Obligations for Role based Access Control
Gansen Zhao, David Chadwick, Sassa Otenko
The computing Lab
University of Kent, UK
{gz7, d.w.chadwick, o.otenko}@kent.ac.uk

Abstract:
Role based access control has been widely researched in security critical systems. Conventional role based access control is a passive model, which makes authorization decisions on requests, and the authorization decisions contain only information about whether the corresponding requests are authorised or denied. One of the potential improvements for role based access control is the augmentation of obligations, where obligations are tasks and requirements to be fulfilled before, after or together with the enforcement of the authorization decisions.

This paper conducts a literature review of role based access control and obligation related research, and proposes a design for the augmentation of obligations in the context of the RBAC standard. The design is then validated by implementation in the PERMIS RBAC authorization infrastructure. The paper also discusses the possible nondeterminism caused by overlapping authorisations.

1. Introduction

Role based access control systems make access control decisions based on the roles that users hold. The traditional output of the access control decision is either “Granted” or “Denied”, which dictates whether the request is authorised or not. In traditional systems, applications submit authorization requests to a decision making policy decision point (PDP) when users attempt to perform operations on protected resources. Authorization responses, which specify whether the requests are authorized or not, will be produced in reply to the requests.

The conventional decisions are generally passive, and do not provide a way of instructing systems to perform further operations besides granting or denying the user’s request. There are some scenarios where conventional responses of “Granted” and “Denied” do not suffice, where certain operations need to be performed together with the enforcement of the decisions. For example, an authorization response might contain information such as “the request is authorized, and the final transaction result must be posted to the administrator”.

Obligations are requirements and tasks to be fulfilled, which can be augmented into conventional systems to allow extra information to be specified when responding to authorization requests. Administrators can associate obligations with permissions, and require the fulfillment of these obligations when the permissions are exercised. PDPs can produce authorization responses containing the “granted” or “denied” authorization decisions along with the corresponding obligations.

1.1 Contribution

The contributions of this work are five-fold. Firstly, we identify the need for augmenting obligations with RBAC and propose a model for the integration of obligations with the ANSI standard RBAC model. Secondly, we recognize the need to provide obligations for authorization requests which are denied access, and we propose a method for implementing this in a RBAC system. Thirdly, we provided a discussion about the non-determinism caused by overlapped authorization rules in which a user with several active roles may be granted the same permission but with different obligations. We argue that non-determinism is acceptable from a security perspective and will not compromise the system’s security. Fourthly, we provided a specification of the PERMIS security policy that can accommodate obligations, and describe how we have implemented this in the PERMIS authorization infrastructure. The implementation is backward compatible, and does not require any changes to the original applications. Fifthly, we define a Chronicle parameter for obligations which we deem is essential in order to inform the application when to perform the obligations. This feature is currently missing from the XACML specification.

1.2 Organization

The organization of the rest of this paper is as follows. Section 2 presents a literature review of the research on role based access controls and obligations. Section 3 briefly introduces the NIST RBAC core model, and proposes an enhanced RBAC core model
which is augmented with obligations. Section 4 presents the PERMIS RBAC authorization infrastructure, focusing on the structure of the PERMIS policy and the monotonic decision making algorithm. Section 5 describes the incorporation of obligations into PERMIS based on the enhanced RBAC core model in Section 3, and the inclusion of obligations for denied access requests. Section 6 is dedicated to the discussion of overlapping authorization that can cause non-determinism. Section 7 concludes the paper.

2 Related Work

2.1 RBAC

Sandhu et al. [14] identified the motivation of using roles as basic constructs in access control models, and introduced several models of role based access control, expecting the models to be treated as reference models. Sandhu et al. conceptualized role based access control into four different models, the base model, the hierarchical model, the constrained model, and the consolidated model. The hierarchical model and the constrained model are advanced models evolved from the base model, and the consolidated model is a combination of the hierarchical model and the constrained model. The base model associates users with roles, and roles with permissions. Users, being members of roles, acquire all permissions associated with the roles. The hierarchical model enhances the base model by allowing senior roles to acquire the permissions of their junior roles. The constrained model improves the base model by imposing a set of constraints to be satisfied by the base model, thus providing further control over the system. The consolidated model combines both the hierarchical model and the constrained model into a sophisticated model, to meet most of the possible complicated requirements. This work has been partially adopted by the NIST RBAC standard [13, 15] discussed later.

Ferraolo and Kuhn [6] presented a detailed description of the RBAC model, and provided the definition of roles, transactions and a formalization of RBAC. Roles were defined by using a set of transactions, and transactions were a set of high level activities that users could perform. A user had the right to perform a transaction if the transaction was a permitted transaction of his current active role.

Sandhu et al. [13] presented the NIST standard model for role based access control. The general idea of the role based access control model is that permissions are associated with functional roles in organisations, and members of the roles acquire all permissions associated with the roles. Allocation of permission to users is achieved by assigning roles to users. In this way, roles serve as an abstraction to users. In this way, roles serve as an abstraction to permissions, as well as groups of related permissions. Roles are expected to be persistent, thus a RBAC based system mainly needs to manage the role memberships only. In this way, it is expected the manageable and scalability of systems can be improved.

Oppinger et al. [11] proposed a way of implementing role based access control based on Attribute Certificates. Attribute certificates are used as protected tokens to convey attribute information. A commercial application running in Switzerland was highlighted to show that the proposed system was realistic and practical.

Gavrila and Barkley [7] formally specified the role management of a RBAC system, and defined the consistency of a RBAC system using a set of properties. Gavrila and Barkley also showed that, given a consistent RBAC system, performing legitimate management operations maintained the consistency of the system.

2.2 Obligations

Minsky and Lockman [9] present the motivation of associating obligations with privileges, although its main focus is on data integrity instead of security. The idea is that violation of data integrity can be tolerated if the violation can be recovered by an obligation in the foreseeable future. Minsky and Lockman also identified the need for associating obligations with privileges for protecting data integrity and present a model of associating obligations with privileges. An obligation is associated with a privilege, and when an operation is performed, the obligation associated with the privilege which authorizes the operation is activated. Obligations are requirements to be performed by a specific deadline. Failure to fulfill an obligation will incur a sanction. Different types of requirements and sanctions are discussed.

Jonscher [8] suggested incorporating duties into RBAC systems, where duties are tasks that users need to perform. This is very similar to the notion of obligations.

Bettini et al. [3] formalized policies with obligations and provisions, allowing policies to specify actions and conditions to be fulfilled before or after a user exercises the granted privileges. The formalization also provides a reasoning mechanism for systems to deduce the set of provisions and obligations to be fulfilled given a policy.

Bettini et al. [2] proposed a model of specifying obligations and managing obligations. Bettini et al. also presented a discussion of policy refinement based on
obligation fulfillment/defaulting, hierarchical obligations and obligation monitoring, including monitoring in the presence of quantitative temporal constraints.

The Extensible Access Control Markup Language (XACML) [10] is a standardised markup language for policy management and access decisions. Security control rules are specified in a set of policies. Each policy is composed of a set of rules, specifying the authorisation in different situations. Rules have different effects. Positive rules grant authorisation, denoted as the PERMIT effect. Negative rules deny authorisation, denoted as the DENY effect. A rule-combination algorithm will be specified for each policy, to resolve conflicts when different rules have conflicting decisions. A set of obligations can also be associated with a policy, which are actions must be performed by the PEP at the same time as enforcing the authorisation decisions. XACML is not designed based on the Role Based Access Control model, but an XACML RBAC Profile [1] has been developed that provides constructs that can be used to build RBAC policies with XACML.

Ribeiro et al. [12] identified a limited type of obligations, which are obligations with two actions having interdependencies between each other. Ribeiro et al. argued that these obligations can be enforced at the boundary of transactions. Transactions can be committed only when all required obligations have been fulfilled. On checking the fulfillment of obligations, obligation augmented policies are converted into history based policies, and systems check all history in order to have information about the fulfillment of obligations.

3. NIST RBAC and Obligations

NIST proposed a reference model of role based access control which was approved as a standard, published in the document ANSI INCITS 359-2004 [15]. The RBAC reference model is defined in terms of four different model components, including the Core RBAC, the Hierarchical RBAC, the Static Separation of Duty Relations, and the Dynamic Separation of Duty Relations. The Core RBAC specifies the essential elements for the RBAC model which comprise the minimum set of elements. The other three components can be integrated with the Core RBAC component to add more features.

3.1. The Core RBAC

The Core RBAC [15] consists of five basic elements, which are the USERS, ROLES, OPS, OBS, and SESSIONS, and five relations, which are the UA, the PA, the U-S, the S-R, and the PRMS. The model is illustrated in Figure 1.

![Figure 1: The Core RBAC Model](image)

**Figure 1: The Core RBAC Model**

 USERS refers to the set of legitimate users in the system. ROLES is the set of roles existing in the system. OPS is the operations that are recognized by the system, and OBS is the set of objects that are protected by the system. SESSIONS is the set of sessions in the system that are handling users’ requests.

Operations and objects are bound to each other to construct permissions, denoted by PRMS where PRMS \( \subseteq \) OPS \( \times \) OBS. A permission is an approval of performing an operation on a specified target. Users are allocated roles, as specified by the UA relation where UA \( \subseteq \) USERS \( \times \) ROLES, which is the user assignment relation. Permissions are allocated to roles, which are specified by the permission assignment (PA) relation where PA \( \subseteq \) ROLES \( \times \) PRMS. U-S \((s :\) SESSIONS\) \(\rightarrow\) USERS is a mapping of a session onto the corresponding users, and the S-R \((s :\) SESSIONS\) \(\rightarrow\) \(\times\)ROLES is a mapping of a session onto a set of roles.

The authorization decision making function CheckAccess takes as inputs the current session, the requested operation, and the object that is the target of the operation. The CheckAccess function will return a Boolean value as a result to indicate whether the request is authorized or not. According to the RBAC standard, this can be formalized as the following.

\[
\text{CheckAccess}(s, op, obj) = \begin{cases} 
\exists r \in \text{ROLES}, & r \in S-R(U-S(s)) \land ((op, obj) \in \text{PRMS} \land (r, (op, obj)) \subseteq \text{PA}) 
\end{cases}
\]

The inputs to CheckAccess are \(s\), \(op\), and \(obj\), where \(s\) is the current session that requests the authorization, \(op\) is the requested operation, and \(obj\) is the object of that operation. The request permission is identified by the input \(op\) and \(obj\).

The CheckAccess function checks if there exists a role \(r\) mapped from the current session, such that the role \(r\) has been allocated the permission to perform the operation \(op\) on the object \(obj\). If such a role exists, a True value will be returned as the decision. Otherwise, a False value will be returned.
3.2 Augmentation with Obligations

The NIST RBAC model allocates privileges to roles based on the PA relationships, which associates roles with permissions. To accommodate obligations in the RBAC model, roles can be allocated with permissions and obligations, such that every permission allocated to a role is associated with a set of obligations. The same permission allocated to different roles can be associated with the same obligations, or with different obligations.

Figure 2 shows the Core RBAC model with the augmentation of obligations as proposed above. The new model introduces a new basic element to the NIST core RBAC model, the OBLGS, which is the set of valid obligations. These obligations are the tasks that can be fulfilled by the system, and will be associated with the permissions allocated to roles.

A new relation \( \text{OPRMS} \subseteq \text{PRMS} \times 2^{\text{OBLGS}} \) is also introduced into the obligation augmented core RBAC model, which is a relation between permissions (PRMS) and obligations (OBLGS). For \( oprm = (prm, oblgs) \in \text{OPRMS} \), \( oprm \) is an obligation augmented permission that specifies if the permission \( prm \) is exercised, the set of obligations \( oblgs \) shall be fulfilled.

![Figure 2: Obligation Augmented Core RBAC](image)

The PA relation is modified into the form of \( \text{PA} \subseteq \text{ROLES} \times \text{OPRMS} \). To each \( p=(r, oprm) \in \text{PA} \), it states that, the role \( r \) is allocated with the obligation augmented permission \( oprm \).

For example, let \( oprm=((park, car), \{pay, report\}) \), the \( oprm \) indicates that, when this permission is associated with role \( r \), then role \( r \) is allocated the permission \( (park, car) \), but \( r \) must fulfill the obligations \( \{pay, report\} \) when \( r \) exercises the permission \( (park, car) \). This can be interpreted as follows. Users who are members of the role \( r \) are allocated with the permission to park their cars with the obligation that they must report and pay for their parking.

3.3. Rendering Responses with Obligations

With the obligation augmentation, the RBAC authorization function needs to be enhanced to cope with the enhanced permission allocation relation, and it also needs to be modified to produce responses that contain both the Boolean type authorization decision and the associated obligations. Thus the type of the \( \text{CheckAccess} \) function must be changed from

\[ \text{CheckAccess} : (\text{SESSIONS,OPS,OBS}) \rightarrow \text{Boolean} \]

...to

\[ \text{CheckAccess} : (\text{SESSIONS,OPS,OBS}) \rightarrow (\text{Boolean, } 2^{\text{OBLGS}}) \]

The new \( \text{CheckAccess} \) function allows users to request permissions based on the session, the requested operation, and the targeted object. The new \( \text{CheckAccess} \) function will respond with a Boolean authorization decision and a set of associated obligations. The Boolean authorization decision indicates whether the request is authorized or not. The set of associated obligations are the tasks that must be fulfilled before, after or together with the enforcement of the authorization decision.

The new reasoning algorithm of \( \text{CheckAccess} \) is as follows. It checks if the requested permission has been allocated to any of the roles from the session. If it has not been allocated to any of the roles, then return a decision of False with an empty set of obligations. If it has been allocated to any of the roles, then return a decision of True with a set of obligations that are combined from all the associated obligations. The combination of obligations is described below, and the choice of which combination algorithm to use is left to the application to decide.

In our model, an obligation comprises two components: the obligated action, and a Chronicle parameter that says when the obligated action must take place. The Chronicle parameter can take one of three values: \( \text{Before, After or With} \). A chronicle value of \( \text{After} \) indicates that the obligation should be enacted only after the user’s access request has been enforced (either granted or denied). A chronicle value of \( \text{Before} \) indicates that the obligation should take place before the user’s request is enforced. A chronicle value of \( \text{With} \) indicates that the obligation and the user’s request should be enforced as an atomic action. It is up to the RBAC policy writer to determine which Chronicle value to use with each obligation. Note that the XACML policy language does not have a Chronicle parameter, since it implicitly assumes the semantics of the \( \text{With} \) value.
For example, say the PA relation contains the following rules:

Rule 1: \( r_1, (park, car), (pay: After) \)
Rule 2: \( r_2, (park, car), (report: Before) \)

Rule 1 allocates the permission \((park, car)\) to the role \(r_1\) with the obligation to pay After parking, and rule 2 allocates the permission \((park, car)\) to the role \(r_2\) with the obligation to report Before parking. If a user with only role \(r_1\) requests permission \((park, car)\), the request will be denied and the CheckAccess function will return \((false, null)\). The false value indicates the request is denied, and the null indicates an empty set of obligation is associated with the decision. (We will discuss allocating obligations to denied decisions in Section 5.) If a user with the roles \(r_1\) and \(r_2\) requests permission to \((park, car)\), both rule 1 and rule 2 can authorize the request. But rule 1 and rule 2 contain different obligations. An obligation combination algorithm must be specified for the CheckAccess function to produce a set of obligations based on these two sets of obligations. Each application will configure the CheckAccess function with its own appropriate obligation combining algorithm. Possible obligation combination algorithms are:

1. Union. The Union combination algorithm calculates a union of all the obligation sets.
2. Any. The Any combination algorithm randomly selects one of the obligation sets.
3. First-Applicable. The First-Applicable combination algorithm selects the set of obligations in the first applicable rule. The first applicable rule depends on the order of applying the PA rules.

The produced set will be returned to the requester as part of the authorization response by the CheckAccess function. Notice that, the Any combination algorithm is a non-deterministic algorithm. In other words, with the Any combination algorithm, multiple invocations of the same request might be returned responses that contain different obligations.

4. PERMIS RBAC Authorization Engine

PERMIS [4] is a role based access control authorisation infrastructure. It provides facilities for privilege management, trust management, and decision making. PERMIS uses security policies to define the RBAC model. The security policies [5] consist of several sub policies, which specify the legitimate set of users, roles, actions, targets, and the permission allocation to roles, respectively. Note that actions and targets are terms used by PERMIS to refer to operations and objects in the core RBAC model. The sub policy, TargetAccessPolicy (TAP), is the policy that specifies the permission allocation to roles in PERMIS.

![Figure 3: PERMIS Target Access Policy (TAP)](image)

4.1. Target Access Policy

The structure of the Target Access Policy is illustrated in Figure 3. The target access policy comprises of a set of target access rules. Each target access rule associates a set of permissions to access a specified resource to a set of roles under certain conditions.

Let \( R \) be the set of all roles that are defined in the system, \( A \) is the set of all legitimate actions, \( T \) is the set of all legitimate targets, and \( C \) is the set of valid conditions. The target access policy defines a relation \( K \), where \( K \subseteq 2^R \times 2^A \times 2^T \times 2^C \), that represents the allocation of permissions to roles regarding to different targets subject to various conditions.

Target access rules are in the form of \((r, a, t, c)\), where \(r, a, t,\) and \(c\) are subsets of \(R, A, T,\) and \(C\) respectively. The target access rule \((r, a, t, c)\) allocates permissions of performing all actions in \(a\) over any target in \(t\) to users who hold all the roles in \(r\), if the condition \(c\) can be satisfied.

Let us suppose there is a policy, whose parameters specify that there are only two roles, Manager and Staff, and only one target to be protected, the phone. There is one action, dial that can be performed on the phone. The policy parameters may be as defined as follows. \( R=\{Manager, Staff\}, A=\{dial\}, T=\{phone\}, \) and \(C=\{}\). The target access policy allocates the dial permission to the Manager role. The target access policy contains only one rule, which is \((\{Manager\},\{dial\},\{phone\},\{}\). As the target access policy has not allocated any permission to the role Staff, Staff is not allowed to dial the phone.
The TargetAccessPolicy specifies the permission allocations to roles, and it contains only allowed authorization rules. The implicit default rule is Deny All Except, which is a prudent security policy. Thus the TargetAccessPolicy is monotonically increasing, as each new rule only serves to increase the set of allowed permissions.

4.2. Authorization Algorithm

On requesting authorisation, an application submits an authorisation request to PERMIS. PERMIS will produce an authorisation response, containing the corresponding authorisation decision. The application is expected to enforce the authorisation decision in order to protect the system.

Authorization requests describe a situation where an authorisation decision is needed. Authorisation requests are in the form of \((r,a,t,c)\), where \(r\) is a set of roles, of whom the user is a directly or indirectly a member; \(a\) is the action the user is requesting to perform; \(t\) is the target of the requested action.

The authorisation response, computed by the authorisation function, is a Boolean value, which is either true or false. The authorisation response dictates PERMIS’s decision towards the request of authorisation. The authorisation response is computed based on the authorisation function, which is defined as follows.

\[
\text{auth}(r,a,t,c) =
\begin{cases}
\text{True} & \text{iff } \exists (r,a,t,c) \in K \text{ and } t \subseteq r \\
\text{False} & \text{otherwise}
\end{cases}
\]

The authorisation function takes an authorisation request as the input, and produces an authorisation decision as output. The request \((r,a,t,c)\) will be authorised only when there exists a target access rule \(l=(r,a,t,c)\) in the target access policy, such that the following conditions are met.

1. The roles held by the user, represented by \(r\), are a superset of the required roles \(r\) in the target access rule \(l\).
2. The requested action \(a\) is contained in \(a\) of \(l\). The set \(a\) of \(l\) specifies the set of actions allowed to be performed. Authorised users are allowed to perform any one of the action.
3. The target \(t\) is contained in \(t\) of \(l\). \(t\) of \(l\) is the set of legitimate targets for the actions allowed by \(a\). The target \(t\) must be one of the legitimate targets specified by \(t\).
4. The current context satisfies the condition \(c\) of \(l\).

If the above rule \(l\) does not exist in \(K\), the authorisation request is denied. It is obvious that the above authorization function is a monotonically increasing function in the sense that, allocating new roles to users or adding new permission allocation rules will only result in the possible conversion of some previously denied requests into authorized requests, but it will not convert any previously authorized requests into denied requests. This allows PERMIS to be optimized in a way that it can stop its reasoning when it encounters the first rule that authorizes the current request, and it can return a granted decision to the PEP. This is an advantage of PERMIS over XACML, as XACML contains Deny rules as well as Granted rules. Consequently XACML needs to evaluate all policies and rules if it does not use the combination algorithm such as “First-Applicable”.

5. Obligations for PERMIS

This section proposes a design to augment obligations for the PERMIS RBAC authorization engine using the model described in Section 3. The proposed design modifies the PERMIS policy to attach obligations to permissions. For the purpose of providing obligations when authorization requests are denied, a new sub policy, the denial obligation policy, is introduced into the current PERMIS policy. The denial obligation policy specifies obligations to be fulfilled when authorization requests are denied. The authorization algorithm of the proposed design is formalized and presented, showing the exact authorization reasoning process of PERMIS.

5.1 Obligation Enhancement

The incorporation of obligations into PERMIS will need to change the syntax of PERMIS’s security policies, to allow system administrators to specify obligations when composing security policies. An obligation enabled PERMIS policy will associate a set of obligations to each target access rule in the target access policy.

The enhancement of the target access policy for the associated set of obligations is shown in Figure 4. Each target access rule is associated with a set of obligations. The set can be empty. With the obligation augmentation, the target access Policy can be reformulated as \(K \subseteq 2^r \times 2^a \times 2^t \times 2^c \times 2^o\), where \(O\) is the set of valid obligations. A target access rule \(k \in K\) is a tuple of five, denoted as \(k=(r,a,t,c,o)\), where \(r,a,t,\) and \(c\) are sets of roles, actions, targets, and conditions respectively. \(o\) is a set of obligations, which will be returned as part of the authorization response when \(k\) grants authorization to the request.
5.2 Obligations on Denial

PERMIS’s RBAC model describes a closed world, where roles have no permissions initially and all actions are denied by default. A role is authorized to perform an action on an object only when the corresponding permission is allocated to the role. Policies thus contain only positive rules and criteria. This results in the fact that there are no specific rules that reject authorisation requests, therefore it is not possible to find a rule that causes the rejection of an authorisation request.

Obligations on denial are obligations that are returned with negative authorisation decisions. As the RBAC model fails to provide a rule for the rejection of a request, we augment the RBAC model with a Denial Obligation Policy (DOP). The DOP has the same structure as the TAP. The difference between the DOP and the TAP is that the TAP specifies the access rules for roles under different condition, whilst the DOP specifies the obligations when specific authorisation requests are not to be authorised.

The DOP can be formulated as $D \subseteq 2^r \times 2^a \times 2^t \times 2^c \times 2^o$. The DOP contains multiple denial obligation rules. Let $d=(r,a,t,c,o)$ be a denial obligation rule in the DOP. $d$ specifies that, given any request that satisfies $d$, then $o$ is the set of obligations to be returned when the request is denied. The satisfaction relation is specified as follows. Suppose the user $u$ with a set of roles $r_u$ requests to perform action $a_v$ on the target $t_v$. The request satisfies the DOP rule $d$ if and only if the following conditions are met.

1. The set of roles $r_u$ is a superset of $r$.
2. The action $a_v$ is contained in the set $a$.
3. The target $t_v$ is contained in the set $t$.
4. The current context satisfies the conditions in $c$.
5. The request is denied.

For example, if $r$ is empty, and $a$ is the set of actions available on top secret resource $t$, then any user who is denied any action on the top secret resource will cause the obligation $o$ to be enacted, such as sending a message to a log and notifying the security officer.

A formal specification is that, the request $q = (r_u,a_v,t_v)$ satisfies the target access rule $d=(r,a,t,c,o)$ if and only if $r \subseteq r_u \land a_v \in a \land t_v \in t \land c= \text{true}$.

5.3 Authorization Decision Making

With the augmentation of obligations for both positive and negative authorisations, the authorisation function has been changed as shown in Figure 5. In response to every authorisation request, PERMIS will first test the request against the TARs. When a TAR is matched, authorisation is granted and the obligation in the matched TAR will be returned together with the decision. When no TAR is matched, authorisation is denied. PERMIS will continue to test the request against the DOP. If a DOP rule is matched, the obligation of the DOP rule will be returned as the obligation for denial. Otherwise no obligation is required.

A more formal specification of the authorisation function $auth: 2^r \times 2^a \times 2^t \rightarrow \text{BOOLEAN} \times 2^o$ can be represented as the followings.

$$auth_u(r_u,a_v,t_v) =$$
6. Overlapped Authorization

Overlapped rules are target access rules that give the same permissions i.e. authorise the same actions to the same target resources, but to different roles and/or with different conditions attached. Overlapped rules may associate different obligations to the same permissions, leading to multiple possible authorisation responses which have different obligations but the same authorisation decision. For example, let target access rules $l_1$ and $l_2$ defined as follows.

$$l_1 = \{\text{[student],[buy],[discountedTicket],}$$

$$\{\text{[Record on Log1:After]}\}$$

$$l_2 = \{\text{[disabled],[buy],[discountedTicket],}$$

$$\{\text{[Record on Log2:After]}\}$$

Rule $l_1$ specifies that students can buy discounted tickets with the obligation that the activity needs to be recorded on Log1 after purchase, and rule $l_2$ states that disabled persons can buy discounted tickets with the obligation that the activity needs to be recorded on Log2 after purchase. Both $l_1$ and $l_2$ grant the same permission i.e. authorise a role to buy discounted tickets. When a student, who is also registered as disabled, requests authorization to buy a discounted ticket, there are several possible responses. When $l_1$ is applied, the request is authorised with the obligation to "Record on Log1 afterwards". When $l_2$ is applied, the request is authorised with the obligation to "Record on Log2 afterwards". This may be confusing in some circumstance, as administrators may have preference for different rules, and would like the system to be deterministic as to which rule (and obligation) takes precedence when a subject passes multiple rules.

The obligation combining algorithm should be used in situations like this in order to determine the correct or best set of obligations to be returned. Unfortunately the best combining algorithm for one policy is not necessarily the best for a different policy. In the policy above, it might be acceptable to use the Union combining algorithm and log the request of a disabled student in both Log1 and Log2. But if the obligations were to give specific discounts instead of write to logs, e.g. students get 5% discount and disabled people get 10% discount, it might not be appropriate that disabled students receive both discounts.

$$\begin{align*}
& (True,o) \quad \text{iff } \exists ((r,a,t,c,o) \in K (r \subseteq a_q \land a_q \in a \land t_i \in t \land c = true)) \\
& (False,o) \quad \text{else if } \exists ((r,a,t,c,o) \in D \land r \subseteq a_q \land a_q \in a \land t_i \in t \land c = true)) \\
& (False,o) \quad \text{otherwise}
\end{align*}$$

It can be argued that overlapping rules are of equivalent priority, and they all provide the same security for the system, since a user must have the requisite roles and satisfy the necessary conditions in order to be granted access by each rule. Thus from a security perspective it is acceptable to choose one rule at random from the overlapping rules without applying any criteria to the selection of which role is used to authorise the action the user requests, since the user does possess whichever role is used. From this perspective, we assume the overlapped rules are equivalent alternatives of each other instead of redefinitions, and consider any one of the alternatives is suitable for the authorization decision since each provides the same security for the system. As a result, PERMIS uses the Random combination algorithm and will select one TAR randomly from all applicable TARs without specific preference or order.

If we consider the previous example of obligations to write to different logs, does it really matter from a security perspective whether the disabled student’s purchase was recorded in Log1 or Log2? Or if different discounts were obligated for students and disabled people, does it really matter from a security perspective whether a disabled student gets 5% or 10% discount? If a disabled student objects to only getting 5% discount, then they can simply not admit to being a student, and then will gain the 10% discount available to disabled people. Either way, the security of the system has not been compromised.

Treating all overlapped rules equally also avoids the computation task of selecting the preferred role from the multiple available roles, and helps PERMIS to run more efficiently and to provide a better response time when handling authorisation requests. When imposing preference or priority on TARs, PERMIS would have to test all the TARs before it could make a decision, to make sure that the result is sound and complete. Further, PERMIS may need to sort all the applicable TARs according to the preference and the priority, which will incur extra computation and resource consumption, leading to longer response times.

To some extent, PERMIS’s treatment of the non-determinism of obligations is also in line with XACML’s policy combining rules. XACML is capable of combining multiple decisions from different policies and policies sets, each of which might return its own decision and obligations. The decision combining is specified by policy combining rules, some of which are not deterministic. It is argued by XACML that these non-deterministic combining algorithms provide better performance and demand less resources, and these are wanted by a significant number of applications. In situations where non-determinism is not acceptable,
XACML suggests users to use deterministic combining algorithms.

7 Conclusion

7.1. Conclusion

This paper discusses the augmentation of obligations with RBAC models. The purpose of the augmentation is to provide more active and flexible security controls for RBAC models. We propose a model for augmenting obligations with role based access controls, which modifies the association between roles and permissions in a way that roles are associated with permissions and obligations. When a permission of a role is to be exercised, the associated obligations should be fulfilled. We also describe how we have implemented the model in PERMIS.

The RBAC model does not have a concept of negative permissions; however we may need to attach obligations to certain denied decisions. These obligations must be fulfilled only when authorization requests are denied under certain specific situations. We describe how we have implemented this in PERMIS.

The non-determinism that is caused by overlapping authorization rules is also discussed. We argue that overlapping rules are alternatives and equivalent to each other from a security perspective. Furthermore, eliminating the non-determinism will incur extra computation costs. Thus we consider the non-determinism is a reasonable implementation option.

7.2 Future Work

Currently the proposed design of augmenting obligations into PERMIS has been implemented except the obligations on denial feature. Obligations have been implemented in conformance to XACML obligations. Future work will be the testing and trial deployment of the current implementation in grid applications and testing the plug and play compatibility of the PERMIS PDP with an XACML PDP. We are also integrating the PERMIS PDP into a role based messaging infrastructure so that policies with obligations can be carried around with the email messages to which they apply.

As this paper focus on the computation of obligations, and deliberately leaves the fulfillment of obligations for separate work, we still need to investigate the necessary infrastructures for applications to fulfill obligations. We also envisage that the monitoring of the obligation fulfillment is an important part of security infrastructures.

ACKNOWLEDGEMENTS

The authors would like to thank Nexor which is sponsoring the PhD research of Gansen Zhao and EPRSC which partly funded the work under GR/S69061/02.

References

