Supporting Authorization Reasoning Based on Role and Resource Hierarchies in an Ontology-Enriched XACML Model

Ha Duc Son Van, Tuan Anh Dang, and Tran Khanh Dang

Abstract—RBAC is an excellent model in security domain. In which, users are not assigned to permissions directly but through their roles. Therefore, permissions of individual users are managed by assigning these users to appropriate roles which are quite stable. Besides, RBAC also supports role hierarchy to reduce the number of authorization policies. However, in organizations those have a large number of roles or roles changed frequently, using one role hierarchy makes the maintenance process become more complicated. Moreover, because RBAC does not support resource hierarchy, the number of policies may be very large for organizations which have many different types of resources. To overcome these drawbacks, we propose a new model to express role and resource hierarchies. These hierarchies are implemented by OWL. We show how to support the NIST standard for RBAC in our model. We also extend XACML model to support reasoning ability by defining new functions that use reasoning services based on the OWL ontology.

Index Terms—XACML, access control model, RBAC, OWL ontology, authorization reasoning.

I. INTRODUCTION

RBAC [1] is a popular model for access control. In which, roles are created for various job functions in an organization. Members or staffs (users) are assigned for particular roles. The permission to execute a particular task is assigned to particular roles, not to users directly. This simplifies common operations such as adding new users, changing users’ roles. In [2], Ferraiolo et al. have proposed the NIST Standard for RBAC including four model components: Core RBAC, Hierarchical RBAC, Static Separation of Duty Relations, and Dynamic Separation of Duty Relations. In this Standard, the basic elements (Subject, Role, Action, and Resource) are defined in the Core RBAC. For Hierarchical RBAC, roles can be structured hierarchically so that a role can inherit permissions of other roles. Besides, constraints of assigning and activating roles to users are considered in Static and Dynamic Separation of Duty Relations.

Many research have applied RBAC to XACML [3]-[5]. However, these approaches still have some shortcomings. They are not really flexible in expressing different relationships between roles or resources. Moreover, it is hard to maintain role and resource hierarchies when these relationships are changed.

In this paper, we propose a new approach to represent the hierarchy relationships between roles or resources. These hierarchies are implemented by Web Ontology Language (OWL) [6]. An extension model of XACML with new components and functions is introduced to support authorization reasoning ability based on these hierarchies. We also show how to support the NIST Standard for RBAC in our model.

The remainder of the paper is organized as follows. We summarize some related work in Section II. In Section III, we present our approach to model role and resource hierarchies. Section IV describes an extended model of XACML to support reasoning ability. Section V demonstrates that our model can support the highest level of NIST Standard for RBAC. The evaluation of our approach is shown in Section VI. Finally, the last section represents the conclusion and further work on this topic.

II. RELATED WORK

Before elaborating our work, we briefly introduce the most relevant researches. Extensible Access Control Markup Language (XACML) [7], [8] is widely accepted because of its outstanding advantages (e.g. standard, generic, distributed and powerful) [9]. It is a standard to describe both policy language and access control decision request/response language. The policy language is used to describe general access control requirements with standard extensibility points including functions, data types, combining logic, etc. Meanwhile, the request/response language allows users to form a query to ask if a given action should be allowed or not.

With the use of XACML, access control policies can be expressed flexibly due to the extensibility of data types, functions and the ability of combining multiple decisions into a single decision.

Besides, OASIS also introduced an XACML profile [5] to implement RBAC. In this profile, a Permission element contains resources and actions along with further conditions such as valid time, system status. It may also contain the references to Permission elements of other roles that are junior to the given role. By this way, a role can inherit all permissions of its junior roles. However, this makes the hierarchy not flexible in case of partial inheritances.

Fini et al. have defined two approaches to model RBAC using OWL in [4]. This is a base for many researches later. In...
Ferrini and Bertino have supposed a framework to support RBAC with XACML and OWL. However, using only one hierarchy makes it still have some disadvantages as described in Section VI. In [10], we have proposed and implemented a flexible framework to support a generalized extension of XACML for spatio-temporal RBAC model with reasoning ability. And in which, we have mentioned the extension of XACML for spatio-temporal RBAC model with reasoning ability. In this paper, we will describe in more details.

III. NEW ROLE AND RESOURCE HIERARCHY REPRESENTATION

In this section, we introduce a new approach to model role hierarchies to increase the flexibility and the ease of maintenance. In other hands, resource hierarchy is also taken into account to reduce the number and complexity of the policy set.

A. Role Hierarchies

Between two roles in an organization, there can be two kinds of relationships:

- Is-a relationship: This relationship is based on the conceptual definition of the roles. For example, in Fig. 1(a), Doctor is a staff; Surgeon is a doctor who does operations. This relationship is stable.

- Senior/junior relationship: This relationship is based on the organizational structure of the organization. In which, superiors have more rights than their subordinates. For example, in Fig. 1(b), Manager is a senior role of Doctor. This kind of relationship can be changed by policies of organizations.

Normally, in role hierarchies of previous researches, only senior/junior relationships are considered. Because these approaches have some disadvantages analyzed in Section VI, we consider the two different relationships by using two hierarchical trees. The first tree is for is-a relationship; the other is for senior/junior relationship.

Suppose that \( R(A, B) \) represents the relation \( R \) between \( A \) and \( B \). For example, \( \text{senior}(\text{Doctor}, \text{Nurse}) \) expresses that Doctor is a senior role of Nurse. In other to support reasoning ability based on the role hierarchical trees, the relation \( R \) should have following features:

- Transitive: \( R(A, B) \land R(B, C) \Rightarrow R(A, C) \)
- If \( R \) and \( R' \) are inverse properties, then \( R(A, B) \Rightarrow R'(B, A) \)

OWL, a W3C Recommendation, is an ontology language for the web. OWL ontologies provide classes, properties, individuals, data values, and are stored as Semantic Web documents. This language is strong enough for modeling role hierarchies. We can easily define the first hierarchy by the OWL built-in property named \( \text{subClassOf} \). For the second one, we have to use user-defined properties called \( \text{juniorRoleOf} \) and \( \text{seniorRoleOf} \). These two properties satisfy the two above features of the relation.

B. Resource Hierarchy

In some applications, when the number of resource classes becomes larger, then the number of policies will explode quickly. Therefore, it is necessary to have a mechanism to combine resources to a hierarchy of resource classes. In this hierarchy, each resource class represents objects in a same type. If a subject can access higher-level (senior) classes, it can also access lower-level (junior) classes. An example about resource hierarchy is shown in Fig. 2. In this example, Patient Record and Private Note are senior resource classes of Patient Info. It means that if a subject is permit to access Private Note objects, it can also have permission to access Patient Info objects.

For example, role Doctor can be defined as follows:

The definitions of properties \( \text{juniorResourceClassOf} \) and \( \text{seniorResourceClassOf} \) of resource classes are as in the Listing 3.
With above properties, we can use them to define the class `PatientInfo` as follows. In which, we specify that class `PatientInfo` is a junior resource class of `PrivateNote`.

```xml
<gis:ResourceClass rdf:ID="PatientInfo">
  <rdfs:label>PatientInfo</rdfs:label>
  <gis:juniorResourceClassOf>
    <gis:ResourceClass rdf:resource="PatientRecord"/>
  </gis:juniorResourceClassOf>
  <gis:juniorResourceClassOf>
    <gis:ResourceClass rdf:resource="PrivateNote"/>
  </gis:juniorResourceClassOf>
</gis:ResourceClass>
```

Listing 4. Definition of class `patientinfo`.

For each application domain, we could use these two properties or define new properties corresponding to other relationships between resource classes. The XACML extended functions for reasoning on resource hierarchy will be described in Section IV.

### IV. Extended XACML Model with Reasoning Ability

Thanks to the extensibility of XACML, some new components and functions could be added to support reasoning ability based on OWL ontology. Compared to the original XACML Model, this extended model (See Fig. 3) has added the following components:

- **Ontology Authority**: This component manages hierarchies defined in OWL ontology repository.
- **OWL Ontology**: A repository contains OWL files created/edited by Ontology Authority component.
- **Reasoning Services**: This component contains an engine which makes inferences based on OWL ontologies and provides necessary functions used by PDP, PIP, and Obligations Service.

A list of extended functions which needs to be provided is shown in the Table I.

| Table I: XACML Extended Functions for Supporting Reasoning Ability on Role Hierarchies |
| --- | --- |
| Function name | return TRUE if |
| role-is-a(r1, r2) | r1 is a sub class of r2 in the hierarchy described in Fig. 1(a) |
| role-is-senior(r1, r2) | r1 is one of the senior roles of r2 in the hierarchy described in Fig. 1(b) |
| role-is-senior-or-equal(r1, r2) | r1 is r2 or r1 is one of the senior roles of r2 |
| role-is-junior(r1, r2) | r1 is one of the junior roles of r2 |
| role-is-junior-or-equal(r1, r2) | r1 is r2 or r1 is one of the junior roles of r2 |

Besides, we provide another function named `transitive-roles(r, P)`. In which, r is a role, P is a transitive property. This function returns a set of roles associated with r through the property P. Property P could be but not limited to `seniorRoleOf`, `juniorRoleOf`. For example, `transitive-roles(Nurse, seniorRoleOf) = {Doctor, Surgeon}`. By this way, we can define any kind of transitive property and use it for reasoning. This makes the model more flexible.

In similar way, we provide reasoning functions on resources hierarchies as shown in Table II.

| Table II: XACML Extended Functions for Supporting Reasoning Ability on Resource Hierarchies |
| --- | --- |
| Functions | return TRUE if |
| resource-is-senior(c1, c2) | c1 is one of the senior resource classes of c2 in the resource hierarchy |
| resource-is-senior-or-equal(c1, c2) | c1 is c2 or c1 is one of the senior resource classes of c2 |
| resource-is-junior(c1, c2) | c1 is one of the junior resource classes of c2 |
| resource-is-junior-or-equal(c1, c2) | c1 is c2 or c1 is one of the junior resource classes of c2 |
We also define the function \textit{transitive-resource-classes} \((c, P)\). In which, \(c\) is a resource class, \(P\) is a transitive property. This function returns a set of resource classes associated with \(c\) through the property \(P\). Property \(P\) could be but not limited to \textit{seniorResourceClassOf}, \textit{juniorResourceClassOf}. This makes our policies more expressive; and more than one resource hierarchy could be defined.

In summary, the work-flow of our model is as follows. In 1a the administrator of the system create policies by PAP and create ontology by Ontology Authority using OWL API [11]. At run-time, the PEP receives an access request (2), converts it to XACML request and sends to the Context Handler (3). The Context Handler forwards the request to PDP (4). The PDP evaluates the request. If there is any missing attribute, the PDP will ask for the Context Handler (5). The Context Handler then asks the PIP to get the answer (6). Missing attributes could be subject attributes (7a, 7b), environment attributes (7c) or resource attributes (7d). When evaluating, if the policy uses reasoning functions, the PDP contacts the Reasoning Services to get the answers (11). An example about the interaction between the PDP and the Reasoning Services is shown in Fig. 4.

### V. SUPPORTING NIST STANDARD FOR ROLE BASED ACCESS CONTROL

In this section, we demonstrate that our model satisfies the highest level of the NIST Standard for Role Based Access Control.

#### A. Core RBAC

RBAC includes basic elements such as Subject, Role, Action, and Resource. In this model, they are defined by OWL and XACML policies. The Listing shows how to define a Subject and assign it to a role in OWL. We can assign a subject to many roles, and many subjects to a role to support many-to-many role-user assignments.

```xml
<owl:Class rdf:about="Subject"
    rdfs:comment="A Subject."
    rdfs:label="Subject"
</owl:Class>
<rbac:Subject rdf:ID="Ha.Van" rdfs:Label="Ha Van">
    <rbac:name>Ha Van</rbac:name>
    <rbac:hasRole rdf:about="Doctor"/>
</rbac:Subject>
```

Listing 5. User-role assignment.

XACML policies are used to perform role-permission assignments [5].

#### B. Hierarchical RBAC

As described in the Section III, we have used two role hierarchies to support the Hierarchical RBAC requirement of NIST Standard.

#### C. Static Separation of Duty Relations

The built-in property \textit{owl:disjointWith} is used to clarify that a user cannot be assigned to both particular roles. For example, the Listing 2 describes the definition of Doctor, and specifies that a user cannot be assigned to both roles Doctor and Nurse. This kind of Static Separation of Duty is checked after use-role assignments are inserted/updated in step 1b in the work-flow shown in Fig. 3.

#### D. Dynamic Separation of Duty Relations

In the work-flow (Fig. 3), after the PEP received the authorization decision from the Context Handler, the obligations service inserts some information to the ontology repository. This historical data is used for checking Dynamic Separation of Duty (DSOD) in future requests. In step 7a, the PIP contacts the Reasoning Services to get the roles of the subject. The Reasoning Services only returns the roles that satisfy DSOD constraints. The idea to support DSOD in this paper is similar to [4]. The constraints can impact on related roles according to the predefined role hierarchies [12], [13].

### VI. EVALUATION

As mentioned above, our model supports well for the systems which have large number of roles, resource classes. In this section we will evaluate the advantages of this new model in detail.

Considering the case if we use only one role hierarchy and it could be the one in Fig. 5. Compared to this approach, our approach has the following advantages.

**Expressiveness**: Considering the rule “any nurse has permission to read their patients' medical records”, it means roles Eyes Specialist Nurse and Children Specialist Nurse have this permission, but role Doctor doesn’t. The rule is easily expressed by our approach by using the reasoning function named role-is-a. But if we only use one hierarchy as shown in Fig. 5, it is very hard to express. Reasoning
functions transitive-resource-classes and transitive-roles are not limited to any property. Therefore, we can define any transitive property and use these functions to reason the relationships between roles or resources.

**Ease of maintenance:** The is-a relationship hierarchy is usually stable, so policies based on this hierarchy is stable as well. Therefore, we just need to maintain the senior/junior hierarchy only. Otherwise, using only one role hierarchy, we have to change the hierarchy when the organizational structure changes. Moreover, it is very hard to maintain in case the organization has a large number of roles.

Moreover, using resource hierarchy helps reduce the number of policies, and therefore reduce the cost of the policy storage and maintenance.

**Good performance:** When trying to express all relationships between roles in one hierarchy, the height of the hierarchy may be high, and this makes the performance not good. By using two hierarchies, the heights of the role hierarchies are normally lower. This makes the response time of reasoning services faster and therefore makes the performance of the authorization system better.

**Flexibility:** In our approach, role and resource hierarchies are modeled by using built-in subClassOf property and adding OWL transitive properties. Therefore, in some organizations with small number of roles, resource classes, if they don't need to use two role hierarchies or resource hierarchy, they just need to use only necessary properties.

**Standard Conformity:** Our model uses XACML language based on OASIS standard to define declarative access control policies in order to apply in RBAC. This model also conforms to the requirements of NIST standard for RBAC. As a result, the model is easier to be implemented and applied in the industrial environment.

VII. CONCLUSIONS

In this paper, we have proposed a new representation of role and resource hierarchies to increase the flexibility and the ease of maintenance. We also implement our model by extending the XACML with reasoning functions based on OWL ontologies. This model satisfies the highest level of RBAC NIST standard. Comparing to other approaches, our solution can achieve better performance in reasoning process, an ease of policy expression and maintenance. Besides, the number of relationships between the roles, resource classes is not limited to one or two. We can define any transitive properties and use them for reasoning without any modifications on the architecture designs or source code. We use XACML 2.0 and OWL 2 to describe policies and hierarchies. The practicability of this work and performance experimental are our future study.

REFERENCES


Ha Duc Son Van received the B.S. degree in computer science from Ho Chi Minh City University of Technology, Vietnam, in 2009. He is currently a master student at the university and an assistant researcher at Data Security Applied Research Lab. His research interests include data security, applications for smart devices and e-commerce.

Tuan Anh Dang received the B.S. degree in computer science from Ho Chi Minh City University of Technology, Vietnam, in 2011. He is currently a master student at the university and an assistant researcher at Data Security Applied Research Lab. His research interests include data security and applications for smart devices.

Tran Khanh Dang is currently an associate professor of computer science and the vice-dean at the Faculty of Computer Science & Engineering, HCMUT, Vietnam. His research interests include data security & privacy, access controls, authentication & cryptography with noisy data, security issues in outsourced databases services, information security visualization, big data for vertical industries. He has published more than 115 scientific papers in international/national journals & conferences. He is the founder of Data Security Applied Research Lab (http://dstar.edu.vn/). He has also participated in and managed many research as well as commercial projects.