Meta-Policies for Distributed Role-Based Access Control Systems

András Belokosztolszki and Ken Moody
University of Cambridge Computer Laboratory
JJ Thomson Avenue, Cambridge, United Kingdom
{andras.belokosztolszki, ken.moody}@cl.cam.ac.uk

Abstract

In this paper meta-policies for access control policies are presented. There has been a lot of research into the various ways of specifying policy for a single domain. Such domains are autonomous and can be managed by the users or by a specific system administrator. It is often helpful to have a more general policy description in order to restrict the ways in which policy can be modified. Meta-policies fill this particular role. With their help changes to policy can be made subject to predefined constraints. Meta-policies are long lived and so can provide users with stable information about the policy of the system. In addition they can provide bodies external to a domain with relevant but restricted information about its policies, so forming a basis for co-operation between domains. For example, a domain's meta-policy can function as a policy interface, thus establishing a basis for agreement on the structure of the objects accessed. In this way it is possible to build service level agreements between domains automatically.

1. Introduction

The goals of access control systems are to protect resources from unauthorized access and to ensure access to those resources for authorized users. There are a number of models of access control which aim to achieve this goal. Traditional models include discretionary access control (DAC) and mandatory access control (MAC) [19]. A promising alternative to these models is role-based access control (RBAC) [5, 6, 13, 16], which allows the specification of access control policy in a way that maps naturally to an organization's structure. This approach brings advantages such as easier understanding of access control policies and scalable administration. Traditional access control models can easily be simulated in RBAC [18].

Roles introduce a level of indirection in the mapping of users to privileges. Instead of mapping users directly to privileges, users can assume a number of roles, and these roles are mapped to privileges. Rules specify what roles can be activated and under what conditions. These role activation conditions can include prerequisite roles and environmental constraints, changing the static user-to-privilege mapping into a dynamic mapping that depends on the environment. These rules increase the expressive power of the role-based access control model.

Organizations often have a hierarchical management structure. Each unit can consist of smaller units, which have their own users and resources. The access control policy for a unit is either managed locally or at a higher level. In RBAC a policy is a set of *activation rules*, which control the user-role mapping, and a set of *authorisation rules*, which control the role-privilege mapping. In order to control access to a policy this same access control mechanism can be used [16], in which case roles and rules are treated as resources.

Within an organization certain policies will be organization wide, to which every unit within that organization must adhere. It is important to specify who may access and modify each of the sets of rules which form the overall access control policy of the organization. Since individual units have a degree of autonomy, they can extend the high-level policy of an organization or even define their own sets of rules. It is difficult to specify a policy that will not conflict with local requirements, and to ensure that the original intentions behind organizational policy are preserved. Local administrators must be able to introduce policy to meet local needs, but it is essential to control the consequences of local policy modifications. Restricting access to local roles and rules is of little help if new roles with their own privileges can be created.

The relationship between autonomous units within an organization is open. They communicate, exchange data, allow access to their resources and cooperate among themselves. This cooperation can be handled by so called *service level agreements* (SLAs), which describe the roles and resources that take part in communications between units. Changes to the organizational policy or to the local policy of either of the units involved in a cooperation described by a SLA require the revision of the SLA itself. In the case of a large number of units the number of contracts that must be changed can become unmanageable. An example is the UK National Health Service (NHS), where a unit can be a hospital. The number of hospitals is large, and all must be able to cooperate. A single policy change can require a very large number of modifications to SLAs.

In this paper we propose a mechanism to control local freedom within units of an organization. The goal is to enable local administrators to establish a local policy that complies with the overall policy of the organization. Users of the local policy can then be sure that any rights guaranteed at organization level have not been restricted in their local environment.

It is also the aim of our work to ease the task of updating SLAs. The same mechanism as is proposed to describe a policy can be used to generate SLAs in an automatic fashion. This is especially important when the requirement for interoperation is rare. For example in inter-hospital communication two hospitals may work together frequently, whereas a third hospital might make no contact with them. However, there must still be a potential to cooperate, since they are all part of the same organization.

We achieve these goals by means of meta-policies, sets of rules that describe the management of policies.

This paper is organized as follows: After a short summary of related work section 2 gives an overview of OASIS, a role-based access control model that includes several extensions to the basic RBAC models. Section 3 describes access control domains, their administration and ways of enabling cooperation among them. This is followed by section 4, which motivates the use of meta-policies and classifies their various types. It also describes the specification of meta-policies, compliance testing and the generation of service level agreements to cater for inter-domain cooperation. Section 5 gives an extended example that illustrates the use of meta-policies. Finally, section 6 concludes the paper.

1.1. Related Work

The idea of meta-policies is not new. RBAC constraints are often insufficient to express certain separation of duties, or expressing separation of duties by using constraints introduces runtime overhead. By using policies that describe policies this runtime overhead can be reduced. Several RBAC implementations use specification time policy checks to resolve rule conflicts, and it was noted that these specification time checks can be used to test the policy for other properties.

Lupu and Sloman in [11] describe a model that supports distributed and automated policy management. Apart from authorization policies their model supports obligation policies and negation. Both of these extensions can lead to policy conflicts. To provide syntactic analysis of policies Lupu and Sloman introduce meta-policies. These meta-policy checks can be integrated into their specification-time and run-time conflict detection mechanisms. Example meta-policies to express self-management and separation of duty are given in [4].

Sandhu introduces the OM-AM framework in [14]. It has a layered architecture that addresses issues like *what* the security objectives are and *how* these objectives can be met. The description of these objectives can be looked at as meta-policies. OM-AM is applied in an RBAC context and a distributed RBAC case study is provided.

Koch et al. consider policy evolution in [10]. Their framework based on graph transformations allows detailed description of policy changes over time.

These approaches do not consider using meta-policies for inter-domain cooperation and for defining relationships among policies.

An early example of work on inter-domain policy cooperation is the Domino project, summarized in [12].

2. OASIS

OASIS (Open Architecture for Secure Interworking Services [2]) is a RBAC architecture developed at the University of Cambridge. It has a clearly defined model based on logic that allows various properties to be checked.

Roles are activated in the context of a session, which starts when a user is authenticated at a service. OASIS does not support role hierarchies, and it is the belief of the developers that role hierarchies are not required. One of the reasons is that organizational roles and their hierarchy do not always map easily to access control roles [1]. If necessary additional role-to-privilege authorisation rules can easily be generated from the specification of a role hierarchy.

The original OASIS defined by Hayton in [8] is derived from a capability system. In OASIS, unlike most RBAC systems, roles can be managed in a decentralized way, since roles and policy rules are specific to a service. Inter-service cooperation is supported. OASIS handles policy enforcement using role activation rules and digitally signed credentials. Role activation rules are expressions based on first-order logic, which specify what prerequisite roles or auxiliary credentials the user must hold in order to enter the new role, and what additional conditions must be met at the time.

Since the first version important extensions have been made to OASIS, the most important being *appointment*. Appointment, as described in [21], abstracts role delegation by allowing users to appoint other users to activate roles, but the appointers are not required to hold the delegated role. Appointment does not itself cause role activation, but instead certificates are issued that can be presented as prerequisites during role entry. These certificates often correspond to long-lived credentials such as academic qualification or membership of an organization.

OASIS stands out from other RBAC implementations by its fast, event-based revocation mechanism. Preconditions in role activation rules can have a *membership* flag that specifies that the new role should be deactivated if the condition should become false. In particular, the role may be deactivated if the user is no longer active in a prerequisite role.

Recent research into RBAC has recognized the need for context awareness [3, 9]. This has been supported since the first version of OASIS by means of role parameters and simple environmental constraints in role activation rules. Role parameters and variables in environmental constraints are strongly typed in OASIS.

Role activation rules have the form:

$$r_1, r_2, ..., r_{n_r}, ac_1, ..., ac_{n_{ac}}, e_1, ..., e_{n_e} \vdash r$$

where n_r, n_{ac} and n_e are the number of prerequisite roles, appointment certificates and environmental constraint predicates respectively. r_i for $1 \leq i \leq n_r$ are prequisite roles, ac_j for $1 \leq j \leq n_{ac}$ are prequisite appointment certificates and e_k for $1 \leq k \leq n_e$ are environmental constraint expressions. Predicate expressions on the left hand side of the rule are called *preconditions*. r is the *target* role.

Preconditions may contain parameters. Parameters have two modes, *in* and *out*. *in*-parameters must be set before a precondition is evaluated, while *out*-parameters are set as a side effect of evaluation. Role and appointment preconditions may contain only *out*-parameters, which are set by pattern matching. Parameters of environmental constraints may be either *in*- or *out*- parameters. Variables set while evaluating preconditions can be used to set parameters in the target role.

Authorisation rules are of the form:

$$r, e_1, ..., e_{n_e} \vdash p$$

where r is the role, e_i for $1 \leq i \leq n_e$ are environmental constraint expressions and p is the privilege.

A set of the above rules defines the policy for an OASIS service. Since role parameters and environmental conditions are included the rules are very expressive, and meta-policies are needed to ease administration.

3. Administrative Domains

Authentication of users and control of access to resources are specific to a domain. Such a domain encompasses a set of users and a set of resources, and access control is governed by its policy. Domains are usually defined to reflect the structure of an organization.

Most research in RBAC has concentrated on access control within a single domain. Little work has been done in the area of cooperation between domains since it is usually assumed that access control is defined for a single administrative unit.

This paper does not address the problems of policy consistency within a single domain. It is assumed that every domain has a consistent policy.

3.1. Administration

Domains are administered locally, within the domain. Sometimes only a small group of people are responsible for managing the rule set of the domain. These people determine local policy, and usually there is little or no means of controlling their actions. Organizations depend on their work, and treasure them.

It is often the case that users require some level of control over the access control rules. For example, users wish to manage the access control policy for the files that they own. Access control to the rules and roles of the domain can be handled by the same access control mechanism used to control access to other resources. This is described in [15].

In self-administered systems it is possible to set up a wide variety of access control policies, allowing users to manage a subset of the resources including those that handle access control to what they manage. There is also a system administrator with full control. He or she is needed to correct user errors and to ensure the overall consistency of the policy. Unfortunately users have no control over the actions of the system administrator, modifications of their access control policy or their rights within the system. This is not because system administrators lack goodwill or wish to have full control over a particular domain, but because the policy rules of a system can be complex, and often this complexity prevents ordinary users from understanding it.

3.2. Inter-Domain Communication

Domains are not closed; they can be accessed from other domains. There are a number of reasons why a particular domain may not want its access control policy to be public, but there must be an interface to allow others to access this domain. Access takes place within a session, following authentication of a user within some domain. If such users are to access resources in other domains then there must be (mutual) trust of authenticated users. Access to external domains is usually through a few exported roles that the external domain accepts, and it is important to exercise control over the use of such roles. This is usually managed by means of SLAs. SLAs behave like contracts between two domains, and control the number and format of the roles accepted externally. Based on these agreements the server issuing role membership certificates (RMCs) will monitor membership conditions and notify the external domain in case of an event that invalidates the exported RMC.

One drawback of SLAs is that they cannot be built dynamically at runtime. When two domains agree mutually to accept certain roles the system administrators of the two domains have to sit down together and set up a SLA. When the number of domains is large and communication among them is rare, it is a tedious task to set up a new SLA with every other domain each time that the local policy changes.

There are a number of problems associated with interdomain communication. The first arises from the various data types involved. Roles can have parameters, and these parameter values can be used in rules. If such roles are exported and used in foreign domains care must be taken to ensure the correct interpretation of these role parameters. Another problem is the handling of external roles. It is important that the privileges granted to the holders of imported roles are appropriate to the users who have been authenticated into those roles.

4. Meta-Policies

Meta-policies are introduced to solve some of the problems outlined on the previous pages of this paper. They provide a mechanism to give users information about an access control system. This information can also be used to tell external units about the access control policy used within a domain or to help managers to control the policy of a subdomain.

Meta-policies specify a set of constraints for a set of rules. The use of meta-policies consists of the following steps:

Specification: This defines the meta-policy. It consists of the data types, general rules, constraints, and so forth. The detailed description is in section 4.1. The goal is to describe the expectations from a policy, such as what user-privilege assignments it *must* contain, what user-privilege assignments it *must not* contain, what data structures or objects are accessed and how these can be accessed remotely.

Compliance check: This process examines a domain's policy or a subset of that policy using the meta-policy specification. The entities in the meta-policy description must first be mapped to the entities in the policy. The compliance test is performed on the basis of this mapping. If the compliance test succeeds a certificate is issued for the domain's policy. This step must be performed each time a policy is modified.

Use of the compliance certificate: The certificate issued can be used to prove to users that the domain's policy complies with a general meta-policy specification with which the users are familiar.

The certificate can also serve as an *export interface* to other domains; using the semantic information encoded in the meta-policy external domains can accept exported roles and associate privileges with them. Only the certificate and the interface are visible to external domains, the internal structure of the exporting domain's policy cannot be accessed.

The certificate can be used as an *import interface* description to ensure that external roles are handled in an appropriate way. This is especially important when generating service level agreements automatically.

4.1. Specifying Meta-Policies

The first step is to specify each meta-policy. Meta-policies describe policies. This is done with the help of an abstract (or generic) policy model. The properties required are formulated with respect to this model. Later this model is mapped to a concrete policy, and the properties expressed using the model can be checked on that policy. To define this more general model the following must be specified:

Data types describe the data types used in role parameters, privileges, functions and environmental constraints. These data types must contain sufficient information about their semantics or must be generally accepted or widely used data types, like the data types of C, SOAP or SQL. The data types defined can be used in the general roles, rules and functions of the abstract model.

Objects specify the generic objects that will map onto a protected object. This description consists of the access methods and their parameters, which at some granularity determine the privilege set for the object.

Functions are described in order to promote agreement on the use of environmental constraints. The exact body of the function implementing the predicate is not given, only a partial description; this includes the signature and possibly some indication of the semantics, for example a reference to a service that provides this function or a natural language description.

Roles are specified with their signature. The signature includes the parameter types, and information about those parameters. It can also include statements about what privileges are expected to be conveyed by the role.

Appointment certificates are specified in a similar way to roles, except for the lack of a privilege mapping.

Explicit rules are specified if certain explicit rules *must* be present in the policy being checked. Since the policy rules are permissive and may not contain negation explicit meta-rules must not contain negation either.

Invariance rules describe invariants of access control policies that comply with this specification. Basically these rules specify role-to-privilege mappings that must be included in the policy. Negative rules can also be included; these specify privileges that *must not* be granted to certain roles (or users). These rules may seem similar to the rules of a policy, but they are specified in terms of the abstract model only. Their equivalent must be provably present (either explicitly or implicitly) or absent (in the case of negative rules) in the concrete policy that is to be checked.

4.1.1. Negation

Meta-policy rules can contain negative rules to restrict certain privileges. Although in many cases negative rules can be expressed by permissive positive rules, the equivalent expressions are cumbersome and are difficult to read or administer [17]. On the other hand, access control rules that include negation directly can be expensive to enforce at runtime. The unification engine of the access control system must be more complex, and care must be taken to resolve any conflicts and ambiguities that may arise.

As meta-policies are checked only when a new access control policy is introduced, negation will not lead to inefficiencies at runtime and the unification engine can be kept simple.

4.1.2. Separation of Duties

Using meta-policies certain static policy checks can be performed, for example static separation of duty constraints [7, 20]. As in the case of negation these checks are performed only once per policy version, and so do not adversely affect the efficiency of unification at runtime.

4.2. Specifying Compliance Meta-Policies

Meta-policies serve two related but distinct purposes: constraining the policy of a particular domain (*compliance*), and enabling domains to interoperate at policy level (*communication*). These purposes are not exclusive, and often a single meta-policy will serve both. For clarity we discuss the two purposes separately.

Compliance meta-policies are a set of constraints against which to check policies. The primary purpose is to examine and audit the policy of a domain based on the rules in the meta-policy. If the check succeeds a certificate is issued. This certificate can be used to build up trust relations; it can be presented as proof of compliance with the meta-policy without the need to make details of the access control policy public. Because of this property, certificates can be shown to both local and external bodies.

Compliance meta-policies can also be used to control policy evolution. General constraints on the policy of a domain can be specified in a meta-policy. Whenever the domain's policy is modified, the new policy can be checked to determine whether the general properties still hold. This use of meta-policies greatly aids policy evolution and policy administration. If there is a meta-policy, authorized users of the system can modify policy rules that relate to resources that they own, provided that the modified policy still complies with the meta-policy.

4.2.1. Example

The following example illustrates some of the features of compliance meta-policies. The example meta-policy states that users must have read access to their personal data, and that no one except the owner of that data can modify this personal data. For this the following must be specified:

- The personal data object. This description can include the structure of the object and the privileges that are associated with the data object.
- A classification of the privileges. This distinguishes *read access privileges* from *modification privileges*.
- An abstract description of the users. In the case of OASIS this is a parametrised initial role.
- Positive rules that associate read access privileges with each user.
- Negative rules that restrict modification privileges for a personal data object to roles held by the owner of the object.

Note that the above policy does not say that the owner of the personal data object can modify it. Also, all roles and privileges are abstract in the meta-policy specification.

This meta-policy is the privacy policy of an organization with a hierarchical domain structure. Every domain in the organization is expected to comply with this meta-policy. If the compliance check succeeds for the policy of a particular domain a certificate is issued. This certificate proves to both users and senior management that the administrators of the domain respect the high-level policies of the organization.

4.3. Checking Compliance Meta-Policies

The following must be provided to check the policy of a specific domain against a meta-policy:

- The *mapping of data types*. If there is a local data type in the domain that is equivalent to the data type of the meta-policy, then the mapping is a bi-directional, one-to-one mapping. If there is no equivalent data type, then a proper conversion must be provided with the help of the functions available to the domain. The meta-policy specifies whether the meta-policy data type must be converted to a data-type of the domain or vice versa.

- The *mapping of functions*. The purpose is to define a mapping of each meta-policy function to a local function of the domain. As functions can have different parameter signatures, local functions can be used to convert the parameters to a different format. An example is the conversion of two ASCII strings (surname and first name) to a UNICODE string (full name). As in the case of data type mapping, the meta-policy specifies the direction of the mapping that must be provided.

- The *mapping of generic objects*. The objects of the abstract model must be mapped to local objects; this includes identifying the relevant access privileges. In this context it may also be necessary to use local functions to convert parameters. It is sometimes difficult to provide a one-to-one mapping of privileges, but often that is not required. The meta-policy can specify that a *privilege subset relation* is sufficient instead of a one-to-one mapping.

- The *mapping of roles*. Abstract roles of the meta-policy must be mapped to the local domain roles. Simple meta-roles, i.e. meta-roles that have no parameters, can easily be mapped to simple policy roles. Problems can arise when meta-roles have to map onto policy roles, but the two roles have different parameter types, a different number of parameters, or different semantics for parameter values. In such cases *interface roles* may need to be introduced, see section 4.5.

- The *mapping of appointment certificates*. Abstract appointment certificates are mapped to concrete appointment certificates in much the same way as roles.

Compliance testing is based on the above mappings. First, check whether all data types, functions, objects, roles and appointment certificates have been mapped, and that the direction of these mappings is correct.

Next, check that each explicit rule of the meta-policy has its counterpart in the concrete policy. Explicit meta-rules are translated into rules that use the roles, privileges and functions of the domain to be checked. The resulting rule must be present in the domain's policy in an explicit form.

Finally, check the invariance rules of the meta-policy. This includes checking the validity of both positive and negative implicit meta-rules. This involves theorem proving and is potentially an expensive process, but it has to be performed only when policy is modified, and the overhead at runtime is not affected.

Note that when checking compliance with a meta-policy that contains negative meta-rules, the entire rule and role set of the domain must be considered. If only a part of the policy is examined, negative meta-rules cannot be checked.

When the policy of a domain is validated against a meta-policy a certificate is issued for that version of the policy, and this certificate can be used in later access control. As mentioned earlier, this certificate guarantees that the policy of the domain complies with organizational standards.

In some circumstances a trusted third party will be made responsible for checking compliance with meta-policies. This trusted third party must have access to the conversion functions available locally, together with their semantics.

4.4. Specifying Interface Meta-Policies

Interface meta-policies are similar to compliance meta-policies. They consist of two parts, an *export* part and an *import* part.

The *export* part describes what must be exported from a domain and in what format. This can include description of roles, appointment certificates and so forth.

The *import* part describes external roles that are accepted via this interface meta-policy, and the semantics expected for such roles in the complying domain. This can include a description of the generic privileges that must or must not be associated with a particular external role.

4.5. Complying with Interface Meta-Policies

Interface meta-policies are used to facilitate cooperation between domains. The policy of a particular domain can comply either with the export, or with the import or with both the export and import parts of the meta-policy. The extent to which a domain's policy complies with a meta-policy determines the relationship of the domain to the meta-policy. This relationship can be *role-exporting*, *role-importing* or both.

When checking compliance with interface meta-policies mappings of data types, functions, objects, roles and rules must be provided. The direction of the mappings will depend on whether the domain is to act as role exporting, role importing or both.

Meta-policies can be taken into account when a domain's policy is designed. In such cases local roles can be created to reflect the structure of the meta-roles. Such a design can lead to simple compliance mappings, since the roles to be mapped have similar structure.

If the local policy of the domain has already been established, then it can be extended with *interface roles* that map easily to the roles of the meta-policy. The rules of the domain can be modified in order to assign privileges to interface roles or to let them serve as prerequisites for activating roles of the existing local policy. This can meet the same needs as when the policy is designed to comply with the meta-policy. Sometimes a policy cannot be extended in such a way; for example, separation of duty constraints may not allow a new role to obtain certain privileges. In such cases the meta-role must be mapped directly to an existing local role which conveys the required privileges; that role may have an inappropriate structure, and it may be necessary to resolve parameter issues by using local functions.

The certificate of compliance with an interface meta-policy may include mapping information. This information can be used to optimize type conversions when generating service level agreements.

4.6. Generating SLAs Automatically

When two domains comply with the same interface meta-policy, one with the export part and the other with the import part, a service level agreement can be generated automatically, see figure 1.

The exporting domain's task in the agreement is to issue local users with exportable roles if those users want to use external services. The exporting domain must also check membership conditions of the exported roles and, whenever such an exported role is deactivated, notification must be sent to the importing domain.

The importing domain accepts the roles and uses them according to the interface meta-policy. It must also accept notification when any external role is deactivated; an event channel is established when an imported role is first validated by the exporting domain. These events cause cascade deactivation of any local roles that were activated according to a rule in which the imported role is a membership precondition.

Appointment certificates are handled in a very similar way. An appointment certificate that is managed in the exporting domain can be used as a prerequisite credential in the importing domain. The structure of the certificate is described in the meta-policy; the exporting domain will issue users with an equivalent appointment certificate that complies to the interface. The importing domain is therefore able to relate the appointment certificate to the local context. In addition the exporting domain must notify the importing domain if the appointment certificate is revoked.

For service level agreements a role mapping is provided. This includes a description of how role parameters should be set for exported roles. Each role exported from the exporting domain is transformed into a generic meta-role, which is accepted by the importing domain as described in section 4.5. Conversion from a role of the exporting domain is based on compliance of the exporting policy with the interface meta-policy; conversion from the meta-role to a local role is based on compliance of the importing policy with the meta-policy.

The mapping information stored in compliance certificates sometimes enables the exported role to be mapped to a role in the importing domain more simply than through the use of a meta-role.

5. Example

This section gives an example of the use of interface meta-policies. It is motivated by the National Health Service (NHS), in which patients' records are stored in a distributed heterogeneous environment and cooperation between domains – hospitals, general practices etc. – is regulated by the NHS. In this example we describe such higher-level regulations by means of meta-policies. The meta-policy states that *a generic GP role must have access to its patients' records*. The example uses a simplified model of patient record fields, namely: *name, address, next_of_kin, biochemistry, haematology*

5.1. The Meta-Policy

The meta-policy consists of the following, see section 4.1:

Data type:

The data type part describes what data types are used in role parameters, functions, and so forth. Types are identified by abstract value set identifiers, together with a representation. In our case we have an identifier for doctors which is represented by a four-byte unsigned integer, a patient identifier represented as a UNICODE string, and boolean.

```
GMC_Id  (integer, 4 byte, unsigned)
NHS_Id  (Unicode)
boolean ({true, false})
```

Objects:

The objects section of the meta-policy specification describes objects and privileges. In this example we have only one object type, the patient record. Access to this object is handled by the following abstract privileges:

```
Read_Name(patient_Id:NHS_Id)
Read_Address(patient_Id:NHS_Id)
Read_NextOfKin(patient_Id:NHS_Id)
Read_Biochemistry(patient_Id:NHS_Id)
Read_Haematology(patient_Id:NHS_Id)
```

All of these privileges require a $patient\_Id$ parameter. Informally, their semantics is to authorise the holder to read the respective fields from the record of patient $patient\_Id$.

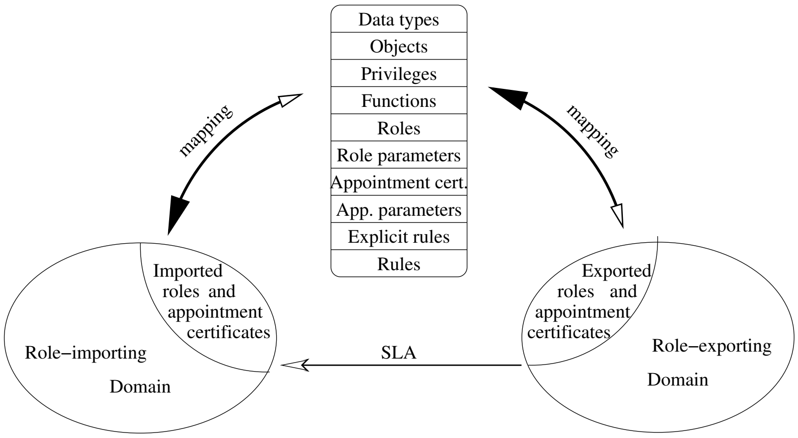

Figure 1. Automatic generation of service level agreements.

Functions:

There is one function in this meta-policy, which checks whether a patient is registered with a certain GP:

$$GP\_patient : Dom_{GP\_Id} \times Dom_{patient\_Id} \rightarrow boolean$$

where $Dom_{variable}$ denotes the domain of the variable.

Roles:

The only role in this meta-policy gives an abstraction of the general practitioner role in the different domains. This generic role is identified by a globally known identifier issued by the General Medical Council.

$$GP(GP\_Id : GMC\_Id)$$

Rules:

There are no explicit rules in this meta-policy, but it contains the following meta-rules:

- $GP(GP\_Id),\ GP\_patient(GP\_Id,\ patient\_Id) \vdash Read\_Name(patient\_Id)$
- $GP(GP\_Id),\ GP\_patient(GP\_Id,\ patient\_Id) \vdash Read\_Address(patient\_Id)$
- $GP(GP\_Id),\ GP\_patient(GP\_Id,\ patient\_Id) \vdash Read\_NextOfKin(patient\_Id)$
- $GP(GP\_Id),\ GP\_patient(GP\_Id,\ patient\_Id) \vdash Read\_Biochemistry(patient\_Id)$
- $GP(GP\_Id),\ GP\_patient(GP\_Id,\ patient\_Id) \vdash Read\_Haematology(patient\_Id)$

These meta-rules state that a GP can access each field of the records of its patients. This is a lower bound on the GP's rights. In a domain that complies with this meta-policy the equivalent of the generic GP role must possess the equivalent of the above privileges.

All of the rules in our example have one prerequisite role $GP(GP\_Id)$ and one environmental constraint $GP\_patient$.

5.2. The Exporting Domain

The exporting domain in this example is a GP practice. As there is no negation in the meta-policy, only a small part of the rules and roles of the practice's policy need be considered. The relevant part is:

Roles:

$$localGP(name : text)$$

This role is the exported role. For compliance this role must be mapped to the meta-policy's $GP(GP\_Id)$ role. For this, a local function must be provided that converts $name$ to GP_Id. Suppose that this function is: $nameToGP : text \rightarrow GMC\_Id$, where GMC_Id is in the format specified by the meta-policy. If required by the meta-policy, an inverse function must also be provided. This inverse function is required when the exporting domain has to validate instances of the generic role.

5.3. The Importing Domain

In this example the importing domain is a hospital. Once again the meta-policy rules do not contain negation, so that it is enough to consider only part of the local policy. The relevant policy description of this domain is:

Roles:

```
GPexternal(GP_Id:GMC_Id)
```

```
record_reader1(GP_Id:GMC_Id)
record_reader2(GP_Id:GMC_Id)
```

There are three relevant roles. The first one is $GPexternal$, a special role created explicitly for external GPs; it has the same parameter type and parameter semantics as the GP role in the meta-policy. This role does not have any associated privileges, it is used only as a prerequisite to enter other roles. In addition to the $GPexternal$ role there are two roles $record\_reader1$ and $record\_reader2$; these roles convey local privileges to read health record data. The authorization rules are:

$$record\_reader1(x), PatientOfGP(x, pid) \vdash readPersonalData(pid)$$

$$record\_reader2(x), PatientOfGP(x, pid) \vdash readHealthData(pid)$$

In the above authorization rules the privileges have one parameter, and that parameter is the patient identifier pid. Note that the privileges of the $reader$ roles are dependent on the patient through the environmental predicate $PatientOfGP$. The privileges distinguish only between $personaldata$ and $healthdata$ fields.

Rules:

The role entry rules of the domain are:

- $GPexternal(x) \vdash record\_reader1(x)$
- $GPexternal(x) \vdash record\_reader2(x)$

Compliance

For compliance the following is included: data type mapping and conversion, object mapping, function mapping, role and rule mapping.

The function mapping includes handling the predicate $GP\_patient(GP\_Id, patient\_Id)$. In order to map this predicate we must convert the general patient identifier specified in the meta-policy to a local identifier pid. This is achieved by the function

$$NHSpidTopid : Dom_{patient\_Id} \rightarrow Dom_{pid}$$

which must be available in the local domain. The predicate equivalent to $GP\_patient$ locally is

$$PatientOfGP : Dom_{GP\_Id} \times Dom_{pid} \rightarrow boolean$$

and the mapping is given by:

$$GP\_patient(x, y) = PatientOfGP(x, NHSpidTopid(y))$$

We also use the conversion function $NHSpidTopid$ for the privilege mapping.

The privilege mapping is provided as follows (the meta-policy privileges have a $M.$ prefix to avoid name conflicts):

$$M.Read\_Name(y) \prec readPersonalData(NHSpidTopid(y))$$

$$M.Read\_Address(y) \prec readPersonalData(NHSpidTopid(y))$$

$$M.Read\_NextOfKin(y) \prec readPersonalData(NHSpidTopid(y))$$

$$M.Read\_Biochemistry(y) \prec readHealthData(NHSpidTopid(y))$$

$$M.Read\_Haematology(y) \prec readHealthData(NHSpidTopid(y))$$

The $\prec$ sign means that the local privilege includes the meta-policy privilege, but it may not be equivalent to it.

The meta-policy GP role is mapped directly to the local $GPexternal$ role. No parameter conversion is needed, as it was a role specially designed for external GPs.

The meta-rules are checked. The five rules have similar structure, so only the mapping of the first one is shown. The meta-rule is

$$GP(x),\ GP\_patient(x, y) \vdash Read\_Name(y).$$

This rule is translated to use the local roles, functions and privileges. The resulting rule is:

$$GPexternal(x),\ PatientofGP(x, NHSpidTopid(y)) \vdash readPersonalData(NHSpidTopid(y))$$

The above rule must have an equivalent in the local policy. It is not trivial to find this equivalent, and often it cannot be done automatically, since the semantics of the functions are unknown. This task can be eased if the inverses of certain functions are known, but usually the process requires human intervention. Here we must show that there is a path from the role $GPexternal(x)$ to the required privilege $readPersonalData(NHSpidTopid(y))$ for patients y who are registered with x. We must first enter the role $record\_reader1(x)$; the role parameter is propagated from the role $GPexternal(x)$. This role conveys the privilege $readPersonalData(NHSpidTopid(y))$, since we know that the two predicates $GP\_patient(x, y)$ and $PatientOfGP(x, NHSpidTopid(y))$ are equivalent. Similarly the remaining four rules can be checked.

5.4. Service Level Agreement

Based on these two compliances a service level agreement can be generated automatically. The exporting domain allows its users to use the local role in the importing domain. The exporting domain notifies the importing domain if the exported role is deactivated. The exporting domain also checks the validity of exported roles at the importing domain's request.

The importing domain accepts the exported role, uses it in rules and grants access to resources appropriately.

6. Conclusion

In this paper we studied the use of meta-policies to check policies at specification time. We categorize these meta-policies as either *compliance* or *interface* meta-policies. We use compliance meta-policies to provide both local and external users with information about an access control policy without making its details public. This builds trust relationships, and it also helps policy evolution. Compliance meta-policies ease the task of policy administrators since they constrain policy design, and thus allow policies to evolve while maintaining the properties specified in a meta-policy.

We also showed how meta-policies can be used in inter-domain cooperation. These interface meta-policies describe how the roles of one domain can be used in another, and how certain guarantees can be made about the use of such *roaming* roles.

Desert, our current proof of concept implementation, consists of a mapping editor and a SLA generator. The mapping editor can create mappings between policies and meta-policies, and it also performs compliance checks. The SLA generator creates SLAs between two domains based on policy mappings with respect to the same meta-policy.

Future work includes defining a formal model for the meta-policies and integrating the current implementation into the OASIS framework.

7. Acknowledgement

András Belokosztolszki gratefully acknowledges the support provided to him by King's College Cambridge, the John Stanley Graduate Fund and the Overseas Research Students Awards Scheme.

References

[1] R. Awischus. Role based access control with the security administration manager (SAM). In *Proceedings of the second ACM workshop on Role-based access control*, pages 61–68, 1997.

[2] J. Bacon, K. Moody, and W. Yao. Access control and trust in the use of widely distributed services. In *Middleware 2001*, volume 2218, pages 300–315, November 2001.

[3] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd. Securing context-aware applications using environment roles. In *Sixth ACM Symposium on Access Control Models and Technologies*, pages 10–20, 2001.

[4] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *Policies for Distributed Systems and Networks,International Workshop, POLICY 2001, Bristol, UK*, pages 18–38, 2001.

[5] D. F. Ferraiolo, J. F. Barkley, and D. R. Kuhn. A role-based access control model and reference implementation within a corporate internet. In *ACM Transactions on Information and System Security*, volume 2, pages 34–64, 1999.

[6] L. Giuri and P. Iglio. A formal model for role-based access control with constraints. In *Proceedings of the Ninth IEEE Computer Security Foundations Workshop*, pages 136–145, 1996.

[7] V. D. Gligor, A. I. Gavrila, and D. Ferraiolo. On the formal definition of separation-of-duty policies and their composition. In *IEEE Symposium on Security and Privacy*, pages 172–183, 1998.

[8] R. Hayton. *OASIS An Open Architecture for Secure Interworking Services*. PhD thesis, University of Cambridge, 1996. Technical Report No. 399.

[9] M. H. Kang, J. S. Park, and J. N. Froscher. Access control mechanisms for inter-organizational workflow. In *Sixth ACM Symposium on Access Control Models and Technologies*, 2001.

[10] M. Koch, L. V. Mancini, and F. Parisi-Presicce. On the specification and evolution of access control policies. In *Sixth ACM Symposium on Access Control Models and Technologies*, pages 121–130, 2001.

[11] E. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, 25(6):852–869, 1999.

[12] J. D. Moffett. *Specification of Management Policies and Discretionary Access Control*, chapter 17, pages 455–479. Addison-Wesley, 1994.

[13] M. Nyanchama and S. Osborn. The role graph model and conflict of interest. *ACM Transactions on Information and System Security (TISSEC)*, 2(1):3–33, 1999.

[14] R. Sandhu. Engineering authority and trust in cyberspace: the OM-AM and RBAC way. In *Proceedings of the fifth ACM workshop on Role-based access control*, pages 111–119, 2000.

[15] R. Sandhu, V. Bhamidipati, E. Coyne, S. Ganta, and C. Youman. The ARBAC97 model for role-based administration of roles: preliminary description and outline. In *Proceedings of the second ACM workshop on Role-based access control*, pages 41–50, 1997.

[16] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[17] R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST model for role-based access control: towards a unified standard. In *Proceedings of the fifth ACM workshop on Role-based access control*, pages 47–63, 2000.

[18] R. Sandhu and Q. Munawer. How to do discretionary access control using roles. In *Proceedings of the third ACM workshop on Role-based access control*, pages 47–54, 1998.

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

[20] R. T. Simon and M. E. Zurko. Separation of duty in role-based environments. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[21] W. Yao, K. Moody, and J. Bacon. A model of OASIS role-based access control and its support for active security. In *Sixth ACM Symposium on Access Control Models and Technologies*, pages 171–181, 2001.