Policy Based Roles for Distributed Systems Security

Emil Lupu, Morris Sloman and Nicholas Yialelis

Imperial College, Department of Computing,
180 Queen’s Gate, London SW7 2BZ, U.K.
E-mail: {m.sloman, e.c.lupu, n.yialelis}@doc.ic.ac.uk

Abstract

Distributed systems are increasingly being used in commercial environments necessitating the development of trustworthy and reliable security mechanisms. There is often no clear informal or formal specification of enterprise authorisation policies and no tools to translate policy specifications to access control implementation mechanisms such as capabilities or Access Control Lists. It is thus difficult to analyse the policy to detect conflicts or flaws and it is difficult to verify that the implementation corresponds to the policy specification. We present in this paper a framework for the specification of management policies. We are concerned with two types of policies: obligations which specify what activities a manager or agent must or must not perform on a set of target objects and authorisations which specify what activities a subject (manager or agent) can or cannot perform on the set of target objects. Management policies are then grouped into roles reflecting the organisational structure. It is thus possible to assign or withdraw persons from the roles without changing their specification. Conflicts may arise in a set of policies which need then to be checked and analysed. We outline principles for conflict analysis and classification. Finally we present the implementation of the access control (authorisation) policies using security agents on a per-host basis to achieve a high degree of transparency to the application level.

Keywords

Distributed systems management, management policies, roles, conflicts, access control.

1. Introduction

Distributed systems are increasingly being used in commercial environments necessitating the development of trustworthy and reliable security mechanisms. These must prevent unauthorised access to resources, compromise of information integrity or loss of confidentiality. Distribution requires the need for encryption over untrusted networks and remote computers may not be fully trusted. Distributed systems may contain millions of objects and cross inter-organisational boundaries forcing decentralisation of security management. There is often no clear informal or formal specification of enterprise authorisation policies and no tools to translate policy specifications to access control implementation mechanisms such as capabilities or Access Control Lists. It is thus difficult to analyse the policy to detect conflicts or flaws and it is difficult to verify that the implementation corresponds to the policy specification.

This paper gives the overview of a security architecture to support an access control model that makes use of domains, explicit policy objects and roles. Domains (Moffett, 1993a) are used to group objects to which a common policy applies, to partition management responsibility in large systems or for the convenience of humans (c.f. file system directories). Domains can contain subdomains to form a hierarchical structure. A role identifies the rights, duties, functions and interactions, associated with a position such as vice president, board director, security administrator, doctor or nurse in a hospital. We model rights as authorisation policies which specify what activities a subject is permitted (or forbidden) to perform on a set of target objects. Duties are modelled as obligation policies which specify what activities a subject must or must not perform on a set of target objects. A role is the set of authorisation and obligation policies which have a particular role position as a subject (Sloman, 1994a). The advantage of using roles for specifying enterprise policies is that individuals can be assigned to or withdrawn
from the role positions without having to respecify the policies composing the role. An object oriented approach to specifying roles permits multiple instances of a basic role to be instantiated e.g. for each nurse in a hospital.

We provide a framework in which roles and inter-role relationships represent the enterprise organisational structure. Roles are expressed in terms of policies which refer to domains to provide a very flexible basis for specifying Role Based Access Control (Lupu, 1995) which caters for large scale, inter-organisational distributed systems. High-level abstract policies can be defined and then refined into a number of implementable policies. Roles therefore rely on management services and in particular on the domain service (Section 2) and on the policy service (Section 3). Roles will be defined and examined in Section 4. Policies within a role and between related roles may conflict. In Section 5 we will analyse and classify the conflicts and outline means of detection and resolution. Section 6 is devoted to the implementation of the authorisation policies.

2. Domains of objects

Large distributed systems may contain millions of objects so it is impractical to specify policies for individual objects. Objects are grouped in domains for specifying a common management policy or to structure and partition management responsibility. A domain is a collection of objects (actually references to object interfaces) which have been explicitly grouped together for the management purposes (c.f. file system directories or folders). A domain is an object, so may also be a member of another domain. Its members are then indirect members of the parent domain. A domain service is provided for the manipulation of the membership information. Further, domain scope expressions can be specified determining the set of objects to which a policy applies. For example D1-D2-O3 represents the objects that are members of D1 with members of D2 and object O3 excluded. Our concept of a domain is very similar to that of a directory in a typical hierarchical file system. The policy which applies to a domain will, by default, propagate to subdomains and to the objects within them, although this propagation can optionally be disabled. Details on the domain structure and the relevant services can be found in (Sloman 1994 a&b).

The Domain Browser is a tool for navigation in the domain structure. In Figure 1 the current domain /Example/Org1/Policies contains two policy objects, has one subdomain (SharedPolicies) and is a member of two parent domains (AllPolicies and Org1).

![Domain Browser](image)

**Figure 1** The domain browser.
3. Management Policies

Policies are used to specify the duties and authorisations of the managers in the system. The policies are interpreted so can be modified, added or retracted without shutting down the system or altering the enforcement of the other policies of the system. We are primarily concerned with two types of policies:

**Authorisation policies** define what activities a subject (manager or agent) can perform on a set of target objects or what monitored information can be received e.g.

```plaintext
A+ *Sregion_agents {"lu1", "lu2": enable(); disable(); reset(); off()} *Sregion
when (time > 08:00) && (time < 18:00)
```

Subjects in the Sregion_agents domain are permitted to perform enable, disable, reset or off operations on objects of type lu1 and lu2 (line units) in the Sregion domain, between hours 08:00 and 18:00.

**Obligation policies** define what activities a manager or agent must or must not perform on a set of target objects. Positive obligation policies are triggered by events e.g.

```plaintext
O+ on overload_event *Sregion_agent {"lu1": disable(), "lu2": enable()} *Sregion
when (08:00 < time) && (time < 18:00)
```

This positive obligation policy is triggered by an overload event and results in the agent disabling line units of type lu1 and enabling line units of type lu2.

```plaintext
O- x:*Sregion_agent {"lineunit": enable(); disable(); reset(); off()} *Sregion
when x.state == standby
```

This negative obligation policy specifies that standby agents must not perform control operations on line unit objects even though they may be authorised to do so.

The general format of a policy is given below with optional attributes within brackets (the braces and semicolon are the main syntactic separators). Some attributes of a policy such as trigger, subject, action, target or constraint may be comments (e.g., /* this is a comment */), in which case the policy is considered high-level and not able to be directly interpreted.

```plaintext
identifier mode [trigger] subject '{' action '}' target [constraint] [exception] [parent] [child] [xref] ';'
```

The **mode** of the policy distinguishes between positive obligations (O+), negative obligations (O-), positive authorisations (A+) and negative authorisations (A-). Negative obligations should be read as “obliged not to” and can be considered as ‘filters’ (Moffett, 1993b) to prevent the actions specified in positive obligation policies being performed under certain circumstances, which is why they cannot be triggered.

The **subject** of a policy specifies the human or automated managers and agents to which the policies apply. The **target** of a policy specifies the objects on which actions are to be performed. Security agents at a target’s node interpret authorisation policies and manager agents in the subject domain interpret obligation policies. Both subject and target can be defined using a domain scope expression which identifies a set of objects in terms of union, difference, intersection and membership operators over sets of domains and objects. By default, policies propagate to subdomains within a domain and hence to indirect members of the parent domain, but the scope expression can limit this propagation to direct members. An advantage of specifying the policy scope in terms of domains is that objects can be added and removed from the domains to which policies apply without having to change the policies. The domain scope expressions are evaluated when detecting potential conflicts to determine the subject and target sets to which the policy applies. The **actions** specify what must be performed for obligations and what is permitted for authorisations. It consists of method invocations or a comment and may list different methods for different object types. Multiple actions can also be specified. The **constraint** limits the applicability of a policy, e.g. to a particular time period, or making it valid after a particular date. An **exception** mechanism is provided for positive obligations to permit the specification of alternative actions and to cater for failures which may arise in any distributed system.
High level abstract policies can be refined into implementable policies. In order to record this hierarchy, policies automatically contain *references* to their parent and children policies. In addition, a manual cross reference list of policies may be kept e.g. to refer to the authorisation policy granting permission for an obligation policy’s activity.

The *policy service* provides tool support for defining policies and disseminating polices to the relevant agents which will interpret them. It also permits policies to be enabled, disabled or removed from the agents (Marriott, 1996a & b). Policies are implemented as objects which can be members of domains (see Figure 2) so that authorisation policies can be used to control access to the policies stored in a policy server, e.g. to permit only authorised managers to define and modify policies.

Figure 2 shows the policy editor the various components of a policy subject, trigger, etc. and the references of the edited policy to parents and other policies (parent-child references are represented in black and cross references are represented in grey).

![Figure 2](imageURL)  The policy editor
4. Roles

Roles specify the obligations and interactions associated with a position in an organisation. Managers can be assigned to or withdrawn from roles without changing the role specification. A single manager may be assigned to one or several roles. In the following we will first describe the implementation of roles when a manager can be assigned to at most one role and then extend this framework to support simultaneous role sessions.

4.1 Roles

A role is composed of a Role Position Domain (RPD) and a set of policies (authorisation and obligation) which have the RPD as a subject. An RPD represents a position or status within the organisation, for instance security administrator, secretary, doctor (Biddle, 1979).

A user is permanently represented within the system by a User Representation Domain (URD). When a person logs in, an adapter object (cf. login shell) is created within the URD to act as the interface process between the person and the computer system. Other agents representing the human and automatically interpreting the implementable policies could also be created in the URD. The user may be assigned to a position and thus to the corresponding role by including her URD in the RPD (see Figure 3). Thus, policies associated with the role will propagate and apply to the URD representing the user. In particular the access control (authorisation) policies associated with a role i.e. which have the RPD as subject domain define the access privileges of the individuals assigned to that role. Individuals can be assigned to or removed from a role without changing the policies and relationships specified for the role. Multiple URDs may be included in a RPD to represent the sharing of a position by a number of people.

![Figure 3](image.png)

A human manager assigned to a role

Note that there may be propagated policies applying to a position domain (and to the URDs included in that domain) which are not a component of the role associated with that position. For example a Professor having her URD included in other organisational domains (research groups, experts on a particular subject) will be subject to other policies than those specified within the role(s) she is assigned to.

A generic role can be specified as a role class in terms of policies which define a set of permitted activities e.g. for a nurse. These could be further specialised e.g. for a surgical nurse (Sandhu, 1996). Multiple instances of surgical nurses could then be created each with their own position and target domains. For example, the nurse assigned to Ward10 would be responsible for a different set of target patients from the nurse assigned to Ward9.

The organisational structure, however, cannot be fully represented by roles since individuals interact and co-operate with each other. The responsibility of the individuals assigned to roles
includes the relationships to other roles e.g. supervision of work. These relationships include the policies between related roles (e.g. Ward10_doctor is permitted to assign tasks to Ward10_nurse) and when and how they can access shared resources such as patient files. Relationships also include the interaction protocols defining the exchange of information and the concurrency constraints on the activities of the related roles. Thus, our framework identifies for a role position: i) the access control and obligation policies related to target objects, ii) the interactions between roles which reflect organisational role relationships, and iii) both intra- and inter-role concurrency constraints. The complete role model is defined in more detail in (Lupu, 1997b).

4.2 Simultaneous role sessions

When a user logs into a system, a process acting as an adapter object between the user and the system e.g. login shell is created within the URD. Note that according to Figure 3 this object inherits the access rights associated with all the roles in which the URD is included. This implies that an action could be performed in a role with access rights inherited from another role. This is not desirable and contradicts the concept of a role as a consistent set of access rights needed to perform the role’s tasks. The activities related to different roles should be presented separately to the user e.g. each role has its own window and the security system must be able to distinguish in which role the user is acting so as to only use the access rights relating to that role.

This can be achieved by keeping the URD out of the RPD and instead specifying a policy permitting the URD to create an agent within an RPD to which it has been assigned. This policy has the RPD as target and is not part of the role’s policies. The adapter object in the URD is similar to an X server maintaining windows for each of the roles in which the user chooses to work. The connection shown between the adapter and agents in Figure 4 does not imply remote communication – the objects could be implemented as threads within a single process but conceptually are members of different domains and so execute with the rights specific to the RPD. The menu within a window can be specific to a role and only display those operations permitted by the policies for that role. The role policies will also specify what target domains can be accessed from the role.

![Figure 4](image)

**Figure 4** Multiple role sessions

The role framework relies on the policy service and the domain structure. Roles are only visible at the specification level and are implemented as domains and policies for access control purposes. The security architecture does not have to know about roles and only sees domains and policies.
5. Conflicts between policies

Multiple policies may apply to the same objects either because objects may belong to several domains e.g. net14, doc (department of computing), ac, uk or in order to reflect different management functionalities e.g. configuration, monitoring etc. Whenever several policies apply conflicts may arise in the set of policies. There are two classes of conflicts: modality conflicts which can be detected by syntactic analysis and application specific conflicts which must be specified by additional constraints on the set of policies.

Modality conflicts are inconsistencies in the policy specification which may arise when two or more policies with modalities of opposite sign refer to the same subjects, actions and targets. This occurs when there is a triple overlap between the sets of subjects, targets and actions as shown in Figure 5, and so can be determined by syntactic analysis of policies. There are three types of modality conflicts:

- **O+/O-** the subjects are both required and required not to perform the same actions on the target objects.
- **A+/A-** the subjects are both authorised and forbidden to perform the actions on the target objects.
- **O+/A-** the subjects are required but forbidden to perform the actions on the target objects (obligation does not imply authorisation in our case).

![Figure 5 Modality conflicts](image)

A second type of conflict refers to the consistency between what is contained in the policies i.e. which subjects, targets and actions are involved and external criteria such as limited resources or the overall policies of the organisation. An example of this type of conflicts arises from the principle of separation of duties (Clark, 1987) e.g. the same manager cannot authorise payments and sign the payment cheques. These conflicts are application specific and cannot be determined directly from the policy specifications – additional information is needed to specify the conditions which result in conflict. These can be specified as a meta-policy i.e. a policy about permitted policies. Several types of application specific conflicts such as: conflict of priorities for resources, conflict of duties, conflict of interests, multiple managers conflict and self-management conflict have been identified in (Moffett, 1994) and classified according to the overlaps between the subject, action and target sets.

Conflict detection between management policies can be performed statically for a set of policies in a policy server or at run time. A run-time mechanism acts as a filter preventing activities that must not be performed (O-) or are not permitted (A-) (Moffett, 1993b). The advantage is that all the constraints of the policies can be evaluated at run time and so all conflicts can be detected, but some conflicts may really be specification errors and should rather be detected by static analysis c.f. compile time vs. run-time error detection for programming languages. The disadvantages of static analysis are that policy constraints cannot be evaluated, as they depend on run-time state, and domain membership may change, so only potential rather than actual conflicts can be detected. Both static and run-time conflict detection are needed, we concentrate here on a static conflict detection tool which assists the users specifying policies, roles and
relationships. In the following we discuss some principles for the detection of the modality conflicts and present an implementation of the conflict detection tool.

Modality conflicts arise from overlapping domains but it is impractical to prevent these overlaps as there is a need for multiple policies to apply to a domain to reflect partitioned responsibility and the various different management functions that can be performed on target objects e.g. different managers may be responsible for maintenance and security relating to a domain of workstations. A precedence relationships between policies must therefore be established. We have considered several precedence relationships among which negative based priority which gives always priority to negative policies and specified priorities which attribute priority labels to the policies. Both these principles fail either because they are too restrictive or too cumbersome. Consider the negative based priority principle and the following policies:

/* All users are forbidden to access the system files */

P1
A- @/users { reboot() } @/workstations

/* The system administrators are authorised to reboot the workstations */

P2
A+ @/users/sys_admin { reboot() } @/workstations

Policy P1, being negative has priority over P2 so the system administrators are denied access to the system files, but then they cannot perform their function. To resolve this conflict it is necessary either to rewrite policy P1 or to exclude the system administrators from the /users domain. Our access control system assumes a negative default authorisation policy and so we would not normally specify P1 but only P2. Database security systems often implement negative authorisation so we permit it at the specification level. Specified priorities are also notoriously cumbersome to use. In particular when several managers across the system are responsible for assigning priorities consistency between the various labels becomes difficult to maintain.

We have chosen to base policy precedence on domain nesting. The principle here is that a more specific policy i.e. a policy applying to a subdomain refers to fewer objects so overrides more general policies applying to an ancestor domain. We recognise that this principle does not apply successfully to all the situations i.e. there are cases in which it is desirable that a global policy overrides more specific ones. For this purpose the conflict detection can be performed with precedence relationships optionally disabled. Figure 6 shows positive authorisation (keys) overriding negative authorisation (crossed out keys). For more details on the conflict detection process please refer to (Lupu 1997a).

![Figure 6](image.png)

Figure 6 The conflict detection window.

Conflicts such as the one shown in Figure 7 may be detected by the tool. The subjects, actions, targets tuple which are common to the policies an which caused the conflict is shown in the
The conflict detection tool enables us to experiment with realistic scenarios and determine the scope of applicability of the domain nesting principle in security related areas.

6. Implementation of access control policies

Once a specification of the access rights is made in terms of authorisation (access control) policies and once the policies have been checked for conflicts it is possible to enforce access control. A security architecture is being developed which aims at enforcing authorisation policies and allowing the development of secure distributed applications on existing operating systems that do not support distributed security. A high degree of authentication and access control transparency is achieved by employing security agents on a per-host basis. These agents are trusted to act on behalf of the application objects on their hosts. Specifically, an Authentication Agent (AA) executes the authentication protocols on behalf of the application objects on the host. In addition, an Access Control Agent (ACA) holds copies of the authorisation policies applying to the objects on its host and determines whether a policy exists to permit a subject to access a target on its host. The Access Control List paradigm is combined with pseudo-capabilities which are used as hints to improve the time-efficiency of the access control decision mechanism. When a subject (which may act on behalf of a user) intends to invoke operations on a remote target, a secure channel is established between these two objects. The channel is identified by a unique channel identifier (chid) and is associated with cryptographic information (session key and cryptosystem used for secure communication), as well as access control information. The ACA on the subject host sends the domain membership certificates and the pseudo capabilities (list of policies applying to the manager invoking the operation) to the ACA on the target host. These are used as hints to search the space of policies applying to the target object (Figure 8).
The target Access Control Agent selects the policies permitting the subject of the established channel to access the target. The list of these policies is referred to as Enabled Policy List (EPL) and it is given to the Reference Monitor (RM) located in the address space of the target (Fig. 9).

A RM makes access control decisions for target objects in its address space using the EPL that applies to a particular subject-target pair (identified by the chid of the established secure channel). Note that the determination of the EPL has to be done once for the lifetime of a secure channel and is based upon the identity and domain membership of the subject. This is authenticated by the target Authentication Agent using a trusted Authentication Service based on symmetric cryptography. A detailed description of the authentication mechanism is given in (Yialelis, 1995). When a new secure channel has been established, the session key for that channel is given to the cryptographic facilities in the address spaces of the subject and target objects. So, further communication between these two objects does not involve the authentication agents. A prototype of this architecture, described in (Yialelis, 1996a&b), has been implemented using Orbix™.

---

™ Orbix is registered Trademark of Iona Technologies Ltd. Dublin, Ireland.
7. Related Work

There has recently been considerable interest in the management policies (DSEJ, 1996; Wies, 1994a & b; Masullo, 1993; Jonscher, 1992). Their purpose is two-fold: to increase automation of the management procedures and to ensure reliable access control. Also a recently emerging area is Role Based Access Control (RBAC, 1995). The work in this area is in general in its preliminary stages, implemented by ad-hoc means and lacking integration. Most of the requirements presented in the First ACM/NIST workshop on Role Based Access Control are satisfied by the role object model presented here.

Little work exists on the conflict analysis issues for management policies. Our concept of domain nesting precedence is based on that of Miró (Heydon, 1990), but they only deal with authorisation policy for file system security. Sandhu (1996) presents constraints which are similar to our meta-policies. The work presented in (Michael, 1993) relates to general policies, expressed in natural language and modelled in an Entity Relationship representation. A theorem prover is used to detect the inconsistencies. The “law governed systems” of (Minsky, 1996) implements a common global set of constraints by means of filters in every node which check that all interactions are consistent with the global law. Another approach, used to detect feature interaction in telecommunication systems (Griffeth, 1993), considers policies as goals and applies planning techniques to resolve situations with incompatible goals. Koch (1996) uses a policy notation based on ours and establishes a semantic graph model to detect ill-behaved policy sets with unsatisfiable pre-conditions. This can also be used to perform “what-if” analysis on chains of policies prior to execution.

8. Conclusions

We have presented a framework for the management of distributed systems based on management policies and domains of objects. We argue that explicit specification of the policies and of the organisational structure is necessary in order to ensure security. The specification of the behaviour can then be incrementally changed and audited without needing to shutdown or isolate the system. Our domains permit grouping of both subjects and targets and help to cater for large scale systems in that policies apply to all objects in a domain. The authorisation policies provide a very flexible means of specifying access control permissions, and include constraints to limit the applicability of the policies. Because policies explicitly identify both subjects and targets, and domains maintain information about policies applying to them, it is easy to analyse the policies to determine those applying to a specific subject or target.

It is clear that roles provide a useful means of modelling organisational structure. However the inter-role duties, responsibilities and interactions need to be specified explicitly as in our role framework. Inheritance relationships between role classes indicate reuse of role specifications which may have nothing to do with hierarchical organisational structures e.g. an engineer may possess access rights that a director cannot inherit as they pertain to activities requiring special skills.

The specification of the access control is drawn from the organisational policies which are often formulated in terms of what the users are not allowed to do, which corresponds to negative authorisations. Allowing both positive and negative authorisations within the same model, leads to potential inconsistent specifications which may be hard to detect and to resolve in an automatic way. Not allowing negative authorisations within the model does not completely solve the problem of inconsistencies. In particular conflicts related to the principle of separation of duty (i.e. the same person cannot be assigned to two particular roles) are application dependent and occur even if negative authorisations are banned. We have outlined directions for conflict analysis and presented a tool for conflict detection.

Our framework makes it easy to define a role administrator role who can create, delete and modify roles so the role structure is evolutionary and can be easily tailored to the needs of the
organisation (Rein, 1993). Other manager roles may be permitted to assign or remove subordinate managers but not change the specification of the roles in the system.

Further work remains to be done relating to the use of the domain nesting precedence at run time and the specification of the meta-policies. Tools also need to be developed for auditing the policy specification in terms of reachability analysis and coverage.

9. Acknowledgements

We gratefully acknowledge financial support for the EPSRC RoleMan project (GR/K 37512) and British Telecom for the Management of Multimedia Networks project. We are grateful to Hewlett Packard for equipment support for this work. We acknowledge the contribution of our colleagues to the concepts described in this paper - in particular Damian Marriott.

10. References


