and *attack*. All these expressions involve the notion of *threat*. Anderson [And80], in the 80's, defined threat and attack as follows.

A *threat* is "the potential possibility of a deliberate unauthorized attempt to: access information, manipulate information, render a system unreliable or unusable".

An *attack* is "a specific formulation or execution of a plan to carry out a threat".

Based on Anderson's definitions, we can argue that an Intrusion Detection System is a mechanism that detects attacks[6] to mitigate or avoid threat. Besides, a threat is directly related to unauthorized access and manipulation to information and systems. Therefore, again we find a connection between access control and intrusion detection. However, there are complications.

[2]Violation of a policy represents violation of a norm specified by a policy [WM93].

[3]Penetration is to "obtain (unauthorized) access to files and programs or the control state of a computer system" [McH01]

[4]We consider inside intruders anyone who has access to the Local Area Network of the organization, being legitimate user or not.

[5]Backdoor is an open, but not evident, access to a host made available by an intruder for later use when launching an attack.

[6]More recently, an attack has been differentiated from an intrusion [KVV05] as a successful violation of a security policy. While an intrusion is a sequence of related actions that result on a compromised state of a target system. However, for the purpose of this report these two expressions will be used as synonyms.

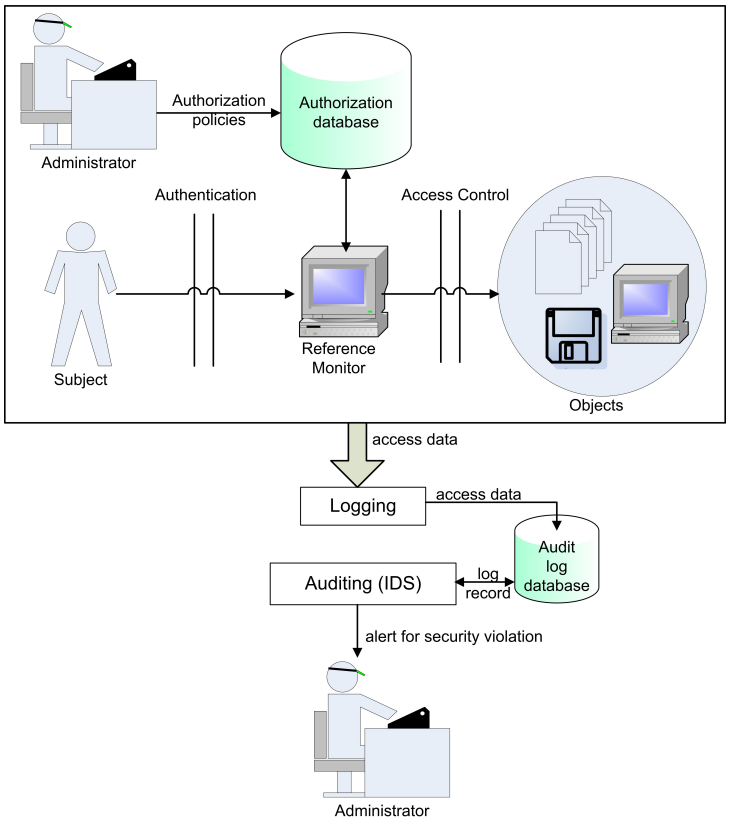


Figure 1.1: Authentication, Access Control, Auditing and passive Intrusion Detection

There are two main sources of data for an IDS: data collected from a single host or single application, or from network traffic. An IDS that uses the former is called a Host IDS (HIDS); an IDS that uses the latter is called a Network IDS (NIDS). A hybrid IDS combines both approaches and can also combine different detection methods. IDSs scan network traffic or also incoming and outgoing host traffic to find potentially malicious packets. Thus, they analyze packets at OSI layers 3 (Network) and 4 (Transport) but are unable to consider the semantics of application protocols like HTTP, for example. As a consequence, IDSs are usually ineffective to detect inside intruders who have access to more information than external intruders and may even be familiar to the security controls of the applications, but who could still be detected by closely inspecting the nature of their interactions within the applications.

The use of application semantics to detect more subtle attacks can be found in the literature since 1986 when Discovery [Ten86], a database IDS, was reported [McH01]. Since then, three different types of application-based IDSs

have emerged [WH01]. In the first type, the IDS uses intercepted traffic going in and out of the application. In the second type, the IDS relies on third-party logs from Operating Systems, databases and firewalls [CGL99, Ten86]. Finally, in the last type, the IDS directly uses internal application messages and library calls [AJ01, AL01]. Thus, the last group provides the possibility of bidirectional on-line interaction between the IDS and the application, and more precise IDS response and analysis. However, although IDSs are increasingly taking advantage of the application semantics, it seems to be possible to narrow even more the gap between access control and intrusion detection.

## 1.3 Narrowing the gap between access control and intrusion detection

Although a theoretical overlap exist between access control and intrusion detection, as discussed in Sections 1.1 and 1.2, in practice there is a gap between them. We present next three alternatives to narrow this gap, as shown in Figure 1.2.

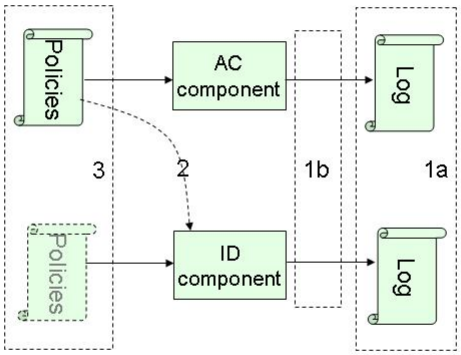


Figure 1.2: Three alternatives to narrow to gap between intrusion detection and access control

1. **Violation as event**[7] (alternative number 1 in Figure 1.2). This alternative explores the combination of output from access control and intrusion detection components. This combination can be acquired in two ways.

[7]The concept of event according to the Intrusion Detection Working Group (IDWG) is an occurrence in the data source that is detected by the sensor and which may result in an alert being transmitted. The IDWG is part of the Internet Engineering Task Force (IETF).

- Correlation of log events (alternative number 1a in Figure 1.2). This approach relies on the correlation of log entries, i.e. alerts, to abstract attack scenarios. Different data models for alert correlation have been proposed in the literature to allow interoperability between different kinds of security sensors such as IDSs, firewalls and routers. One of these data models is the Intrusion Detection Message Exchange Format (IDMEF) [DCF06] proposed by the Intrusion Detection Working Group (IDWG) [WE02]. It is a standard representation of intrusion alerts based on the Extensible Markup Language (XML) syntax. Many correlation engines [CCM02, GG05] build up correlation engines based on IDMEF format logs. Other data models have also been proposed along with correlation methods such as the one by Kruegel [KVV05] which comprehends steps of alert normalization, aggregation, correlation and prioritization.
- Correlation of real-time events (alternative number 1b in Figure 1.2). This approach relies on the correlation of events as soon as they happen to abstract attack scenarios. The correlation of events in real-time is a difficult task because of the huge amount of events which occur everyday in large networks. One approach to analyze real-time events is the splitting of the data stream into smaller and more manageable streams [KVVK02]. Another approach is to have real-time event sensors scattered across the network feeding a centralized correlation engine. However, this approach has the draw back of sensors control and coordination.

2. **Extra information** (alternative number 2 in Figure 1.2). Access control policies can be used as extra information when modeling normal behavior on a specification-based anomaly detector [SGF+02, KRL97]. The combination of policies (signature detection), and anomaly detection allows decreasing the main problem of the latter, i.e. the high rate of false alarms, while preserving the ability to detect unknown attacks, the main problem of the former approach.

3. **Merged policy** (alternative number 3 in Figure 1.2). This alternative explores the combination of input from access control and intrusion detection components, assuming that both input are in the format of policies. We notice in the literature that these two research fields are particularly apart in terms of policy specification. The distance between these two domains may have a probable explanation: while access control involves concepts from the organizational domain, such as subjects, objects and permitted actions, intrusion detection involves concepts from a much more technical domain, such as the network infra-structure, network nodes, protocols and packets. This report focus on this alternative, as outlined in Section 1.4.

## 1.4 Organization of the report

We have discussed in Section 1.3 three ways to narrow the existing gap between intrusion detection and access control. This report, however, focus in only one alternative, the "Merged policy". Therefore, an evaluation framework is proposed to (1) investigate if the syntax of access control languages support expressing some basic elements needed for intrusion detection policies and to (2) identify if the access control languages meet a set of criteria that will be described in section 3. These criteria identify which policy types and constraints are relevant from the perspective of an IDS.

We motivate our choices for the access control languages after we have presented the comparison criteria in Chapter 3.

Therefore, Chapter 2 gives a comparative overview of the generic security policy models found in the literature, Chapter 3 discusses the comparison criteria which will be used to analyze the policy specification languages. Chapter 4 presents the chosen access control policy specification languages and a comparison summary table for each language. Chapter 5 discusses the applicability of the found most promising access control language in terms of merging policies for the specification of attack scenarios. Finally, in Chapter 6, conclusions and future work is presented.

# Chapter 2

# Access Control Policy Models

The languages compared in Chapter 4 allow the specification of access control policies. However, access control policies can follow three different security models. In the current chapter, we provide an overview of these models: Discretionary Access Control (DAC), Mandatory Access Control (MAC)[1] and Role Based Access Control (RBAC) [PP03]. DAC, MAC and RBAC have all been recognized as official standards, the first two models by the U.S. Department of Defense Orange Book [SS97] and the last model as a NIST[2] standard [FSG+01].

## 2.1 Discretionary Access Control

As we have seen in Chapter 1, *access control* is about controlling which subjects (active entities, such as users or systems) have authorized access, depending on access rights and modes, to which object (passive entities, such as files, bank accounts or printers). In the Discretionary Access Control (DAC) model, access is granted or denied based on the triple (s, o, m), where s is the set of subjects, o is the set of objects and m is the set of access modes (e.g. read, write and execute). An access matrix uses this triplex information to obtain authorizations, as shown in the example 2.1. Thus, the state of the matrix at a point in time represents the authorization state of a subject over objects. The size of the matrix can become very large because of empty spaces (i.e. no permissions for a subject on an object) and because a single user can represent several subjects, depending for example, on which project he is working on. Therefore, in practice this matrix is usually implemented as Access Control Lists (linked lists headed by object, used in Unix OS), Capabilities Lists (linked lists headed by subject) or Relations (table with a single authorization per row, used

[1] The Chinese Wall model [BN89] is considered part of the MAC model because both are lattice-based model [San92]. Thus, is not presented as a separate model in this report.

[2] National Institute of Standards and Technology

in database management). A review of different implementation is provided by Sandhu [SS97] and Samarati [SdV01].

Access rights, in the DAC model, are administrated in an owner-based fashion. This administration has three variations: (1) In a Strict DAC model, the owner of an object can propagate access rights to other subjects but not transfer ownership; (2) In a Liberal DAC model, the owner can delegate the authority to propagate access rights to other subjects; (3) In a DAC with ownership change, the owner can transfer ownership to other subjects.

The DAC model is also known as Commercial Security Policy. Clark and Wilson [CW87] uses policies based on "well-formed transactions", which embodies the concept of access matrix for deriving authorizations based on the triplex (user, data items, transactions procedure) to introduce the concept of separation of duties [FSG+01] later on incorporated to the RBAC model.

| | Object 1 | Object 2 | Object 3 |
|---|---|---|---|
| Subject 1 | read | read<br>write | execute |
| Subject 2 | | read | read<br>write<br>execute |
| Subject 3 | read<br>write | read<br>write<br>execute | |

Table 2.1: Access Matrix example

## 2.2 Mandatory Access Control

The Mandatory Access Control (MAC) model differs from the DAC model because, while the latter enforces access based on subject discretion, the former enforces access based on mandatory rules determined by an authority. Therefore, in the MAC model the access is granted or denied based on: the subject's clearance level and the object's security level. *Both* these levels are determined by (1) the confidentiality of the information (hierarchically defined) such as Top Secret, Secret and Confidential and by (2) a project[3] such as Nuclear, Crypto, and Alpha. Access to information (i.e. to objects) is based on the "need-to-know" principle and, therefore, a subject can only access an object with confidentiality level below or equal to his clearance level *and* belonging to the the same project[4]. Moreover, two other properties, called *-Property, are required to hold: (1) *read-down* property defines that a subject's clearance

[3]The concept of projects can be replaced by categories or compartments.

[4]This basic access rule is called *Simple Security Property* [Dam02].

must dominate the object's security level[5]; (2) *write-up* property defines that a subject's clearance must be dominated by the object's security level[6].

The MAC model, also known as Military Security Policy, was initially formalized by Bell and La Padula [BP73] and enforces data confidentiality. Biba's model followed the same theory but enforces data integrity [Bib77]. Therefore, Biba's model defines integrity levels (e.g. Crucial, Important, Unknown) analogous to Bell-LaPadula confidentiality levels. Both Bell-LaPadula and Biba models enforce that MAC is a lattice-based model since they provides a "one-direction information flow in a lattice of security labels"[7], according to Sandhu [SS94].

## 2.3 Role-Based Access Control

The Role-Based Access Control model (RBAC) combines the strengths of both the DAC and MAC models: it grants access to subjects on objects (such as in DAC) but allows regulations on the use and assignment of authorizations (such as in MAC). Subjects are assigned to roles and permissions are also assigned to roles, therefore, access is granted or denied based on roles. The RBAC core model [FSG+01] defines five basic elements: (1) *user*: human beings; (2) *role*: job function empowering authority and responsibilities which are extended to the user assigned to the role; (3) *object*: entity that contains or receives information; (4) *operation*: actions or functions which can be invoked by a user; (5) *permission*: approval to perform an operation on objects. Furthermore, RBAC defines the concept of *session* as mapping between a user and a set of activated roles assigned to the user. Therefore, a user can play different roles and hence have different permissions in different sessions.

The RBAC model has been first introduced in the mid 90's and, since then, became a standard security model widely spread among different applications and domains. For a history line on RBAC, refer to [FSG+01, Sch03, Dam02].

## 2.4 Policy Models Comparison

Table 2.2 shows a parallel summary of the models presented in sections 2.1, 2.2 and 2.3.

[5]It means that the object's confidentiality level must be less than or equal to the subject's confidentiality level and also that the object's project or category must be a sub-set of the subject's project. For example, an object with security level <Secret,Nuclear> could be read by someone with clearance level <Top Secret, [Nuclear,Alpha]> but not by someone with clearance level <Top Secret,Alpha>

[6]It means that the subject's confidentiality level must be less than or equal to the object's confidentiality level and also that the subject's project must be a sub-set of the object's project or category. For example, someone with clearance level <Secret,Nuclear> could write to an object with security level <Top Secret,[Nuclear, Alpha]> but not to an object with security level <Confidential,Nuclear>.

[7]Labels refer to the classification in Top Secret, Secret, etc and in projects.

| | Discretionary Access Control | Mandatory Access Control | Role-Based Access Control |
|---|---|---|---|
| Access | Access to objects is regulated by the subject's own discretion or judgment, if the object is owned by the subject. | Access to objects is regulated by mandatory rules imposed by a central authority. | Access to objects is regulated by the subject's authority and responsibilities in an organization. |
| Authorization | Access modes govern the access of subjects to objects. | Security levels govern the access of subjects to objects. | Roles govern the access of subjects to objects. |
| Weaknesses(-) and Strengths(+) | (-) propagation of access rights can become a safety problem, especially Liberal DAC and DAC with ownership change approaches.<br>(-) permissions management is time consuming because changing a set of permissions requires deleting old ones and entering new ones individually.<br>(+) access matrix implementation is simple and effective for linear access rights between subjects and objects.<br>(-) inadequate to enforce information flow policies.<br>(-) use of programs that creates automatic copies of an object can represent a Trojan Horse security breach (refer to Sandhu [San93, page 8]). | (-) appropriate only for enforcing security in a *one-directional* information flow lattice.<br>(+) can be applied to almost any situation where one-directional information flow is involved, according to Sandhu [San93]. Therefore, its application is not restricted to military environments. | (+) simple management of permissions.<br>(+) policy-neutral i.e. can be used to express MAC and DAC security policies.<br>(+) hierarchical roles allow the inheritance of privileges and permissions.<br>(+) least-privilege enforced since users are only empowered with privileges within a session.<br>(+) separation of duties obtained by defining conflicting roles and imposing constraints.<br>(-) user's identity is not captured by a role but may be necessary. For example, doctor "A" should only have access to his patients records and not to all records allowed by his role. |

Table 2.2: Access Control Models [FSG+01, SS97, OSM00, San93, SS94, SdV01]

# Chapter 3

# Comparison Framework

In Chapter 1, we have identified three possibilities of using access control for improving intrusions detection rate and quality. The possibility further investigated by this report is the merging of policies (section 1.2), which drives the framework used for comparing the languages. In this chapter, we set the stage for the comparison by first discussing the notion of security policy and then the comparison criteria.

The concept of security policy is not always clear. Sterne [Ste91] had already indicated back in 1991 that the term *Security Policy* is overloaded. The situation remains the same typically for three reasons. Firstly, security policies can have different levels of abstraction, from high-level policies expressed in natural language to low-level configuration rules [LSK01, MOR01]. Secondly, security policies can apply to several domains (e.g. database [BBC+02], Web documents [BCF01]) and disciplines other than computer science (e.g. national security, social security). Finally, the diversity of aspects involved with the notion of security itself: three basic objectives i.e. confidentiality, integrity and availability [LZ97] and several properties such as reliability, performance, safety, trust and privacy [SWY+02, BA05].

In this report the Ponder concept of policy is used: *a policy is a rule that defines a choice in the behavior of a system.* This concept enforces that policies are a way of parameterizing system management (e.g. with access control components) allowing changes in the behavior of a system without actually changing the implementation of its components [Slo94].

Our comparison framework is built by arguing about three aspects of access control policies that are considered essential for intrusion detection policies. First, we argue about each type of access control policy to identify if a policy violation of that type is meaningful for detecting intrusions. Second, we present basic elements of access control and intrusion detection policies that are required or represents an added-value towards the "Merged policy" possibility, discussed in section 1.2. Finally, we discuss constraints to determine which ones are relevant from the intrusion detection perspective.

## 3.1 Types of Access Control Policy

Four types of access control policy are distinguished in the Ponder specification language [DDLS01, Sch03]: authorization (positive and negative), obligation, delegation (positive and negative) and refraining. They are used as reference in this work because they are comprehensive, i.e. no other types of access control policies could be identified in the literature. We argue for each type of policy if it is interesting under the intrusion detection perspective and to which extent.

1. **Authorization policies** specify the actions a subject is permitted to perform over an object. Wieringa and Meyer [WM93] define the purpose of authorization mechanisms as to "... protect the integrity of resources in operating systems and databases, to guard the security of sensitivity resources, and to protect the privacy of sensitive data.". Therefore, intrusion detection and authorization policies are related since detection of unauthorized access is a mechanism used to guard and protect resources. The ability to express authorization policies is a requirement for comparing the languages.

2. **Obligation policies** specify the actions a subject must perform on an object, if an event occurs. These policies define the duties of the subject in an organization and require the specification of a deadline and countermeasures if the policy enforcement mechanism detects that the obligation is not fulfilled. Therefore, obligations involve a choice the subject has between fulfilling or not an authority regulation. For example, "a subject has to show valid credentials to enter a system" is not really an obligation since if the rule is not fulfilled the subject simply does not get any result. It involves no deadline or sanction actions because the outcome is immediately visible to the subject. Obligations are more subtle rules which involve watchdog guards to have it enforced. Nevertheless, subjects in a broad sense can be considered as *anything* which can initiate actions and, consequently, can be an automated agent for example. In this case, the agent can have the *obligation* to monitor the login process and report unsuccessful attempts under certain conditions. Thus, unfulfilling this obligation can represent a breach of security and thus an opportunity to intruders. Based on this reasoning, obligations will be considered as a comparison item.

3. **Delegation policies** specify a transfer of authorization from one subject to another. By definition [WM93], delegation is "an act in which the authority to perform a task is transferred to someone (the delegate), but where the final responsibility for the performance of the task is kept by the one who delegates the task (the principal of delegation)". Thus, the principal of delegation can transfer a *subset* of its authority, i.e. a subset of the actions he can perform over an object. A delegation policy maps then to two authorization policies: one for the principal of delegation and one for the delegate. A positive delegation can be revoked either

automatically by a time constraint clause indicating its validity or by the specification of a negative authorization. A negative delegation policy specifies a delegation which is forbidden, i.e. a subject cannot transfer the permission to perform a task to someone else. For example, a manager is not allowed to delegate the access rights over a cost file to employees that are not also managers [DDLS00]. Therefore, positive delegation will be considered as a comparison parameter for the same reason as authorization policies. However, negative delegation will not be considered as such because its violation seems to have no practical and direct usage for intruders.

4. **Refrain and negative authorization policies** both specify actions that a subject is *not* permitted, i.e. is forbidden, to perform over an object. However, in Ponder [Sch03] the former are meant to be enforced by authorization mechanisms active in the subject's controller, e.g. when the target[1] is not trusted by the subject, while the latter are meant to be enforced by the mechanisms active in the target's controller. This differentiation is not relevant for our comparison because it is specific to Ponder implementation. Nevertheless, negative authorization can also be used in two other circumstances: to temporarily remove access rights from a subject on an object and to restrict access when a permissive policy is the default, i.e. when "everyone" has access to "something" unless it is explicitly forbidden [DDLS00]. In both cases, intruders can exploit negative authorization policies for malicious purposes and the ability to express this type of policy will be taken as a comparison item. Nevertheless, refrain policies will be considered as already covered by negative authorization policies.

In summary, it is essential that the access control languages support authorization policies (both positive and negative), delegation policies (only positive) and obligation under certain circumstances.

## 3.2 Elements

The basic elements found in intrusion detection policies[2], extracted from the Snort Users Manual[3] [SP], are listed next. These elements represent just a small sample of network-based IDS and aim to identify the possibility of representing elements not belonging to the core of access control policies.

- Protocol (e.g. TCP/IP and UDP);
- IP Address;
- Port number;

[1] Target refers to target objects, i.e. a subject accesses target objects.
[2] We are considering signature-based IDSs.
[3] Snort is an open source IDS.

- Protocol flag (e.g. ACK and SYN);
- Packet content: to be matched with a string.

The core access control policy elements: subject, object and action are essential from the perspective of intrusion detection. Furthermore, expressing access rights in terms of sets increases scalability, manageability and efficiency.

- a set of subjects (e.g. role and group);
- a set of objects (e.g. group or directory in the computer system);
- multiple operations.

The languages ability to support the elements mentioned above will be evaluated as an indication of possible combination between access control and intrusion detection elements into merged policy.

## 3.3 Constraints

Constrains are expressions used to limit or to restrict a variable within bounds. The applicability of a policy is set by constraints. On the one hand, intrusion detection involves essentially constraints of time and causality between events. On the other hand, authorization (positive and negative) and delegation (only positive) policies involve essentially constraints of time and restrictions over values of attributes. We want to determine which constraints might be useful for combining elements from both areas into a single policy.

Kumar's signature classification for IDS [Kum95] is used as a reference for reasoning about constraints.

1. **existence** of a state at a certain point in time (e.g. the violation of an authorization policy as a whole);
2. **sequence** of several actions or activities happening either at repeated intervals (e.g. action $a$ $\Delta$ action $b$ $\Delta$ action $c$) or within a duration (e.g. $\Delta$ = [action $a$, action $b$]);
3. **patterns** of actions and activities:
   (a) concurrence of actions and activities, i.e. occurring jointly but in any order (e.g. action $a$ and action $b$);
   (b) absence of match in the entire searching space (e.g. (action $a$ and action $b$) and not action $c$);
   (c) combination of matches over generalized selection of actions and activities (e.g. from set = {action $a$, action $b$, action $c$} select the combinations which could be malicious, such as {action $a$, action $b$} and {action $b$, action $c$} for example).

Table 3.1 shows time constraints and Table 3.2 shows other constraints, extracted from the classes of signatures identified by Kumar, discussed above, and from the core access control policy elements, discussed in Section 3.2.

| Time constraints | Examples |
|---|---|
| time interval | authorization valid between 9:00 and 17:00 |
| time duration | delegation valid during 24 hours |
| date interval | authorization valid from 20/01/06 to 30/01/06 |
| every date or day of week | negative authorization repeatedly valid every Saturday |
| "after" time | authorization valid after 12:00 |
| "before" time | authorization valid before 22:00 |

Table 3.1: Time constraints summary from Kumar events *existence* and *sequence* classes

| Other constraints | Examples |
|---|---|
| if-then-else condition | if subject belongs to group then |
| boolean operators (and) | authorization applies to both actions $ab$ |
| boolean operators (or) | authorization applies to either action $a$ or $b$ |
| boolean negation | action $a$ not followed by action $b$ within an interval |
| sets operations (union) | members of group $a$ plus members of group $b$ |
| sets operations (difference) | members of group $a$ except members of group $b$ |
| sets operations (intersection) | members belonging to both groups $a$ and $b$ |

Table 3.2: Other constraints summary from Kumar events *patterns* class and access control policy elements

## 3.4 Comparison Structure

The comparison criteria are structured next, according to discussions in Sections 3.2, 3.1 and 3.3. The access control models supported by the language (seen in Chapter 2) and the availability of a toolkit and technical documentation are also considered as comparison items.

- Access control elements
- Intrusion detection elements
- Positive authorization
- Negative authorization
- Delegation

- Obligation
- Time constraints
- Other constraints (condition, boolean operations and sets operations)
- Access Control model
- Toolkit and manual availability

# Chapter 4

# Access control Languages

In this chapter, we present a list of access control languages and motivate our choice of languages for the comparison. Next, we provide an overview of the selected languages and finally we present a table, as structured in Section 3.4, for each language.

The classification of access control approaches is not clear-cut. Thus, we present in this report a combined classification from published work [Dam02, Sch03, DBSL02]. One language representative of each class has been selected for this report, even though many languages fit in more than one class. The names of the selected languages are emphasized in italic. Two groups of languages have been left out of this report: the Logic-based Languages and Trust Specification languages. The former group will remain as future work and the latter is considered out of scope because we are interested in enforcing security inside one organization and trust languages deal with cross-organizational security.

- Role-based Access Control Specification (e.g. Temporal Role-Based Access Control [BBF00], Role Definition Language [HBM98], RCL2000 [AS00], RBAC state-related constraints [CS96], *Ponder*).
- Management Policy (e.g. *Policy Description Language*, PCIM [MESW01]).
- Enterprise and Collaboration Policy (e.g. ODP-RM [ISO99], *Law-Governed Interaction*, Event-Trigger-Rules [SLL+01]).
- High-level Policy Languages (e.g. *Security Policy Language*, Tower [HV01], XACML [ftAoSIS02], LaSCO [HPL98]).
- Logic-Based Languages (e.g. Authorization Specification Language [JSS97], Z [Bos95], Alloy [Sch03], Standard Deontic Logic [WM93], Maude [DHV03]).
- Trust Specification (e.g. PolicyMaker [BFL96], KeyNote [BFIK99], Trust Policy Language [HMM+00]).

The remaining of this chapter contains the overview of the chosen languages followed by a comparative table for each of them.

## 4.1 Ponder

Ponder is a declarative language based on an object-oriented model. It is used to specify role-based policies such as access control policies [DDLS01], QoS policies [LLS03] and management policies [DDLS00]. Furthermore, it is supported by a toolkit [SDL02], which is, however, no longer available for the research community as of February 2005. The Ponder toolkit includes a compiler framework for translating policies to low-level enforceable representations such as firewall rules, Linux security kernel calls, Java objects and XML files.

### 4.1.1 Ponder access control elements

Ponder allows expressing the basic elements of access control (subject, object and action) and also permits expressing five kinds of grouping [DDLS00].

- **Domains** are sets of objects to which policies can be applied. Domains follow the concept of a file system directory, structuring objects hierarchically and allowing reference by relative and absolute path[1]. Furthermore, domains can be combined through scope expressions, evaluated each time a policy is interpreted. Therefore, it is possible to retrieve domain members, perform operations over domain sets, and to traverse the structure of sub-domains in different ways.
- **Groups** are sets of policies related semantically and, therefore, instantiated together. A group contains policies related to the same target object which can be, for example, a department or an application.
- **Roles** are sets of policies related to a position or a job description in an organization. Thus, roles allow grouping subjects with the same duties, responsibilities and rights. For example, the role *nurse* describes permissions applied to individual subjects, i.e. human being with nurse qualifications. New hired nurses are then assigned to this predefined role.
- **Relationships** are sets of policies which determine rights and duties between roles. For example, a relationship *MorningReport* could describe the obligation that subjects belonging to the role *nurse* have to issue a report addressed to subjects belonging to role *ward-supervisor* every morning. This relationship can then be instantiated for specific individuals.
- **Management structures** are sets of policies related to organizational units. It defines which roles, relationships and other management structures should be instantiated together. For example, the management structure *Ward* could contain the other structures *MaternityWard* and *EmergencyWard* and several relationships between the roles *nurse*, *patient* and *ward-supervisor*.

[1] Similar to Unix file pathnames

### 4.1.2 Ponder permissions

Ponder also supports several types of policies as described next [DDLS00].

First, positive authorization policies define which actions[2] a subject is permitted to perform on a set of objects in the target domain. An optional when-constraint clause and filters on actions can be used to limit the applicability of the policy. Figure 4.1 shows an example[3] of a positive authorization policy authorizing network administrators to perform actions *load*, *remove*, *enable* and *disable* on routers between 19:00 and 20:00.

```
inst auth+ adminRouters {
  subject  /NetworkAdmin;
  action load(), remove(), enable(), disable();
  target /routers;
  when  time.between("‘19:00,"’20:00"’);
}
```

Figure 4.1: Positive authorization in Ponder (the + sign on the first line indicates that this is a positive authorization)

Second, a negative authorization policy defines which actions a subject is forbidden to perform on a set of objects in the target domain. No filters on actions can be specified for this type of policy. Figure 4.2 shows a negative authorization policy forbidding test engineer trainees to perform action performance tests on routers.

```
inst auth- testRouters {
  subject  /testEngineers/trainee;
  action performanceTest();
  target /routers;
}
```

Figure 4.2: Negative authorization in Ponder (the - sign on the first line indicates that this is a negative authorization)

Third, delegation policies define which actions a subject allows another subject to perform on a set of object in the target domain on his behalf. These policies have three basic elements: a grantor, a grantee and a predefined authorization that the grantor must possess. The target and the list of actions on a delegation policy, if specified, must be a sub-set of the target and the action list specified in the corresponding authorization policy. Cascading delegation is also possible in Ponder. The use of a hops-clause limits how many levels of

[2] Actions in Ponder can be specified using the toolkit. They are Java classes which extend an Action superclass and are loaded by Policy Management Agents. For details refer to Ponder Implementation Guide [Pon03].

[3] All Ponder examples have been extracted from [DDLS01] with minor changes.

sub-delegations are allowed. Furthermore, the valid-clause limits the validity of the delegation. Figure 4.3 shows a delegation which has the authorization adminRouter (from example Figure 4.1) as parameter. Thus, the authorization subject, i.e. the grantor member of the /NetworkAdmin domain, can delegate the actions *enable* and *disable* to the grantee member of the /DomainAdmin domain, on objects in the target domain routers/routers-mainBuilding. After the validity period, i.e. 24 hours after the policy has been enforced, it is automatically revoked.

```
inst deleg+ (adminRouters) delegAdminRouters {
  grantee  /DomainAdmin;
  action enable(), disable();
  target /routers/routers-mainBuilding;
  valid   time.duration(24);
}
```

Figure 4.3: Delegation in Ponder

Finally, obligation policies are triggered by events and require the specification of at least one obliged action. The concept of obligation in Ponder presuppose that the subject has the corresponding authorization to perform the required actions over the object [Slo94]. Furthermore, obligations can be used for either specifying an obligation imposed on a subject and for catering for network or target object failures. No due date and sanction actions are required for an obligation in Ponder. Figure 4.4 shows an obligation triggered by 3 consecutive *loginfail* events performed by the same *userid*. Upon these events, a security administrator from */NRegion/SecAdmin* has to disable the *userid* on the */NRegion/users* domain and log the userid.

```
inst oblig loginFailure {
  on   3*loginfail(userid);
  subject  s = /NRegion/SecAdmin;
  target t = /NRegion/users^{userid};
  do t.disable() -> s.log(userid);
}
```

Figure 4.4: Obligation in Ponder

### 4.1.3 Ponder extensions

Another type of policies, called meta-policies, and some other features are also supported by Ponder.

- **Meta-policy** allows the specification of a series of constraints for a group of policies to limit permissions on a system or to avoid conflict between

policies. For example[4], the specification of the role *accountant* can include a meta-policy to make sure that the obligation *registerPayment* is performed before the obligation *IssueCheque*.

- **Policy type** uses the same concept as template and, therefore, can be instantiated by parameters. A type can contain an extends-clause for specialization through inheritance. For example, the role type *specialized-nurse* can extend the role type *nurse* by adding new obligations to the set of obligations inherited from the *nurse* role.
- **Import statements** is used to import policies and scripts from other domains. A script, i.e. a code object, can be used for complex action filters in a positive authorization.

### 4.1.4 Ponder access control models

The Discretionary Access Control (DAC) model (section 2.1) relies on the discretion of the subject which owns an object to control and grant access rights to other subjects. In Ponder, DAC policies can be specified by using authorization policies and constraints on object attributes. Figure 4.5 shows a positive authorization which restricts the actions of deleting and reading a file to the file owner. It remains at the discretion of the object's owner to delegate access rights to other subjects.

```
inst auth+ ownerAccess {
  subject s = /users/financeDept;
  target  f = /file/payroll;
  action  f.delete(), f.read();
  when  f.getOwner() = s;
}
```

Figure 4.5: DAC policy in Ponder

The Mandatory Access Control (MAC) model (section 2.2) requires subjects and objects classifications to allow access rights on a lattice structure of security labels. Damianou [DDLS00] has mapped the Bell-LaPadula model to Ponder using policies and domains but concluded that, although the mapping is possible, any addition of label or level in the lattice would represent a re-computation of the domain structure. Therefore, in practice the mapping is not viable.

### 4.1.5 Ponder comparison summary

Table 4.1 shows the comparison summary of Ponder.

[4]Example extracted from [DDLS00].

## 4.2 Law-governed Interaction - LGI

Law-Governed Interaction (LGI) [MU98, UM00, Min05] is a coordination and access control mechanism applicable to a distributed environment. As a coordination mechanism, LGI offers middleware components for message exchange between actors. Figure 4.6 shows the LGI decentralized architecture. A controller $T_x$[5] is composed of: (1) the law[6] $\mathcal{L}$, (2) the mechanism responsible for ruling the law $I$ and (3) the control state $CS_x$ of agent $x$. The combination of an actor and a controller represents an LGI-agent (e.g. human beings, software processes or clients/servers). Besides, the engagement of an actor under the regulation of a LGI-law is voluntary and the current enforcement mechanism of LGI does not support the mandatory compliance of agents to a law.

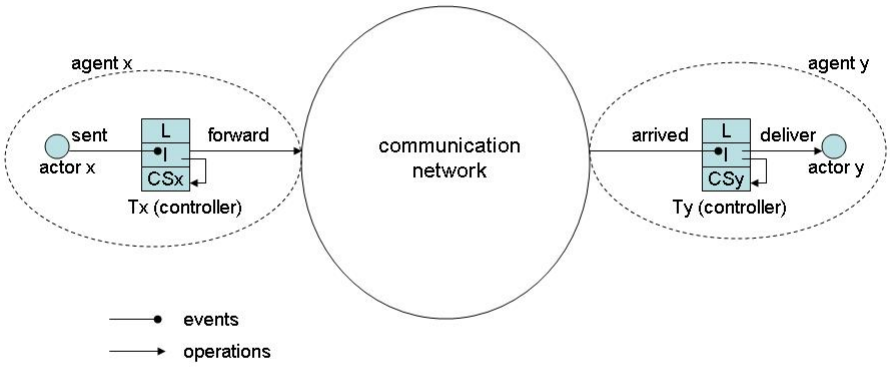


Figure 4.6: LGI-agents interacting [AMN02]

LGI is related to access control as follows. Instead of offering a passive message-exchange infra-structure, LGI middleware actively monitors messages[7] exchange checking compliance with a formal policy set, called the law. This policy takes the form of ECA-rules, which can be represented in either a Prolog-based or Java-based language. In this report, we use the Prolog-based representation for our examples. It is important to note that Prolog rules in LGI should not be read as if-then rules. Instead, each Prolog rule in LGI represents an ECA pattern [Bra99] of the form: $e$ :- $c_1, ..., c_n, do(o_1), ..., do(o_n))$, where $e$ is an event, $c_1, ..., c_n$ is a possibly empty list of conditions and $do(o_1), ..., do(o_n))$ is a list of operations (actions). Given these rules, the law maps every possible pair (event,state) to a pair consisting of (state,operation). Formally, given a set

[5] T stands for trusted.

[6] Laws must be published on the HTTP law server from which they can be collected by controllers.

[7] LGI is able to work with asynchronous communication via TCP/IP messages. Synchronous communication is planned for future releases of LGI.

of events $E$, a set of states $S$ and a set of lists of operations $O$, a law $\mathcal{L}$ is a function: $\mathcal{L} : E \times S \rightarrow S \times O$.

Under Moses, an implementation of LGI, each controller can serve different agents and deal with different laws. LGI provides an extensive set of predefined events, operations and states. In the remaining of the LGI section, we present some complementary relevant concepts, the predefined facilities of the language and then we discuss LGI from the perspective of our evaluation criteria.

### 4.2.1 LGI complementary concepts

A few concepts are important for understanding the LGI semantics.

- The Prolog-law language uses two nonstandard predicates: (1) the predicate @ such as in $t@CS$[8] returns true or false depending on whether term t is present or not in an agent control state CS; (2) the predicate $do(t)$, already mentioned, appends term t to a law, such as in $do(add(level\ 0))$ used to add term *level* 0 to an agent control state.

- The predicate *not* for Prolog-laws has the basic semantic of negation such as in $not(pingTo(Y)@CS)$, meaning that term pingTo(Y) is not present in the CS.

### 4.2.2 LGI access control elements

The elements of a law $\mathcal{L}$, i.e. events $E$, operations $O$ and states $S$, are described next.

Events occur at the level of a single agent, i.e. at the controller (also called *home*) of the event, and can be of three types.

- Events related with issuing a request and receiving a reply, i.e. with the passing of messages between agents. For example:
  - adopted($par(argList), cert(certList)$). This event is the first one which occurs in the "life" of an agent. It makes explicit the agent's intention to cooperate under a law. The parameter $argList$ is a list of arbitrary terms such as the agent's name and role. The parameter $certList$ is list of certificates[9].
  - arrived($[y, \mathcal{L}'], m, x$). This event occurs when a $\mathcal{L}'$-message $m$ sent by agent $x$ arrives at agent $y$ ($y$ is the *home* agent of the event).
  - sent($x, m, [y, \mathcal{L}']$). This event occurs when agent $x$ sends a $\mathcal{L}'$ message $m$ to agent $y$ ($x$ is the *home* agent of the event).
  - certified, created (counterpart of adopted), connected, disconnected and reconnected, and submitted (counterpart of arrived) are also supported. See the LGI Manual [Min05] for details.

[8] CS is an agent control state.

[9] Certificates permit the authentication of an agent governed by law $\mathcal{L}$ who wants to cooperate with another agent governed by law $\mathcal{L}'$.

- Events related with exceptions or faults which may occur during the transmission of a message. For example:
  - exception(op,diagnostic). This event occurs if an operation *op* fails and *diagnostic* is an explicative text message. The agent which evoked the failed event is the home of the exception event.
- Event related to the deadline of an obligation imposed on an agent. For example:
  - obligationDue(oType). This event occurs when an agents pending obligation is no longer due.
  - stateChanged. This event occurs when an agent is discharged of a pending state-obligation because the obligation is no longer due.

Operations are actions mandated by a law which may trigger events. They can be of three kinds:

- communication-operations
  - forward$(x, m, [y, \mathcal{L}'])$. This operation forwards a $\mathcal{L}'$-message $m$ addressed to agent $y$; if agent $y$ receives the message, an event of the type arrived$([x, \mathcal{L}], m, y)$ will be triggered at $y$.
  - deliver$([x, \mathcal{L}], m, y)$. This operation delivers a $\mathcal{L}$-message $m$ sent by agent $x$ if the message has arrived at agent $y$.
  - multicast$(x, m, destinationList)$. This operation[10] forwards message $m$ to all agents listed on the destinationList.
  - release$(x, m, [h, p])$. This operation releases a message $m$ to an agent not governed by LGI laws, using TCP/IP protocol. Parameters $p$ and $h$ specify port and host respectively.
- state-operations
  - add$(t)$ or $+t$. This operation adds a term t to an agent control state.
  - remove$(t)$ or $-t$. This operation removes term t from an agent control state.
  - replace$(t_1, t_2)$. This operation replaces term $t_1$ of an agent control state by term $t_2$.
  - incr$(f, d)$ and dcr$(f, d)$. These operations increment and decrement an agent control state term $f$ with quantity $d$.
  - replaceCS($termList$). This operation replaces an agent whole control state with the given list items.
  - addCS($termList$). This operation appends an agent control state with the given list items.

[10] This is an operation restricted to Prolog-laws.

- obligation-operations
  - imposeObligation($oType, dt, timeUnit$). This operation imposes an obligation of type $oType$ on the agent; the parameter $dt$ determines when the obligation is due and the parameter $timeUnit$ determines the time unit (h, min, sec, ms) of $dt$. It automatically includes a term *obligation* to the agent $DCS$.
  - repealObligation($oType$). This operation removes all obligations of type $oType$ from the agent $DCS$.
  - imposeStateObligation($termList$). This operation triggers a stateChanged event upon any change on the agent $CS$ indicated by the $termList$. It automatically includes a term *audited* to the agent $DCS$.
  - repealStateObligation($termList$). This operation removes all terms audited(termList) from the agent $DCS$.

States are a snapshot of the agent at an instant. It can have three parts.

- **Context** is a set of variables that provides information used only during the evaluation of a law such as ThisLawName, CS and DCS (references to the other two parts of the state), Msg (message being sent or received), self (address of the home agent).
- **Control state** ($CS$) is a list of terms which have no predefined meaning, i.e. the terms meaning is defined by the law. Some examples extracted from the LGI Manual [Min05] are: manager, role(manager), name(joe,smith), [1,2,3], children([joe,jane,jim]).
- **Distinguished control state** ($DCS$) is a list of terms with predefined meaning and syntax, as described next.
  - obligation($oType, t_0, dt$). This term indicates that an agent has a pending obligation of type $oType$, starting at time $t_x$ and ending at time $t_0 + dt$.
  - audited($termList$). This term has as parameter a list of terms to be audited in consequence of the most recent imposeStateObligation operation. Any addition or deletion of a item from the list triggers a stateObligationDue event.
  - authorityTable($A$). This term has as parameter a list of certification authorities initialized automatically when an agent adopts a law $\mathcal{L}$. Authorizations can be added to or deleted from the list $A$ through the operations addAuthority and delAuthority.

### 4.2.3 Roles in LGI

LGI has no built-in concept of role [AM04]. However, the term role(doctor) on a $CS$ of an agent can represent that this agent belongs to the doctor role. A role can be assigned to an agent x in two ways: 1- an agent claims and activates a role by presenting a digital certificate issued by another agent (e.g. administrator). Figure 4.7 shows an agent (Self) claiming to activate its first role in the session by presenting a certificate issued by the admin; 2- an external agent, operating under the same law as agent x, sends a message to x appointing x to a role. So the law will be used to define the process of acquiring roles, the dynamic behavior of roles and the effects of the ability of agents to operate. Figure 4.8 shows an agent belonging to role "branchA-manager" assigning role "branchA-employee" to an userId.

```
R1. certified(issuer(admin),subject(Self),
attributes([role(R)])) :-
not(role(R1)@CS),do(+role(R))
```

Figure 4.7: Role assignment by presenting a digital certificate

```
R1. sent(X,assign-role(userId),Y) :-
role(branchA-manager)@CS, do(forward)
R2. arrived(X,assign-role(userId),Y) :-
  do(+role(branchA-employee)), do(deliver)
```

Figure 4.8: Role assignment by an external agent

### 4.2.4 LGI permissions

LGI provides two kinds of operations and corresponding events to settle and revoke obligations. However, it leaves open the details of the law in terms of the circumstances under which the obligation applies to the agent, the sanctions which will incur if the obligation is not fulfilled before the due date and the treatment of pending obligations.

The operation imposeObligation($oType, dt, timeUnit$) is one kind of obligational operation in LGI. This operation causes a term obligation($oType, t_0, dt$) to be automatically added[11] to the agent $DCS$ for the due time control. The event obligationDue($oType$) will be triggered when time $t = t_0 + dt$ is reached and can be associated with sanction actions. A pending obligation can be revoked before the due date by the operation repealObligation(oType). Figure 4.9 shows: (1) rule R1 in which agent X receives the "onDuty" obligation (from [AM04])

[11] This term is automatically removed from the agent $DCS$ when the obligation due date is reached.

assigned by agent Y; the term "onDuty" is added to the agent's $CS$ and the obligation type "dutyExpired" and deadline are settled, (2) rule R2 which determines that, when the event of type "dutyExpired" is fired, no sanctions are applied and the term "onDuty" is simply removed from the agent's $CS$.

```
R1. arrived(X,onDuty,Y) :- do(+onDuty),
    do(imposeObligation(dutyExpired,[12,hour]))
R2. obligationDue(dutyExpired) :- do(-onDuty)
```

Figure 4.9: Obligation in LGI

The operation imposeStateObligation(termList) is another kind of obligational operation in LGI. This operation causes a term audited(termList) to be automatically added[12] to the agent $DCS$. The event StateChanged will be triggered when any change on the agent $CS$ indicated by the termList occur[13].

Authorization in LGI is acquired by sending an access request over an object to another agent, which can be a server. Figure 4.10 shows agent C (client) sending an authorization request (extracted from [AM04]) to agent S (server) to perform an access action *Op* over the medical-records of a patient (identified by Pid) under the following circumstances: if agent C belongs to role doctor or nurse, represented by role(doctor) and role(nurse) in its $CS$, if agent C is the attending-physician or attending-nurse of patient *Pid*, represented by attending-physician(Pid) and attending-nurse(Pid) in its $CS$ and if agent C is on duty, represented by the term onDuty in its $CS$. Finally, the access request will only be delivered to agent S, i.e. accepted by agent S, if agent S is a server that belongs to role(pr-server).

```
R1. sent(C,access(Op,m-record(Pid)),S) :-
((role(doctor)@CS, attending-physician(Pid)@CS);
(role(nurse)@CS, attending-nurse(Pid)@CS)),
onDuty@CS, do(forward)
R2. arrived(C,access(Op,m-record(Pid)),S) :-
role(pr-server)@CS, do(deliver)
```

Figure 4.10: Authorization in LGI - example 1

LGI allows the assignment of permissions to roles and the checking of permissions as shown in Figure 4.11[14]. In rule R1, an agent S (server) assigns the permission a role $R$ has to perform an action *Op* over its object *Obj*. In rule R2, a client (agent C) sends an access request to any server (agent S), attaching

[12]This term is automatically removed from the agent $DCS$ when an event stateChanged is fired.

[13]The termList can be "all" indicating that any change at all on the agent $CS$ will trigger a StateChanged event.

[14]This figure shows just an extract of Figure 6 from [AM04]

the list *ARS* of its activated roles. In rule R3, the request is only accepted (i.e. delivered) by a server if any of the roles in list *ARS* has the permission to perform the requested action over the specified object.

```
R1. sent(S,permission(R,Op,Obj),Self) :-
do(+permission(R,Op,Obj))
R2. sent(C,access(Op,Obj),S) :-
activated-roles(ARS)@CS,
do(forward(C,access(Op,Obj,myRoles(ARS)),S))
R3. arrived(C,access(Op,Obj,myRoles(ARS)),S) :-
has-permission(ARS,Op,Obj),
do(deliver(C,access(Op,Obj),S))
```

Figure 4.11: Authorization in LGI - example 2

Delegation of access rights in LGI is acquired when an agent X sends a delegation request to an agent Y. Figure 4.12 shows a delegation request (rule R1) sent by an agent belonging to role *doctor* and which is the attending physician of patient *Pid* to an agent belonging to the role nurse. The request, when accepted, delegates the attending task of the patient to a nurse (rule R2). The doctor is allowed to appoint a maximum of two nurses for any patient. This example is complemented by the already shown example in Figure 4.10 which specified the authorization rights to the patient's medical records.

```
R1. sent(X,delegate(Pid),Y) :- role(doctor)@CS,
attending-physician(Pid)@CS,
if (not(deleNo(Pid,N)@CS)) then
(do(+deleNo(Pid,1)), do(forward))
else (N<=2, do(deleNo(Pid,N)<-deleNo(Pid,N+1)),
do(forward))
R2. arrived(X,delegate(Pid),Y) :- role(nurse)@CS,
do(+attending-nurse(Pid)), do(deliver)
```

Figure 4.12: Delegation in LGI

Negative authorizations, i.e. prohibitions, in LGI can be carried out by explicitly blocking interactions between agents, if certain conditions are not satisfied. Figure 4.13 shows a purchase order being explicitly blocked by agent Y (e.g. a server) if the agent X does not belong to the role *manager* and the payment involved is greater than 1000 (from [AMN02]).

### 4.2.5 LGI access control models

The Discretionary Access Control model (DAC) (section 2.1) relies on the discretion of the subject which owns an object to control and grant access rights

```
R1. sent(X,order(item(I),payment(P)),Y) :-
P>1000, not(role(manager)@CS), do(blockS)
```

Figure 4.13: Prohibition in LGI

to other subjects. In LGI, DAC policies can be specified in a similar way as we have seen in the example shown in Figure 4.11, where an agent determined the access rights over its own objects. The agent decides whether another agent will be allowed to perform operation *Op* over its object *Obj* either by granting or by blocking the access.

The Mandatory Access Control model (MAC) (section 2.2) requires subjects and objects classification to allow access rights on a lattice structure of security labels. The classification of agents, regulated by mandatory rules and imposed by a central authority, can in LGI be achieved as described next.

- Minsky (Figure 2.5 in [Min05]) presents a Dynamic Layering Law where a "manager" of the law sets security levels for agents willing to interact under this law. The agents then are allowed to interact with each other only if their security levels are identical or if the "sender" agent has higher security level than the "receiver" agent. In this approach, the "manager" of the law is identified by its absolute address in the law preamble[15]. Therefore, the identification of the authority is hard-coded in the law.

- As mentioned in Section 4.2.1, digital certificates authenticate agents that are allowed to interact by using a public-key issued and signed by a Certification Authority (CA). The same concept of security levels present in the Dynamic Layering Law can be achieved by the certificates approach (Figure 3.1 in [Min05]). However, by using certificates the identification of the "manager" of the law is no longer hard-coded in the law itself but is determined by the presence of the term *role(mgr)* in the "manager" $CS$. Therefore, this approach is a more flexible way to define the authority which can set security levels for regulating agents interactions in a MAC fashion.

In LGI, objects and access to them can be classified through the use of terms in an agent $CS$ and of operations like access($Op, Obj(id, level)$). This way, an agent server can determine and control the access to its own resources. Furthermore, the combination of security levels for agents interactions and of security levels for objects seems also possible.

### 4.2.6 LGI comparison summary

Table 4.2 shows the comparison summary of LGI.

[15] For example, by using the preamble command *alias(mgr,'manager@ramses.rutgers.edu').*

## 4.3 Security Policy Language - SPL

SPL is a constraint-based, event-oriented language for specifying policies [RG99]. In SPL, a policy is a group of rules and sets that govern a particular domain of events. It allows the specification of abstract policies and the development of policy libraries by parameterizing these groups of rules and sets. Thus, policies can be instantiated several times with different (set) parameters. A policy, when instantiated, behaves as a rule and therefore can be combined with other rules and included in another policy. Besides, SPL assumes there is a stream of events that is being monitored which is checked against policies for compliance.

### 4.3.1 SPL access control elements

The basic concepts of the language are described next.

- **Entities** are objects with explicit interfaces which allow property querying. Entities manipulated by SPL can be internal SPL constructs such as sets and policies but can also be references to entities outside SPL such as users, files and events.
- **Sets** allow the classification of entities in categories and the aggregation of entities in groups.
- **Rules** express constraints in terms of relations between entities and sets. They consist of two logical expressions, namely a domain-expression, which establishes the domain of applicability of a rule, and a decide-expression, which decides the applicability of the rule. The basic syntax of a rule is *label* : domain-expression :: decide-expression. The main goal of a rule is to decide upon the acceptability (allow, deny or neutral, i.e. not-apply) of each event[16] under control of the SPL access monitor.
- **Policies** govern a particular domain of events by composing rules with other rules in conjunctions, disjunctions and negations and by composing rules with sets into logical units. Each policy rule is in fact a query rule identified by a question mark before its label.

### 4.3.2 Grouping in SPL

SPL allows two kinds of grouping: sets and roles, as described in the next paragraphs.

Sets can be categories and groups.

- **Categories** are sets that allow classifying entities based on their properties[17] by using the restriction operator @. For example:

[16] Events in SPL can be grouped in past, current and future sets; the expression *ce.event* is an event that belongs to the current events set.

[17] A property of an entity is written with a dot, e.g. target.type is referring to the property *type* of the target object.

- user *set* localUsers=AllUsers@{.hostname=localhost} defines a category of users which have the common property of being logged in a specific host;
- object *set* invoices=AllObjects@{.doctype="invoice"} defines a category of objects which have the common property of being invoices.

The restriction operator can also be used to restrict the events to which rules and policies apply. For example:

FORALL $r$ IN u.userPolicy{r@{.target.owner=u}}

defines that all private rules of a user only apply to target objects owned by this user.

- **Groups** are just collections of internal or external entities. For example, user set *Authorized*, object set *Protected*, external operation set *AllActions* and external event set *AllEvents*.

The concept of role as a users group with common responsibilities and authority is a building block in SPL. Roles are considered as policies and are specified by using two sets: one set representing the users which are allowed to play the role and another set representing the users actually playing the role. Figure 4.14 shows a role policy (from [RG99]). The first set is the *Authorized* set and the second set is the *Active* set. Events trying to *insert* a user into the *Active* set are only allowed if the user (represented by *parameter*[1]) is in the *Authorized* set.

```
policy simpleRole (user set Authorized,
 user set Active)
{
?simpleRole: ce.action.name = "insert" &
 ce.target = Active
:: ce.parameter[1] IN Authorized;
}
```

Figure 4.14: Role in SPL

### 4.3.3 SPL permissions

Positive and negative authorizations are specified in SPL in terms of rules, where each rule is either a permission or a prohibition which identifies events that are allowed or denied. Therefore, every event where the domain-expression is true can be permitted or prohibited based on the result (true or false) of the decide-expression. Figure 4.15[18] shows a rule which states that *paymentOrder* approvals are allowed when the author of the approval is not the owner of the

[18]Example taken from [RG99].

*paymentOrder*. This rule is an authorization for events that make the decide-expression true and is a prohibition for events that make the decide-expression false.

```
DutySep: ce.target.type = "paymentOrder" &
 ce.action.name = "approve"
:: ce.author != ce.target.owner;
```

Figure 4.15: Authorization in SPL - example 1

Generic policies can be specified and then instantiated in SPL. Figure 4.16 shows a generic authorization policy with three parameters, mainly based on the concept of sets, which allows permission for subjects ($AllowUsers$) to perform actions ($RestrictActions$) on target objects ($ProtObjects$)[19]. The last line is an instantiation of the ACL policy applying the authorization to *clerks* to perform the action *write* on *invoices*.

```
policy ACL(
user set AllowUsers
object set ProtObjects
interface RestrictionActions)
{
?PolicySimple: ce.action IN RestrictActions &
 ce.target IN ProtObjects
:: ce.author IN AllowUsers
}
DoInvoice: new ACL(clerks, invoices,
 AllActions@{.name = "write"});
```

Figure 4.16: Authorization in SPL - example 2

SPL allows the specification of rules which prohibit events not explicitly allowed by other policies. Figure 4.17 shows an example of such a default-prohibition policy query rule. Thus, if events do not satisfy either *authorRule* or *userPolicyRule* then they are denied access to a target object[20].

```
?DAC: authorRule OR userPolicyRule OR deny;
```

Figure 4.17: Prohibition in SPL

SPL specifies obligations by considering a rule $domain - expression$ as the obligation trigger expression and the decide-expression as the obliged expression. The tags $TIMEOUT$ and $COMPENSATE$ can be included in an obligation

[19] This example is a combination of Figures 9, 10 and 11 from [RG99].
[20] For more details about the rules involved in this example, refer to [RG99].

```
policy OnlineShop
{
?BookPayment():
TIMEOUT = 1 hour
COMPENSATE = blacklist(ce.author, ce.target)
EXIST fe IN FutureEvents {
ce.action = "Buy_book"
 :: fe.action = "Pay_book"
};
}
```

Figure 4.18: Obligation in SPL

```
policy DAC
{
authorRule: ce.target.owner = ce.author
   :: true;
}
```

Figure 4.19: DAC policy in SPL

policy to determine the deadline and the sanction action respectively. Heimdall, as the next generation of SPL is called, provides an obligation enforcement mechanism for assuring that the sanction action is automatically triggered if the obligation is not fulfilled within the deadline[21]. Figure 4.18 shows the generic obligation[22] of paying for a bought book.

### 4.3.4 SPL access control models

Discretionary Access Control (DAC) policies can be partially specified in SPL, such as shown in Figure 4.19[23] in which it is specified that the owner of a target object can perform any action on it. However, because SPL does not support delegation, the owner of an object cannot transfer access rights at his own discretion.

Mandatory Access Control (MAC) policies could in theory be specified in SPL considering that entity attributes can be configured. Thus, an object's security level and a subject's clearance level could be considered as attributes the same way as attributes like name, owner and so on. This security and

[21] The Heimdall enforcement mechanism transforms the obligation into two new policies: one for the trigger event and one for the fulfill event both based on present events which will be evaluated when they are executed. A time keeper deals with the deadline constraints involved. See Gama [GF05] for detailed information. Ribeiro [RZF01] also discusses the enforcement of obligations in SPL.

[22] The example is a combination of Figures 2 and 3 from [GF05].

[23] Extracted from [RG99].

clearance levels are based on both confidentiality level of information and on projects. However, no publicly available SPL documentation seems to indicate whether the language allows multi-valued attributes such as needed for projects. Apart from this issue, a detailed study is required for analyzing the viability of the mapping between the Bell-LaPadula model, for example, and SPL features.

### 4.3.5 SPL comparison summary

Table 4.3 shows the comparison summary of SPL.

## 4.4 Policy Description Language - PDL

Bell Lab's PDL is a real-time production rule system specialized for defining policies [LBN99]. It is an event-based language developed for application in the context of telecommunication networks, Web Services routing preference settings, assignment of users to tasks in workflows, and fault management related to network routers. A policy in PDL is a function that maps a series of these events into a set of actions. Policies are described by a collection of expressions of two kinds.

1. *Policy rule proposition*: "event **causes** action **if** condition" means that if the event occurs in a situation where the condition is true then the action will be executed.

2. *Policy-defined event proposition*: "event **triggers** policy-defined-event **if** condition" means that if the event occurs in a situation where the condition is true then the policy-defined-event will be triggered.

Events can be primitive or complex. Primitive events can be system defined, i.e. initiated by changes in the environment, or policy defined, i.e. initiated by policy-defined event propositions. Complex events are a combination of primitive events, for example, parallel events, sequential events, or absence of an event in order to enforce a policy.

Bertino [BMP05] applies PPDL, an extended PDL with preferences, to workflows and presents a modified policy-defined event proposition. It has the format *true* **causes** $a$ **if** $C$ where *true* represents a fact that is always true and $a$ represents an action which will be triggered if condition $C$ is satisfied. Permissions are based on user, task and role consisting of: (1) the assignment of users to roles and (2) the assignment of roles to tasks. A permission to execute a task can be granted in two ways.

- Figure 4.20 shows a rule that assigns permission to user U with role R to execute task T (expressed by term $ass(T, R, U)$), if user U belongs to role R ($is(R, U)$) and the fact that user U cannot execute task T does not exist ($not\ neg(T, U)$).

- Figure 4.21 shows a rule that assigns permission to user U with role R to execute task T, if a positive exceptional condition $C$ is satisfied.

```
poss(T,R,U) causes ass(T,R,U) if is(R,U), not neg(T,U)
```

Figure 4.20: Permission to execute a workflow task in PPDL

```
posex(T,R,U) causes ass(T,R,U) if C
```

Figure 4.21: Conditional permission to execute a workflow task in PPDL

Furthermore, preferences between roles and users for the execution of a task can also be specified in PPDL. For more details, refer to [BMP05].

However, although it is possible to express access control in a specific scenario (workflow management), PDL and PPDL do not really support access control policies in a broad sense. In terms of workflow, the concept of object is replaced by the concept of task and permissions are applied to subject-task-action instead of to subject-target-action.

### 4.4.1 PDL comparison summary

Table 4.4 shows the comparison summary of PDL.

| Criteria | Summary |
|---|---|
| Access Control elements | All the basic elements of access control can be represented in Ponder, i.e. subject, object and actions. It allows assembling subjects in roles and in groups and also allows expressing a list of actions. Domains permit grouping of objects. |
| Intrusion Detection elements | None of the basic elements of intrusion detection can be expressed in Ponder. Besides, the Ponder documentation does not make explicit that new elements could be added to the language through Java classes. |
| Positive authorization | Positive authorizations can be expressed in the format subject-target-action. |
| Negative authorization | Negative authorizations can be expressed also in the format subject-target-action. |
| Delegation | Delegations can be expressed in the format grantor-grantee-action-target. |
| Obligation | Obligations can be expressed in the format event-condition-action. Deadline and sanction actions are neither required nor possible in Ponder obligations. |
| Time constraints | Ponder supports time constraints which use the OCL library *Time* [Gro00] and allow expressing duration, interval, before, after, date, month, day of the week, time and current time. Thus, Ponder provides the ability to apply all the time constraints mentioned in Table 3.1. |
| Other constraints | Ponder supports: (1) conditions applied to authorizations policies; (2) boolean and operations over sets (union, intersection and difference) applied to subjects and objects attributes; (3) concurrency constraints applied to actions expressing sequence, parallelism and alternative. |
| Access Control model | Ponder allows the specification of DAC policies but MAC policies cannot be specified in a viable way. |
| Toolkit and manual availability | The Ponder toolkit is no longer available. However, a technical manual is available |

Table 4.1: Ponder Comparison Summary

| Criteria | Summary |
|---|---|
| Access Control elements | The three basic elements of access control can be represented in LGI. First, subjects are agents which can interact with other agents representing humans, servers or processes. The distinction between agents is achieved by the agent control-state $CS$ and its semantics are defined by the law. Second, agents can have resources, i.e. objects, and services associated with them. Finally, actions are operations which can be performed between agents. The semantics of an agent control-state are defined by the law and, therefore, sets of subjects in roles, grouping of objects and lists of actions can be achieved by providing meaning to control-state terms. |
| Intrusion Detection elements | The basic elements of intrusion detection can be obtained by providing semantics to agents' control-state terms defined by the law. Furthermore, a new HTTP law-server can be created to publish and retrieve new laws since the current LGI toolkit provides a very simple and small law-server. |
| Positive authorization | Positive authorizations are expressed in terms of an access request from one agent to another agent. The request can be accepted or denied. |
| Negative authorization | Negative authorization are achieved by blocking interactions between agents. |
| Delegation | Delegations are expressed in terms of a delegation request from one agent to another agent. The request can be accepted or denied. |
| Obligation | Obligations can be imposed by law, triggering deadline events and sanction actions automatically. |
| Time constraints | The time constraints available depend on the law-language used and the packages imported to specify the law. Thus, in Java for example, law-classes can explicitly import packages other than the basic LGI package. The current LGI toolkit uses the Jinni Prolog interpreter[a] and it seems that no sophisticated time constraints are available. [a] http://www.cs.unt.edu/~tarau/ |
| Other constraints | Law rules can include conditions and boolean operators. However, sets operations are not built-in supported. |
| Access Control model | LGI allows the specification of DAC and MAC policies. |
| Toolkit and manual availability | The LGI toolkit called Moses is available the link http://www.moses.rutgers.edu/download.html. A technical manual is also available. |

Table 4.2: LGI Comparison Summary

| Criteria | Summary |
|---|---|
| Access Control elements | All the basic elements of access control can be represented in SPL, i.e. subjects and objects are entities and actions are operations. Sets of subjects, objects and actions are possible. |
| Intrusion Detection elements | None of the basic intrusion detection elements can be expressed in SPL. Besides, SPL does not provide a toolkit which allows the flexibility of specifying new entities and new Java methods for example. |
| Positive authorization | Positive authorizations can be expressed in terms of decide-expressions that return true on a SPL rule. |
| Negative authorization | Negative authorizations can be expressed as rules with decide-expressions returning false or in a generic format using the *deny* predicate. |
| Delegation | Delegations are not supported by SPL. |
| Obligation | History-based obligations with deadline and sanction action are supported by SPL. |
| Time constraints | SPL supports some time constraints listed in Table 3.1, such as timeout duration $TIMEOUT = 1$ hour and obligation effectiveness timeframe from $15h00$ to $20h00$[a]. [a] According to Gama in [GF05] |
| Other constraints | The following sets operations are supported by SPL: index $myset[i]$, membership $elementINmyset$, join operator $myset1 + myset2$ and meet operator $myset1 * myset2$. Logic operators are all supported and conditions are indirectly supported, for example, by set membership. |
| Access Control model | SPL does not really support the specification of DAC policies because an object owner cannot propagate access to his objects. MAC policies could be potentially expressed but more investigation and access to technical information is required. |
| Toolkit and manual availability | Neither SPL toolkit nor technical documentation is available. |

Table 4.3: SPL Comparison Summary

| Criteria | Summary |
|---|---|
| Access Control elements | Not all basic access control elements can be represented in PDL or PPDL, for example, the concept of object is absent. However, the language provides powerful events handling and allows, for example, the grouping of events. However, although groups of users in roles are also allowed they are applied only to workflow task execution. |
| Intrusion Detection elements | Currently no intrusion detection basic elements are available. |
| Positive authorization | Authorizations for subject to perform actions on objects cannot be expressed in PDL. |
| Negative authorization | Not applicable. |
| Delegation | Not applicable. |
| Obligation | Not applicable. |
| Time constraints | PDL provides a type *Time* as an attribute of an event. Time attributes can contain values such as *morning* associated with 6:00am. Time intervals can be specified as shown in the following example from Lobo [LBN99]: $hour[3].Time - hour[1].Time = 5$ over the complex event hour, hour,hour ( hour means zero or more events of the same type). The expression is true if the time interval between the first and the third events of a series equals five. |
| Other constraints | PDL supports condition expressions and boolean operations but do not support sets operations. PDL has *temporal aggregates* which are similar to function calls and can perform operations such as Count, Sum, Min, Max and Avg on events or events' attributes. |
| Access Control model | Not applicable. |
| Toolkit and manual availability | Toolkit and technical manual currently not available. |

Table 4.4: PDL Comparison Summary

# Chapter 5

# Discussion

Our main goal when comparing access control languages was to identify the possibility of merging access control and intrusion detection policies. Assuming we have identified the right criteria in our comparison framework and considering the set of languages we compared, we believe that Law-Governed Interactions (LGI) contains the most promising *ingredients* for combining these policies. In summary, LGI provides:

- the basic elements and type of policies for access control policies.
- the concepts of agent state and event, since LGI is a stateful language.
- flexibility for additions of intrusion detection elements. Besides, TCP/IP elements (considered in this report as basic elements for intrusion detection) fit well with the idea of interacting agents.
- practical instruments for creating new *laws* (i.e. policies governing interactions), such as the availability of an updated toolkit and documentation, and for programming new state attributes, new events and new actions.
- decentralized and scalable policy enforcement. Therefore, since each LGI controller is an independent process which can be placed anywhere in the network, we could have controllers in strategic network nodes intermediating network and host traffic.
- possibility to impose constraint on interactions.
- mandatory policy enforcement between agents already engaged in a law since interactions are not possible if the policy on both agents controllers are not satisfied.

However, despite of the *ingredients* for the merging policies found in LGI, we identified a mismatch between LGI events and intrusion detection stateful languages events. We discuss next whether this has any consequences which might restrict the possibility of using LGI for describing attack scenarios.

```
transition t (s1 -> s2)
nonconsuming
{
[IP d1 [TCP t1]]:
(d1.source == 192.168.0.1) &&
 t1.destination == 23)
 {log("message");}
}
```

Figure 5.1: TCP/IP attack rule in STATL

On the one hand, events for LGI occur at a single agent level. A *law* decides what should be done if an event occurs at a given agent state and triggers operations, i.e. actions, at another agent state. So, as described in section 4.2, a law maps $(event, state) \rightarrow (state, operation)$. On the other hand, events in intrusion detection stateful languages [Kum95, EVK02] are related to the transition between states. Considering that these languages describe a complete attack scenario: (1) states represent snapshots of a system during the evolution of an attack; (2) transitions connect two states and have events description associated with them. Thus, if at a state $S1$ the event stream matches an event described in a transition, this transition is fired causing a move to a next state $S2$. In such a way, a scenario maps a sequence of $(state) \rightarrow (event, operation) \rightarrow (state)$. As we can see, there is an event mismatch between LGI and stateful intrusion detection languages and we suspect that, as a consequence, LGI might only be viable to express one-step attack scenarios. This is what we investigate next using two attack scenario examples.

First, let's suppose we want to catch events that match any IP datagram containing a TCP segment, with source address 192.168.0.1 and destination port 23. Figure 5.1 shows an example in the STATL[1] language [EVK02] that performs this matching. The transition[2] states that if the event "[IP d1 [TCP t1]]:" is matched then the *log* command is executed. Figure 5.2 shows the translation of this example to LGI in terms of two agents interacting. At the agent X event home (event *sent*), the source address is checked and, at the agent Y event home (event *arrived*), the destination port is checked. As we have see, LGI is able to express this one-step attack scenario.

Now, let's suppose we want to catch events that match the whole TCP connection as a step, for example, to identify half-open TCP connections which are building blocks for a TCP-flood attack. Figure 5.3 shows a graphical representation of the states and transitions involved, taken from Kumar [Kum95]. In order to represent this scenario in LGI we need to map each pair (source,destination) as a two agents' interaction, similar to the example shown in Figure 5.2. There-

[1] STATL stands for State Transition Analysis Technique Language.

[2] The translation being nonconsuming means that at each event matched a new instance of the transition is fired.

```
R1. sent(X,packet(IPaddress(IP),port(P),Y) :-
IP=192.168.0.1, do(forward)
R2. arrived(X,packet(IPaddress(IP),port(P),Y) :-
P=23, do(log("message"))
```

Figure 5.2: TCP/IP attack rule in LGI

fore, we would need to log three interactions separately: (1) one interaction between agent "source" and agent "destination" matching protocol flag SYN. This interaction would be logged in agent "destination" and would correspond to $TCP_1$; (2) one similar interaction corresponding to $TCP_2$ and logged in agent "source" and (3) again one similar interaction corresponding to $TCP_3$ and logged in agent "destination". Therefore, we would have at the end three log entries logged in two different agents, representing no risk if analyzed separately. Unlike LGI, STATL and the language proposed by Kumar, log multi-step attacks as a whole providing the necessary level of abstraction for composing attack scenarios.

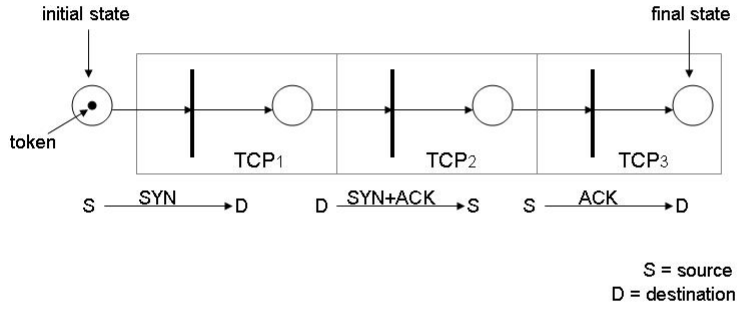


Figure 5.3: Colored Petri Net representation of a TCP connection [Kum95]

In summary, although LGI is a stateful language, it seems that it only allows efficient representation of one-step attack scenarios. However, more in-depth theoretical study as well as practical experimentation is required to validate this hypothesis.

# Chapter 6

# Conclusions

This report discussed the reasons why it would be beneficial to bring access control and intrusion detection together, by combining policies. It described the three existent models of access control policies, i.e. DAC, MAC and RBAC, providing a comparative table with the main advantages and disadvantages of each model. It defined a comparison framework from the perspective of intrusion detection which supported the comparison of some access control languages. It presented a discussion about the applicability of the most promising access control language in terms of combining policies from the perspective of describing attack scenarios.

As future work, we identified four mains topics while working on this report:

- a comparative analysis of the three possibilities for narrowing the gap between access control and intrusion detection presented in Section 1.3.
- the inclusion of a logic-based language in the scope of this report for insights on the applicability of this kind of language for intrusion detection.
- the extension of the comparison framework used in this report with the policy enforcement mechanism used by each language. It would provide insights on the possible ways and benefits of different approaches.
- a more detailed analysis of LGI for representing multi-step attack scenarios.

### Acknowledgments

I would like to specially thank my supervisor Pascal van Eck for the constant support, essential insights and comprehensive reviews.

# Bibliography


[AJ01] Y. Lin. A. Jones. Application intrusion detection using language library calls. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC'01)*, pages 442–449, 2001.

[AL01] Magnus Almgren and Ulf Lindqvist. Application-integrated data collection for security monitoring. In *Recent Advances in Intrusion Detection (RAID 2001)*, LNCS 2212, pages 22–36, Davis, California, October 2001. Springer. `http://www.sdl.sri.com/papers/raid2001/`.

[AM04] Xuhui Ao and Naftaly H. Minsky. On the role of roles: from role-based to role-sensitive access control. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 51–60, New York, NY, USA, 2004. ACM Press.

[AMN02] Xuhui Ao, Naftaly Minsky, and Thu D. Nguyen. A hierarchical policy specification language and enforcement mechanism for governing digital enterprises. In *POLICY'02: Proceeding of the Third IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 38–49, Washington, DC, USA, June 2002. IEEE Computer Society. `http://doi.ieeecomputersociety.org/10.1109/POLICY.2002.1011292`.

[And80] J. P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Co., Fort Washington, PA, 1980. `http://seclab.cs.ucdavis.edu/projects/history/papers/ande80.pdf`.

[AS00] Gail-Joon Ahn and Ravi S. Sandhu. Role-based authorization constraints specification. *Information and System Security*, 3(4):207–226, 2000. `citeseer.ist.psu.edu/ahn00rolebased.html`.

[BA05] Travis D. Breaux and Annie I. Anton. Deriving semantic models from privacy policies. In *Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 67 – 76. IEEE Computer Society, june 2005.

[BBC+02] V.S. Batra, J. Bhattacharya, H. Chauhan, A. Gupta, M. Mohania, and U. Sharma. Policy driven data administration, June 2002.

[BBF00] Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. TRBAC: a temporal role-based access control model. In *RBAC '00: Proceedings of the fifth ACM workshop on Role-based access control*, pages 21–30, New York, NY, USA, 2000. ACM Press. `http://doi.acm.org/10.1145/344287.344298`.

[BCF01] Elisa Bertino, Silvana Castano, and Elena Ferrari. On specifying security policies for web documents with an xml-based language. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 57–65, New York, NY, USA, 2001. ACM Press.

[BFIK99] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The keynote trust-management system, version 2. Request for Comments: 2704, September 1999.

[BFL96] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *SP' 96: Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173, Washington, DC, USA, 6-8 May 1996. IEEE Computer Society. `http://doi.ieeecomputersociety.org/10.1109/SECPRI.1996.502679`.

[Bib77] K. J. Biba. Integrity considerations for secure computer systems. Technical Report TR-3153, Bedford, Massachusetts, USA, 1977.

[BMP05] Elisa Bertino, Alessandra Mileo, and Alessandro Provetti. PDL with preferences. *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 213–222, June 2005.

[BN89] David F. C. Brewer and Michael J. Nash. The chinese wall security policy. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 206–214. IEEE Computer Society, May 1989. `http://doi.ieeecomputersociety.org/10.1109/SECPRI.1989.36295`.

[Bos95] Anthony Boswell. Specification and validation of a security policy model. *IEEE Transactions on Software Engineering*, 21(2):63–68, 1995. `http://dx.doi.org/10.1109/32.345822`.

[BP73] David Elliott Bell and L. J. La Padula. Secure computer systems: mathematical foundations and model. Technical Report M74-244, Bedford, Massachusetts, USA, 1973.

[Bra99] Ivan Bratko. *Prolog Programming for Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 1999.

[CCM02] Nathan Carey, Andrew Clark, and George Mohay. Ids interoperability and correlation using idmef and commodity systems. In *ICICS'02: Proceedings of 4th International Conference on Information and Communications Security*, LNCS 2513, Berlin Heidelberg, December 2002. Springer-Verlag.

[CGL99] Christina Yip Chung, Michael Gertz, and Karl N. Levitt. DEMIDS: A misuse detection system for database systems. In *Proceedings of the Third International IFIP TC-11 WG11.5 Working Conference on Integrity and Internal Control in Information Systems*, pages 159–178. Kluwer Academic Publishers, 1999.

[CS96] Fang Chen and Ravi S. Sandhu. Constraints for role-based access control. In *RBAC '95: Proceedings of the first ACM Workshop on Role-based access control*, page 14, New York, NY, USA, 1996. ACM Press. `http://doi.acm.org/10.1145/270152.270177`.

[CW87] David D. Clark and David R. Wilson. A comparison of commercial and military computer security policies. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 184–194. IEEE Computer Society, 1987.

[Dam02] N. Damianou. *A Policy Framework for Management of Distributed Systems*. PhD thesis, Imperial College, University of London, Department of Computing, 2002. `citeseer.ist.psu.edu/damianou02policy.html`.

[DBSL02] N. Damianou, A. Bandara, M. Sloman, and E. Lupu. A survey of policy specification approaches. Technical report, Department of Computing, Imperial College of Science Technology and Medicine, London, 2002. `citeseer.ist.psu.edu/damianou02survey.html`.

[DCF06] H. Debar, D. Curry, and B. Feinstein. The intrusion detection message exchange format. `http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-15.txt`, February 2006.

[DDLS00] Nicodemos Damianou, Naranker Dulay, Emil C. Lupu, and Morris Sloman. Ponder: A language for specifying security and management policies for distributed systems. Technical Report DoC 2000/1, Imperial College of Science Technology and Medicine - Department of Computing, October 2000.

[DDLS01] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, LNCS 1995, pages 18–38, London, UK, 2001. Springer-Verlag.

[DHV03] Francisco Duran, Javier Herrador, and Antonio Vallecillo. Using UML and Maude for Writing and Reasoning about ODP Policies. In *4th International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, pages 15–25, Lake Como, Italy, June 2003. IEEE Publishing.

[EVK02] Steve T. Eckmann, Giovanni Vigna, and Richard A. Kemmerer. STATL: An Attack Language for State-based Intrusion Detection. *Journal of Computer Security*, 10(1/2):71–104, 2002.

[FSG+01] David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *Information and System Security*, 4(3):224–274, 2001.

[ftAoSIS02] OASIS (Organization for the Advancement of Structured Information Standards). XACML Language Proposal, version 0.8. `http://www.oasis-open.org/committees/xacml`, January 2002.

[GF05] Pedro Gama and Paulo Ferreira. Obligation policies: an enforcement platform. In *POLICY'05:Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 203–212, Washington, DC, USA, June 2005. IEEE Computer Society. `http://dx.doi.org/10.1109/POLICY.2005.18`.

[GG05] Fabio Roli Giorgio Giacinto, Roberto Perdisci. Alarm clustering for intrusion detection systems in computer networks. In *MLDM'05: Proceedings of the 4th International Conference on Machine Learning and Data Mining in Pattern Recognition*, LNCS 3587, pages 184–193, Berlin Heidelberg, July 2005. Springer-Verlag.

[Gro00] Object Management Group. Object constraint language specification, version 1.3. `http://www.cs.queensu.ca/~cisc422/2006w/readings/papers/uml-section7.pdf`, March 2000.

[HBM98] R. J. Hayton, J. M. Bacon, and K. Moody. Access control in an open distributed environment. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 3–14, Oakland, CA, USA, May 1998. IEEE Computer Society. `citeseer.ist.psu.edu/hayton98access.html`.

[HMM+00] A. Herzberg, Y. Mass, J. Mihaeli, D. Naor, and Y. Ravid. Access control meets public key infrastructure, or: assigning roles to strangers. In *SP' 2000: Proceedings on the 2000 IEEE Symposium on Security and Privacy*, pages 2–14, Washington, DC, USA, 14-17 May 2000. IEEE Computer Society.

[HPL98] James A. Hoagland, Raju Pandey, , and Karl N. Levitt. Security policy specification using a graphical approach. Technical Report CSE-98-3, Computer Science Department, UC Davis, July 1998.

[HV01] Michael Hitchens and Vijay Varadharajan. Tower: A language for role based access control. In *POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, pages 88–106, London, UK, 2001. Springer-Verlag.

[ISO99] ISO/IEC. Jtc1/sc7/wg3-3n34, Information Technology - Open Distributed Processing Reference Model - Enterprise Viewpoint. ISO/IEC 15414 - ITU-T Recommendation X.911, January 1999.

[JSS97] Sushil Jajodia, Pierangela Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 31–42, Washington, DC, USA, 4-7 May 1997. IEEE Computer Society.

[KRL97] C. Ko, M. Ruschitzka, and K. Levitt. Execution monitoring of security-critical programs in distributed systems: a specification-based approach. In *SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy*, page 175, Washington, DC, USA, 1997. IEEE Computer Society.

[Kum95] Sandeep Kumar. *Classification and Detection of Computer Intrusions*. PhD thesis, Purdue University, Department of Computer Sciences, 1995. `citeseer.ist.psu.edu/kumar95classification.html`.

[KVV05] Christopher Kruegel, Fredrik Valeur, and Giovanni Vigna. *Intrusion Detection and Correlation - Challenges and Solutions*, volume 14 of *Advances in Information Security*. Springer Verlag, New York, NY, USA, 2005.

[KVVK02] Christopher Kruegel, Fredrik Valeur, Giovanni Vigna, and Richard Kemmerer. Stateful intrusion detection for high-speed networks. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 285, Washington, DC, USA, 2002. IEEE Computer Society.

[LBN99] Jorge Lobo, Randeep Bhatia, and Shamim A. Naqvi. A Policy Description Language. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence*, pages 291–298. The MIT Press, July 1999.

[LLS03] Leonidas Lymberopoulos, Emil Lupu, and Morris Sloman. An Adaptive Policy Based Framework for Network Services Management. *Journal of Network and Systems Management*, 11:277–303, 2003.

[LSK01] Ingo Luck, Christian Schafer, and Heiko Krumm. Model-based tool-assistance for packet-filter design. In *POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, LNCS 1995, pages 120–136, London, UK, 2001. Springer-Verlag.

[LZ97] J. Leiwo and Y. Zheng. Layered protection of availability. In *Proceedings of the 1997 Pacific Asian Conference on Information Systems*, april 1997.

[McH01] John McHugh. Intrusion and intrusion detection. *International Journal of Information Security*, 1:14–35, August 2001.

[MESW01] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen. Policy core information model, version 1 - specification. RFC 3060, `http://www.ietf.org`, February 2001.

[Min05] Naftaly Minsky. Law governed interaction (LGI): A distributed coordination and control mechanism, version 0.9.2. `http://www.moses.rutgers.edu/documentation/manual.pdf`, October 2005.

[MOR01] James Bret Michael, Vanessa L. Ong, and Neil C. Rowe. Natural-language processing support for developing policy-governed software systems. In *TOOLS '01: Proceedings of the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39)*, page 263, Washington, DC, USA, 2001. IEEE Computer Society.

[MU98] Naftaly Minsky and Victoria Ungureanu. Unified support for heterogeneous security policies in distributed systems. *Proceedings of the 7th security conference. (USENIX Association: Berkeley, CA)*, 1998.

[OSM00] Sylvia L. Osborn, Ravi S. Sandhu, and Qamar Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *Information and System Security*, 3(2):85–106, 2000.

[Pon03] Ponder. Ponder implementation guide. `http://www-dse.doc.ic.ac.uk/Research/policies/ponder/docs/PONDERImplementationGuide.pdf`, July 2003.

[PP03] Charles P. Pfleeger and Shari Lawrence Pfleeger. *Security in computing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 3rd edition edition, 2003.

[RG99] C. Ribeiro and P. Guedes. SPL: An access control language for security policies with complex constraints. Technical Report RT/0001/99, INESC, Lisboa, Portugal, January 1999.

[RZF01] Carlos Ribeiro, Andre Zuquete, and Paulo Ferreira. Enforcing obligation with security monitors. In *ICICS '01: Proceedings of the Third International Conference on Information and Communications Security*, LNCS 2229, pages 172–176, Berlin Heidelberg, November 2001. Springer-Verlag.

[San92] R. S. Sandhu. A lattice interpretation of the chinese wall policy. In *Proc. 15th NIST-NCSC National Computer Security Conference*, pages 329–339, 1992. `citeseer.ist.psu.edu/article/sandhu92lattice.html`.

[San93] Ravi S. Sandhu. Lattice-based access control models. *Computer*, 26(11):9–19, November 1993. `http://dx.doi.org/10.1109/2.241422`.

[Sch03] Andreas Schaad. *A Framework for Organisational Control Principles*. PhD thesis, University of York, Department of Computer Science, 2003. `http://www.cs.york.ac.uk/ftpdir/reports/YCST-2003-05.pdf`.

[SDL02] Morris Sloman, Naranker Dulay, and Emil Lupu. The ponder policy based management toolkit. http://www-dse.doc.ic.ac.uk/Research/policies/ponder/PonderSummary.pdf, August 2002.

[SdV01] Pierangela Samarati and Sabrina De Capitani di Vimercati. Access control: Policies, models, and mechanisms. In *FOSAD '00: Revised versions of lectures given during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design on Foundations of Security Analysis and Design*, pages 137–196, London, UK, 2001. Springer-Verlag.

[SGF+02] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based anomaly detection: a new approach for detecting network intrusions. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 265–274. ACM Press, 2002.

[SLL+01] Stanley Y. W. Su, Herman Lam, Minsoo Lee, Sherman Bai, and Zuo-Jun (Max) Shen. An information infrastructure and e-services for supporting internet-based scalable e-business enterprises. In *EDOC'01: Proceedings of the Fifth IEEE International Enterprise Distributed Object Computing Conference*, pages 2–13, Washington, DC, USA, September 2001. IEEE Computer Society. `http://computer.org/proceedings/edoc/1345/13450002abs.htm`.

[Slo94] M. Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2:333–360, 1994.

[SP] Snort Project. Snort Users Manual, version 2.3.3. http://www.snort.org/docs/snort_htmanuals/htmanual_233/.

[SS94] Ravi S. Sandhu and Pierrangela Samarati. Access control: Principles and practice. *IEEE Communications Magazine*, 32(9):40–48, 1994.

[SS97] R. Sandhu and P. Samarati. Authentication, access control, and intrusion detection. In A. B. Tucker, editor, *The Computer Science and Engineering Handbook*, pages 1929–1948. CRC Press, 1997.

[Ste91] Daniel F. Sterne. On the buzzword "security policy". In *IEEE Symposium on Security and Privacy*, pages 219–231. IEEE Computer Society, 1991.

[SWY+02] K. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobson, H. Mills, and L. Yu. Requirements for policy languages for trust negotiation. In *POLICY 2002: Third International Workshop on Policies for Distributed Systems and Networks*, pages 68–79. IEEE Computer Society, 2002.

[Ten86] William T. Tener. Discovery: An expert system in the commercial data security environment. In *Proceedings of the IFIP Security Conference*, 1986.

[UM00] Victoria Ungureanu and Naftaly H. Minsky. Establishing business rules for inter-enterprise electronic commerce. In *DISC'00: Proceeding of the 14th International Symposium on Distributed Computing*, LNCS 1914, pages 179–193. Springer-Verlag, October 2000.

[WE02] M. Wood and M. Erlinger. Intrusion detection message exchange requirements. October 2002.

[WH01] Marc G. Welz and Andrew Hutchison. Interfacing trusted applications with intrusion detection systems. In *RAID '01: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, LNCS 2212, pages 37–53, London, UK, October 2001. Springer-Verlag.

[WM93] Roel J. Wieringa and John-Jules Ch. Meyer. Applications of deontic logic in computer science: A concise overview. In J.-J. C. Meyer and R. J. Wieringa, editors, *Deontic Logic in Computer Science: Normative System Specification*, pages 17–40. Wiley, New York, 1993.