Obligation 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 used 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 not. One of the potential improvements for role based access control is the augmentation of obligations, where obligations are tasked and requirements to be fulfilled together with the enforcement of authorization decisions.

This paper conducts a comprehensive literature review about role based access control and obligation related research, and proposes a design of the augmentation of obligations in the context of RBAC standard. The design is then further consolidated in the PERMIS RBAC authorization infrastructure. Details of incorporating obligations into the PERMIS RBAC authorization infrastructure are given. This paper also discusses the possible nondeterminism caused by overlapped authorisation.

1. Introduction

Role based access control systems make access control decisions based on the roles that users hold. Traditional output of the access control decisions are “Granted” and “Denied”, which dictate whether the requests are authorised or not. In conventional systems, applications submit authorization requests to decision making servers when users attempt to perform operations on protected resources. Authorization responses, which specifies 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 ways of instructing systems for further operations besides ``Granted'' and ``Denied''. 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. Such as, 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 extras information to be specified when responding to authorization requests. Administrators can associated obligations with permissions, and required the fulfillment of the obligations when the permissions are exercised. Systems can produce authorization responses containing the authorization decisions, either “granted” or “denied”, and the corresponding obligations.

The organization of the rest of this paper is as the followings. Section 2 presents a comprehensive literature review of the research of role based access control and the research of 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 PERMIS policy and the monotonic decision making algorithm. Section 5 proposes a design for incorporating obligations into PERMIS based on the enhanced RBAC core model in Section 3. Section 6 dedicates to the discussion of overlapped authorization that can cause nondeterminism. 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 model evolved from the based model, and the consolidated model is a combination of the hierarchical model and the constrained model. The base model associated users with roles, and roles with permissions. Users, being members of roles, acquired all permissions associated with the roles. The hierarchical model enhanced the base model by allowing senior roles to acquire permissions of their junior roles. The constrained model improved the based model by imposing a set of constraints to be satisfied by the base model, thus providing further control over the system. The consolidated model combined 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 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 of 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 manageability and scalability of systems can be improved.

Oppliger 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 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*] presented the motivation of associating obligations with privileges, thought it's mainly focusing 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 presented a model of associating obligations with privileges. An obligation is associated with a privileges, and when an operation is performed, the obligation associated to the privilege which authorizes the operation is activated. Obligations are requirements to be performed by a specific deadline. Failure of the fulfilling an obligation will incur a sanction. Different types of requirements and sanctions have been discussed.

Jonscher [*8*] suggested to incorporate duties into RBAC systems, where duties are tasks users need to be performed, which is very similar to the notion of obligation.

Bettini et al [*3*] formalized policies with obligations and provisions, allowing policies to specify actions and conditions to be fulfilled before or after user' exercising of 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 discusson on the topics of policy refinement based on obligation fulfillment/defaulting, hierarchical obligations and obligation monitoring, including monitoring in the presence of quantitative temporal constraints.

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 PERMIT effect. Negative rules deny authorisation, denoted as 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 when enforcing the authorisation decisions. XACML is not designed based on Role based Access Control model, but an XACML RBAC Profile [*1*] has been developed that provides constructs that can be used to build RBAC systems.

Ribeiro et al. [*12*] identified a limited type of obligations, which are obligations with two actions having interdependencies on between each other. Ribeiro et al. argued that these obligations can be enforced with 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 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 RBAC model, which comprise of 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] is consisted 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 can be illustrated by Figure 1.

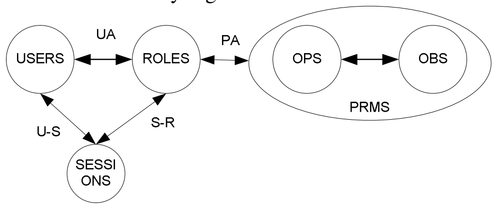

Figure 1: The Core RBAC

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. The SESSIONS is the set of sessions in the system that are handling business.

Operations and objects are bound to each other to construct permissions, denoted by PRMS where PRMS $\subseteq$ OPS × OBS. A permission is an approval of performing an operation on a specified target. Users are allocated with roles, as specified by the UA relation where UA $\subseteq$ USERS × ROLES, which is the user assignment relation. Permissions are allocated to roles, and it is specified by the permission assignment (PA) relation where PA $\subseteq$ ROLES × PRMS. U-S (s : SESSIONS) → USERS is a mapping of a session onto the corresponding users, and the S-R (s : SESSIONS) → 2^{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 followings.

$$CheckAccess(s,op,obj) = \exists r \in ROLES,\ r \in S\text{-}R(U\text{-}S(s)) \quad ((op,obj) \in PRMS \quad (r, (op,obj)) \in PA)$$

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 the operation op targets at. 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 of Obligations

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

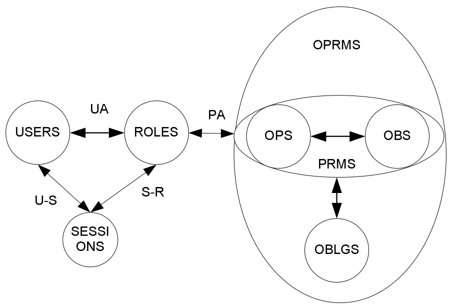

Figure 2: Obligation Augmented Core RBAC

The *PA* relation is modified into the form of $PA \subseteq ROLES \times OPRMS$. To each $p=(r,oprm) \in PA$, 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, the 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 member of the role *r* are allocated with the permission of parking their cars with the obligations that they must report and pay for their parking.

3.3. Rendering Responses with Obligaitons

With the obligation augmentation, the RBAC authorization function shall be enhanced to cope with the enhanced permission allocation relation, and it shall also be modified to produce response that contain both the Boolean type authorization decisions and the associated obligations. Thus type of the *CheckAccess* function shall be changed from

CheckAccess : (SESSIONS,OPS,OBS) ->Boolean

to

CheckAccess : (SESSIONS,OPS,OBS) ->(Boolean, 2^{OBLGS}*)*

The new *CheckAccess* function allows users request permissions based on the session, the requested operation, and the targeted object. The new *CheckAccess* function will response with a Boolean authorization decision and the 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 together with the enforcement of the authorization decision.

The new reasoning algorithm of *CheckAccess* is as follows. It checks if there is the requested permission has been allocated to any subset of roles of the set of roles mapped 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 obligation. If it has been allocated to the roles, then return a decision of True with a set of obligations that are combined from all the associated obligations. The combination algorithm of obligations is left to be decided according to application requirements.

For example, the current session s can be mapped to a set of roles $\{r_1,r_2\}$. And the PA relation contains the following rules

Rule1 $\{r_1,((park,car),pay)\}$

Rule2 $\{r_2,((park,car),report)\}$

Rule1 allocates the permission *(park,car)* to the role r_1 with the obligation of pay, and *rule2* allocates the permission *(park,car)* to the role r_2 with the obligation of report. If a user with only the role r_3 requests the permission of *(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. If a user with the roles of r_1 and r_2 requests for the permission of *(park,car)*, both *rule1* and *rule2* can authorize the request. But *rule1* and *rule2* contain different obligations. A combination algorithm must be specified for the *CheckAccess* function to produce a set of obligations based on these two set of obligations. Possible combination algorithm can be

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, the same request might be replied with 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. It's not this paper's position to provide a comprehensive specification of the PERMIS authorization infrastructure. Interested readers are referred to other publication of PERMIS, including [4], and [5]

PERMIS uses security policies to define the RBAC model. The security policies [5] are consisted 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. The sub policy, TargetAccessPolicy, is the policy that specifies the permission allocation to roles in PERMIS.

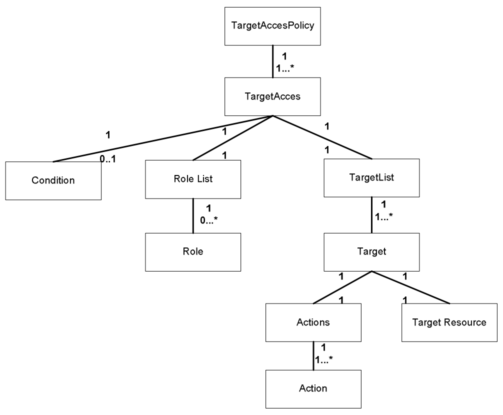

Figure 3: PERMIS Target Access Policy

4.1. Target Access Policy

The structure of 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 roles under a certain conditions.

Let R is 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.

*T*arget 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 suppose there is a policy, which specifies 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 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 shall contains only one rule, which is (*{Manager},{dial},{phone},{}*). As the target access policy has not allocatedd the 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 possible authorization rules. Thus the TargetAccessPolicy is considered as monotonic.

4.2. Authorization Algorithm

On requesting authorisation, an authorisation request is submitted to PERMIS. PERMIS will produce an authorisation response, containing the corresponding authorisation decision. The application enforces the authorisation decision to protect the system.

Authorization requests describe the situation where the authorisation is needed. Authorisation requests are in the form of $<r_q,a_q,t_q>$, where r_q is a set of roles, of whom the user is a directly or indirectly member; a_q is the action the user requests the authorisation to perform; t_q is the target of the requested action. The interpretation of an authorisation request $<r_q,a_q,t_q>$ is that, a user holding a set of roles as r_q, requests to perform the action a_q over the target t_q.

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

$$auth(r_q,a_q,t_q) = \begin{cases} True & \text{iff } \exists\ (r,a,t,c) \in K\ (r \subseteq r_q \wedge a_q \in a \\ & \wedge\ t_q \in t \wedge c = true) \\ False & otherwise \end{cases}$$

The authorization function takes an authorisation request as the input, and produces an authorisation decision. The request (r_q,a_q,t_q) 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 hold by the user, represented by r_q, is a superset of the required roles r in the target access rule l.
2. The requested action a_q 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_q is contained in t of l. t of l is the set of legitimate targets for the actions allowed by a. The target t_q 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 monotonic function in the sense that, allocating new roles to users or adding new permission allocation rules will only result in the possible conversion some of 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 the reasoning on the encounter of the first rule that authorizes the current request and produce a granted decision to the PEP. This is also an advantage of PERMIS over XACML, as XACML needs to evaluate all policies and rules if it does not use the combination algorithm such as "First-Applicable" etc.

5. Obligations for PERMIS

This section proposes a design to augment obligations for the PERMIS RBAC authorization engine. The proposal design modifies PERMIS policy to attach obligations to permissions, thus exercising permissions will be required to fulfill the associated obligations. For the purpose of providing obligations when authorization request 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. 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 of obligations is shown in Figure 4. Each target access rule is associated with a set of obligations. The obligation 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.

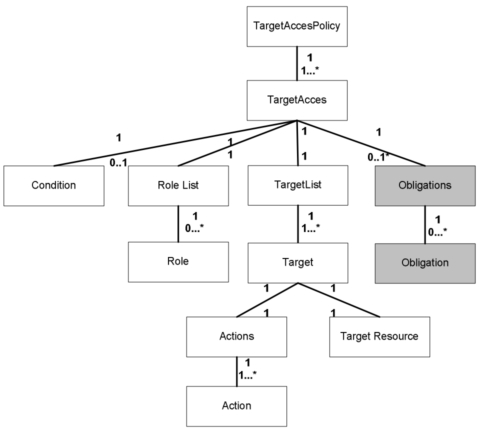

Figure 4: PERMIS Target Access Policy with Obligation

5.2 Obligations on Denial

PERMIS's RBAC model describes a close 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 requests.

Obligations on denial are the obligations that are returned with negative authorisation decisions. As 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, while the DOP specifies the obligation when authorisation requests are not authorised.

The DOP $D \subseteq 2^R \times 2^A \times 2^T \times 2^C \times 2^O$. 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 of a user u, if the request stratifies d, then o can be used as the set of obligations when the request is denied by the policy. The satisfaction relation can be further specified as follows. Suppose the user u with a set of trusted role r_u, requests to perform action a_u on the target t_u. 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_u is contained in the set a.
3. The target t_u 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 $q = (r_u, a_u, t_u)$ satisfies the target access rule $d=(r,a,t,c,o)$ if and only if $r \subseteq r_u \wedge a_u \in a \wedge t_u \in t \wedge c = true$.

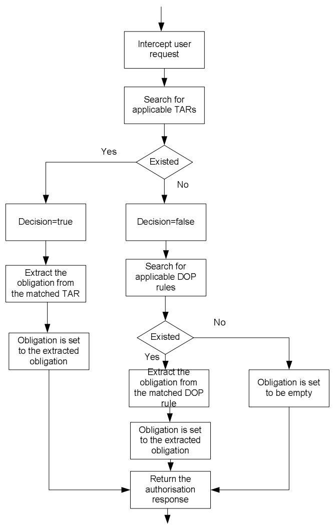
Intercept user request
Search for applicable TARs
Existed
Yes
No
Decision=true
Decision=false
Extract the obligation from the matched TAR
Search for applicable DOP rules
Obligation is set to the extracted obligation
Existed
No
Yes
Extract the obligation from the matched DOP rule
Obligation is set to be empty
Obligation is set to the extracted obligation
Return the authorisation response

Figure 5: Authorization Decision Making

5.3 Authorization Function

With the augmentation of obligations for both positive and negative authorisation, the authorisation function has been changed.

Figure 5 shows the control flow of the decision making of PERMIS after PERMIS is augmented with obligation and denial obligation. 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 $auth$: $2^R \times 2^A \times 2^T$ -> $BOOLEAN \times 2^O$ can be represented as the followings.

$$auth_o(u_q,a_q,t_q) = \begin{cases} (True,o) & \text{iff } \exists\ ((r,a,t,c,o) \in K\ (r \subseteq u_q \wedge a_q \in a \wedge t_q \in t \wedge c = true)) \\ (False, o) & \text{elseif } \exists\ (\ (r,a,t,c,o) \in D \wedge r \subseteq u_q \wedge a_q \in a \wedge t_q \in t \wedge c = true)) \\ (False, \phi) & otherwise \end{cases}$$

6. Overlapped Authorization

Overlapped rules are target access rules that can authorise the same actions but to different roles and/or with different conditions attached. Overlapped rules may associate different obligations to the same actions, 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. *({student},{buy},{discountedTicket}, {},{Record on Log1})*

l_2. *({disabled},{buy},{discountedTicket}, {},{Record on Log2})*

Rule l_1 specifies that, students can buy discounted tickets with the obligation that the activity needs to be recorded on Log1, and rule l_2 states that, disabled persons can buy discounted tickets with the obligation that the activity needs to be recorded on Log2. Both l_1 and l_2 authorise to buy discounted tickets. When a student, who is also registered as disabled, requests authorization for buying discounted tickets, there are several possible responses. When l_1 is applied, the request is authorised with the obligation of ``Recorded on Log1". When $r_2$ is applied, the request is authorised with the obligation of ``Recorded on Log2". This may be confusing in some circumstance, as administrators may have preference over different rules, and would like the system to be deterministic as to which rule (and obligation) takes precedence when a subject passes multiple rules.

PERMIS argues that overlapped rules are of equivalent priority, and they all provide the same security for the system. Thus it is acceptable to choose one role from the overlapped roles without applying any criteria to the selection of the role to authorise the action the user requests. From this perspective, we take the overlapped roles take equivalent alternatives of each other instead of redefinitions, and consider any one of the alternatives is suitable for the authorization and 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.

Treating all overlapped roles equally also avoids the computation task of selecting the preferred role from the overlapped 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 can 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 for sure incur extra computation and resource consumption, leading to longer response times.

To some extend, PERMIS's treatment of the nondeterminism is also in line of XACML. XACML is capable of combining multiple decisions of different policies and policies sets, each of which might return its own decision and obligations. The decision combining is specified by policy combining algorithms, some of which are not deterministic. It is argued that, these non-deterministic combining algorithms provide better performance and demand less resources, and these are wanted by a significant amount of applications. In situations where nondeterminism 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. The discussion is mainly based on the PERMIS's implementation of RBAC model. We envisage the discussion can be easily generalized to implementations that comply with the RBAC standard.

This paper proposes a design for augmenting obligations with role based access control. The proposed design modifies the association between roles and permissions in the way that, roles are associated with permissions and obligations. When a permission of a role is to be exercised, the associated obligations shall be fulfilled.

Negative obligations to be fulfilled when authorization requests are denied are also defined in the proposed design. Negative obligations must be fulfilled only when authorization requests are denied and the requests fall in some specific situations as defined by the policies.

The nondeterminism that is caused by overlapped authorization is also discussed. We argue that overlapped rules are alternatives and equivalent to each other, all of which are suffice to provide the expected security. Further, eliminating the nondeterminism will incur extra computation cost as well as administration cost. Thus we consider the nondeterminism as a reasonable phenomenon. .

The security policy format of PERMIS has been modified to accommodate the association of obligations in the allocation of permissions to roles, and a new sub policy, Obligations on denial. These changes allow security administrators to specify obligations for both the situations of granting authorisation and denying authorization.

7.2 Contributions

The contributions of this work are six-folded. Firstly, we identified the need of augmenting obligations with RBAC and proposed a model for the integration of RBAC and obligations. The proposed model is capable of providing obligations for the situation of authorisation granted and the situation of authorisation denied. Secondly, we provided a discussion on the nondeterminism caused by overlapped authorisation in the proposed model. We considered that nondeterminism is acceptable and will not compromise the system security. Thirdly, we provided a specification of the security policy that can accommodate obligations, and defined an interface for consulting PERMIS decision engine for authorization response. Fourthly, we implemented the proposed design into PERMIS authorization infrastructure. The implementation is backward compatible, and does not require any changes to the original applications. Fifthly, the proposed design and the implementation can be considered as an improvement to PERMIS, so that PERMIS can be an alternative to XACML in most of the scenarios. Lastly, the obligation definition is XACML compatible, which will allow the authorization response to be easily converted to XACML response context. It is envisaged that, XACML PEPs can easily switch between XACML and PERMIS.

7.3 Future Work

Currently the proposed design of augmenting obligations into PERMIS has been partially finished, except the obligations on denial feature. The future work of this research will be the testing and trial deployment of the current implemented features. User feedbacks will be reviewed and revision of the current design might be considered.

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

ACKNOWLEDGEMENTS

The author would like to thank Nexor who is sponsoring the Gansen Zhao's research.

References

[1] Anne Anderson. OASIS Standard: Core and hierarchical role based access control (RBAC) profile of XACML v2.0, February 2005.
[2] C. Bettini, S. Jajodia, X. Wang, and D. Wijesekera. Obligation Monitoring in Policy Management. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, 2002.
[3] Claudio Bettini, Sushil Jajodia, Xiaoyang Sean Wang, and Duminda Wijesekera. Provisions and Obligations in Policy Management and Security Applications. In *VLDB*, 2002.
[4] David Chadwick and Alexander Otenko. The PERMIS X.509 Role Based Privilege Management Infrastructure. *Future Generation Computer System*, 19(2):277--289, 2003.
[5] D.W. Chadwick and A. Otenko. RBAC Policies in XML for X.509 Based Privilege Management. In M. A. Ghonaimy, M. T. El-Hadidi, and H.K. Aslan, editors, *Security in the Information Society: Visions and Perspectives: IFIP TC11 17th Int. Conf. On Information Security (SEC2002), Cairo, Egypt*. Kluwer Academic Publishers, May 2002.
[6] David Ferraiolo and Richard Kuhn. Role-based Access Control. In *Proceedings of 15th National Computer Security Conference*, 1992.
[7] S. Gavrila and J. Barkley. Formal Specification for Role Based Access Control User/Role and Role/Role Relationship Management. In *Third ACM Workshop on Role-Based Access Control*, 1998.
[8] Dirk Jonscher. Extending access controls with duties - realized by active mechanisms. In *Database Security VI: Status and Prospects*. North-Holland, 1993.
[9] Naftaly H. Minsky and Abe D. Lockman. Ensuring integrity by adding obligations to privileges. In *ICSE '85: Proceedings of the 8th international conference on Software engineering*, 1985.
[10] OASIS XACML TC. XACML 2.0 Core: eXtensible Access Control Markup Language (XACML) Version 2.0, Oct, 2005. Available on http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml#XACML20.
[11] Rolf Oppliger, G unther Pernul, and Christine Strauss. Using attribute certificates to implement role-based authorization and access controls. In S. Teufel K. Bauknecht, editor, *Sicherheit in Informationssystemen (SIS 2000)*, pages 169--184, Zurich, 2000.
[12] Carlos Ribeiro, Andre Zuquete, and Paulo Ferreira. Enforcing Obligation with Security Monitors. In *ICICS '01: Proceedings of the Third International Conference on Information and Communications Security*, 2001.
[13] R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST Model for Role Based Access Control: Towards a Unified Standard. In *5th ACM Workshop on Role Based Access Control*, July 2000.
[14] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2), 1996.
[15] ANSI INCITS 359-2004. Role Based Access Control. American National Standards Institute, Inc., February 2004.

Formatted: Italian Italy