An Extended Role Based Access Control Method for XML Documents

Xiaofeng Meng and Daofeng Luo
Information School, Renmin University of China
Beijing 100872, China
xfmeng@public.bta.net.cn, luodao2001@yahoo.com

Abstract

As XML has been increasingly important as the Data-change format of Internet and Intranet, Access-Control-On-XML-Properties rises as a new issue. Role-Based Access Control (RBAC) is an access control method that has been widely used in Internet, Operation System and Relation Data Base these 10 years. Though RBAC is already relatively mature in the above fields, new problems occur when it is used in XML properties. This paper proposes an integrated model to resolve these problems, after the fully analysis on the features of XML and RBAC.

Keyword: XML, RBAC

1. Introduction

XMl has been increasingly important as the standard exchange format of Internet and intranet, and the issue of Access Control on XML Properties correspondently gets more and more attention. Some researchers, including Ernesto Damiani, Sabrina De Capitani di, etc, have done some fruitful research on this issue, and brought forward some new questions: level of access control, propagation of authorization, etc. Given these, some new concepts such as DTD-based policy, Document-based policy and Fine-level authorization were brought out in [2] and [3].

Meanwhile, Role-based Access Control (RBAC) is an access control technology that came to be mature these ten years. RBAC embodies the “Group” in Unix, “Privilege grouping” in DBMS and “separation of Duty” in early research. The greatest advantage of RBAC is: user submits a request in certain role, and Role System maps naturally to the enterprise personnel structure. As such, when the personnel changes, all need to do is just to re-assign roles to users, rather than update the access control policy files, which proves to be a hard job. A complete RBAC theory model includes three parts: Core RBAC, Hierarchical RBAC and Constrained RBAC. Ferraiolo, Sandhu and Gavrila made a proposed Standard for Role-Based Access Control[1].

We base our work on these two fruitful researches. It focuses on some new features and questions when RBAC is applied to XML properties. We carry out a complete solution to these new questions, and take the “University People Roles System” ([4]) as an illustration to our new model.

The remainder of this paper is organized below: Section 2 describes the special access control problems we concern in XML document; section 3 carries out an extended access control model and its formal description. Section 4 presents how this new model solve the mentioned problems. Section 5 provides an example to illustrate the method; Section 6 is the system framework and algorithm, and Section 7 concludes this paper and views the prospect and the future work.

2. Problems

Traditional access control model is a tri-tuple: <subject, object, action>. For example, in DBMS, “Grant Select on Table Scores to U1” is an instance to this model. Though it has been a relatively mature model in DBMS, it fails to meet the need when applies to XML properties due to XML’s features.

2.1 Role Reuse

The User-Based Access Control model directly assigns authorizations to given users. This method is able to distinguish each user, but it is unable to avoid a mass of unnecessarily repetitious operations, which is just what RBAC tries to avoid. In RBAC, it is roles rather than users that have direct relationships with authorizations. Authorizations are assigned to roles, and roles are assigned to users. In a session, a user submits his requests in certain roles. Thus, users are separated from authorizations by roles. In fact, a role represents the commonness of those users to whom this role is assigned. For example, if a user is a Student, he can query grades. This is the commonness of students, and role Student can represents this commonness. However, the role Student can and only can queries his own grade” means the same role may have different authorizations. That is, role may have specialties when assigned to different user.

So, how can roles represent both users’ commonness and specialties? That’s a problem of reuse of Roles.

2.2 Conflicts due to Multiple Authorization Levels

Authorization level is another problem. In DBMS, there are four authorization levels: Database level, Table level, Field level and Tuple level. Since XML has two features: one is that every valid XML Document has its correspondent DTD, the other is that tags in XML Document are rigorously nested. Given this, we can easily see that there are three levels in Access control on XML: DTD level, Instance level and element level.

Authorization on DTD level means that this authorization is applied to all valid XML Documents that are instances of this DTD. Since every DTD has a collection of instances, DTD level can avoid authorization on single documents. However, there exits distinction even if two documents are both instances of the same DTD. In this case, we can apply authorization on a much finer level: Instance level. Instance level means authorization on a single tag. Since even in the same XML document, some tags are accessible, while others are not, finer levels are needed to solve this problem. That is Element level.

Multiple authorization levels may give rise to conflicts. For example, authorization on DTD level may allow some roles to access some data in a document while authorization on this document does not. It's an important issue how to deal with this kind of conflict.

2.3 Unnecessary repetitious authorizations

In tri-tuple <subject, object, action>, action commonly refers to read, write, create and delete. In most cases, if a role is granted to write a document, he must obtain the read at first. As such, we have to authorize twice: one for read, the other for write. These two authorizations have the same subject and object, and the only difference is the "action" name. In other cases, system may allow some role to "write" a document, but not allow it to "read". How can we avoid unnecessary repetitious authorizations and at the same time distinguish these two cases?

3. An Extended Access Control Method for XML Documents

We introduce two new elements to the traditional access control policy tri-tuple <subject, object, action>, making the policy quintuple-tuple. We refer to it as An Extended Access Control Policy. Please note that in our new model, subject, object and action can all be treated as tree in logic. For example, an XML document (object) can be treated as a DOM Tree [6]; There are inheritance relationships between roles (subject) and thus form a Role Tree; and we will import a new concept of "inheritance of Action" and make actions to be an Action Tree. We will explain this tuple in detail below.

3.1 Formal description

Our extended policy is a quintuple-tuple:< subject, object, action, conditions, properties>. Where:

Subject::=<role+>

Subject is the roles who submit requests. "+" means there is one or more than one role in a access control policy.(the concepts and features of Role can be found in [1]);

Object::=<(URL, Xpath)+>

Object is the target that subject requests to access. URL is the path and filename of the target file(file may be DTD or some specific XML document); Xpath is the path of the target tag in the target document.

Action:::=<(name,inherit)+ >

Name::=<read | write | create | delete>

Inherit::=<true | false>

Action is the operation that subject requests to do. Name is the action name, it may be read, write, create and delete; Inherit means whether this action is inherited or not. It has two alternative values: true or false.

Conditions::=<(Bool | condition)+>

Bool::=<and | or | not>

Condition::=<function, parameters*>

Parameters::=<PCData | XQL | function>

Conditions means under what conditions the policy is valid. There may be multiple conditions, which is joint by logic operator bool, namely "and/or/not". Condition is a function with optional parameters. For example, compareStr ("eq","Tom","Jone") is a function to check whether two strings are equal or not. If they are equal, returns true, else returns false. The parameters can be a string, an XQL clause([8]) or a returned value of a function.

Properties::=<level?, propagation?, priority?, sign>

Level::=<DTD | Instance>

Propagation::=<local | recursive>

Priority::=<hard | soft >

Sign::=<grant | deny>

Properties are attribute values of this policy. They are: level, propagation, priority and sign. The former three are optional, and the last one is required. The default value of propagation is local. If value of level is DTD, the default value of priority is soft, else hard.

3.2 DTD of policy

The DTD of the above extended policy is shown in Figure 1.

```
<!ELEMENT policy_list  (policy+ )>
<!ELEMENT policy  (subject , object ,
  acitons , conditons? , properties )>

<!ELEMENT subject  (role+ )>
<!ELEMENT role  (#PCDATA )>

<!ELEMENT object  (URI , Xpath+ )+>
<!ELEMENT URI  (#PCDATA )>
<!ELEMENT Xpath  (#PCDATA )>

<!ELEMENT acitons  (action+ )>
<!ELEMENT action EMPTY>
<!ATTLIST action  name    CDATA  #REQUIRED
                  inherit CDATA  #REQUIRED >
<!ELEMENT XQL  (#PCDATA )>

<!ELEMENT conditons  (bool | condition )+>
<!ELEMENT condition  (function* )>
<!ELEMENT bool EMPTY>
<!ATTLIST bool  name CDATA  #REQUIRED >
<!ELEMENT parameters  (#PCDATA | XQL |
function )*>
<!ELEMENT function  (parameters* )>
<!ATTLIST function  name CDATA  #REQUIRED >

<!ELEMENT properties
(level,propagation?,priority?,sign )>
<!ELEMENT level EMPTY>
<!ATTLIST level  value CDATA  #REQUIRED >
<!ELEMENT propagation EMPTY>
<!ATTLIST propagation  value CDATA  #REQUIRED >
<!ELEMENT priority EMPTY>
<!ATTLIST priority  value CDATA  #REQUIRED >
<!ELEMENT sign EMPTY>
<!ATTLIST sign  value CDATA  #REQUIRED >
```

Figure1: Policy.dtd

4. Multiple-level Access Control Based on Reused Role and Inherited Action

Given the above-extended policy, we carry out a new access control method, which is called Multiple-level Access Control based on Reused-Role and Inheritable-Action. Let’s see how this new method resolves the three problems mentioned above.

4.1 Resolving the problem of Role Reuse.

We import the concept of “conditions” to solve the problem of “Role reuse”. Only those who satisfy the conditions can be fit to the policy.

For example, “Student can and only can query his own grades”, the condition is: uid=Student_id, where uid is the login user’s id and student_id is a tag of the target document scores.xml. The request submitted in role of “Student” only returns grades that Student_id equals to the user id.

Another example is “ Dean can only check the grades of his own department”. Suppose there are two documents: scores.xml records the grades of students, class_id,course_id and teacher_id are it’s attributes; department.xml records information about deparments such as dean_id and class_id. Now a dean role submits a request to query the grades of students. The conditions is: firstly, we check department.xml to find out which department this dean belongs to (uid=dean_id); secondly, check whether class_id of target document (scores.XML) belongs to the class_id set of this department. If true, this user (dean) are authorized to query the scores, else the request is refused.

Please note that the importing conditions not only can solve the problem of “Role reuse”, but also can support much richer semantic expression. For example, “Professor can only modify the grades of students between 8:00 AM Jan 1st and 4:00PM Jan 23th”; “Dean can query the personnel information of his own department only after the secretary log”, etc..

4.2 Resolving the problem of conflict due to multiple authorization levels

To deal with the conflict of multiple levels, we follow the principle of “ finer level takes precedence”. This principle means when conflict occurs, authorization on Element level takes precedence over that on instance level, and instance level takes precedence over DTD level.

This principle is quite reasonable: finer level means more explicit and pertinent, so finer level has higher priority. Though this principle is proper in most cases, some opposite cases exist: Some authorizations on DTD level are not legal to be covered by authorizations on finer level. This opposite case is quite common in Distributed Database. To distinguish this two cases, we import two attributes: level and priority. Level’s value may be DTD or Instance, and Instance takes precedence over DTD by default; priority’s value may be hard or soft. Hard means this authorization policy is not legal to be covered, while soft means the opposite.

A question about Element level is: since tags (elements) are nesting, whether authorizations on an element can be applied to its sub-element?

We introduce an attribute “propagation”, it has two alternative values: local or recursive. “local” means the authorization on this element can not be applied to its sub-elements, while “recursive” means the opposite.

Thus we provide a powerful means to deal with conflicts.

4.3 Resolve the problem of unnecessary repetitious authorizations

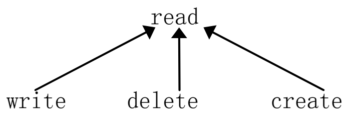

Figure 2

Hierarchical Actions

We introduce a new concept of “ inheritance of actions” (see Figure 2).

If operation op1 inherits op2 and some role r is granted to op1 on some subject, then r is automatically granted to op2 on this subject. For example, if r is authorized to write some object, it can read the same object without another authorization, because write inherits from read. We can thus avoid unnecessary repetitious authorizations.

However, in some other cases, we do not want this kind of automatic mechanism. So we import an attribute inherit, it has two alternative values: true or false. The former means if op2 inheites op1 and r is authorized to op2 then r is authorized to op1 automatically, and the latter means r is not automatically authorized to op1.

Please note that the importing of inheritance of Actions can enable administrators to define more complex and rich-semantic operations.

5. Illustration

We take the University People Roles System (see Figure 3 from[4]) as our example to illustrate our model. Note that all

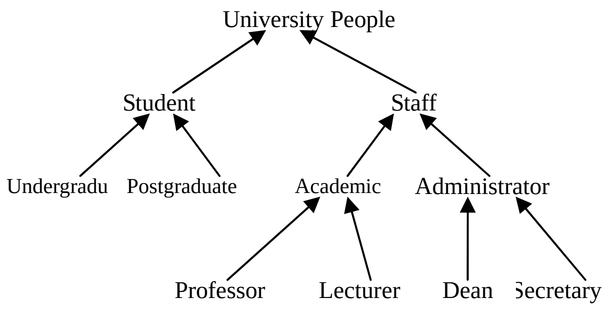

Figure 3: University People Roles

of our files are in XML.

Figure 4 consists of four files: roles.xml, scores.xml and departmetn.xml. (In roles.xml, SSD is Static Separation of Duty([1]).)

```
……
<Role_list>
   <role id="University People" cardinality="unlimited">
     <role id="Student" cardinality="unlimited">
       <role id="Undergraduate" cardinality="10000">
           <SSD>
               <role_id>Staff</role_id>
           </SSD>
       </role>
       <role id="Postgraduate" cardinality="5000">
           <SSD>
               <role_id>Professor</role_id>
               <role_id>Administrator</role_id>
           </SSD>
       </role>
     </role>
     <role id="Staff" cardinality="3000">
     </role>
  </role>
 </Role_list>
                         Roles.xml
```

```
……
<scores_list class-id="infor97"
   course_id="Database" teacher_id="Liuyi">
  <score>
      <student_id>S971310</student_id>
      <grade>B</grade>
  </score>
  <score>
      <student_id>S971311</student_id>
      <grade>A</grade>
  </score>
</scores_list>
                        Scores.xml
```

```
<department_list>
  <department d_id="information">
      <class_list>
          <class c_id="infor97"/>
          <class c_id="infor98"/>
          <class c_id="math98"/>
      </class_list>
      <dean>T1001</dean>
  </department>
</department_list>
         department.xml
```

Figure 4: XML Documents

Figure 5 shows the policy examples.

In Example 1, condition means: when user's identifier is equal to the value of score_list/score/student_id, the value of score_list/score/grade is allowed to return. CompareStr is a function to compare two strings: the first parameter eq means equal; the second parameter is a function getUid, which will return the identifier of current user; the third parameter is also a function getValue, which will get the value of the given path. If the value of the second and the third parameter is equal, condition's value is true, level value=DTD means this policy is on DTD level, so it can be applied to all valid instance of scores.dtd; propagation value=local means this policy can only be applied to the object element, excluding its subelement; priority value=hard means this policy is not allowed to be covered by other policy on finer level; sign value=grant means if the value of condition is true, the action in this policy is granted.

In Example 2, the condition is: find out of which department this dean is(according to user's login identifier and department.xml), then get all the class_id of this department (in

```
  <policy>
      <subject>
          <role>Student</role>
      </subject>
      <object>
          <URI>/scores.dtd </URI>
          <Xpath>score_list/score/grade</Xpath>
      </object>
      <acitons>
          <action name="read" inherit="true"/>
      </acitons>
      <conditons>
          <condition>
            <function name="CompareStr">
                <parameters>eq</parameters>
                <parameters>
                    <function
                         name="getUid"></function>
                </parameters>
                <parameters>
                    <function name="getValue">
                      <parameters>
                         score_list/score/student_id
                      </parameters>
                    </function>
                </parameters>
            </function>
          </condition>
      </conditons>
      <properties>
          <level value="DTD"/>
          <propagation value="local"/>
          <priority value="hard"/>
          <sign value="grant"/>
      </properties>
   </policy>
                            Example1
```

```
……
<condition>
  <function name="belongTo">
     <parameters>
       <function name="getValue">
         <parameters>
           score_list[@class_id]</parameters>
       </function>
      </parameters>
      <parameters>
        <XQL>
          construct <result>{
            where
              <department_list>
                 <department>
                   <class c_id=$c_id/>
                    <dean>$dean_id</dean>
                 </department>
               </department_list>
            in document.xml
            $dean_id=getUid()
            construct <class c_id=$c_id/>
          }
          </result>
        </XQL>
       </parameters>
   </function>
</condition>
……
                              Example2
```

Figure 5: Policy Examples

department.xml), and at last check whether the class_id of the target document is one of the class of this department. The function is belongTo. The first parameter is the value of class_id (attribute of the tag score_list), and the second parameter is an XQL([8]) clause, which queries the class_id in department.xml when user's login identifier equals to the dean_id and returns the result in form of:

```
<results>
    <class_id>infor97</class_id>
    <class_id>math98</class_id>
</results>
```

. The third parameter is the path of the result set. When the value of the first parameter belongs to the set (the second parameter), the condition is true, else is false.

6. System Architecture and Algorithm

6.1 System architecture figure

Figure 6 shows the architecture of our access control system.

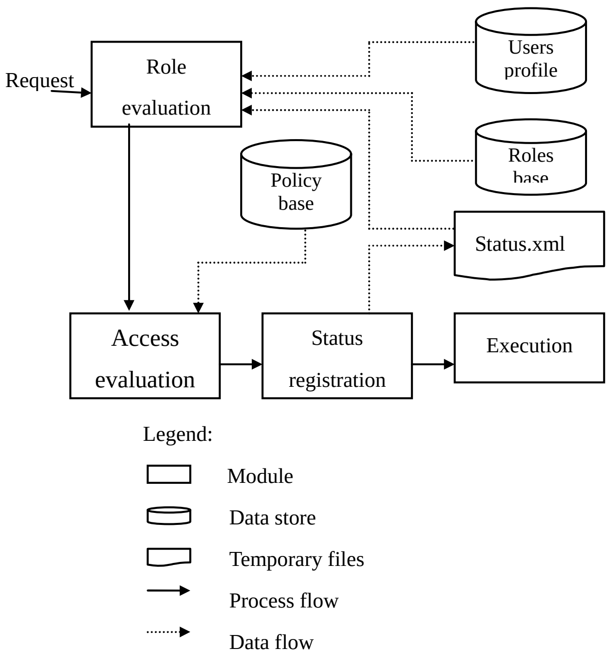

Figure 6: System architecture

Given a request submitted by a user in some role, the performance is by the following four steps: 1)Role evaluation evaluates whether this user has the right to submit request in this role. Check whether he is assigned this role or is he submitting another request in a role which has a DTD relation with this role. 2)Access evaluation evaluates whether this role is authorized to execute the action on the target object; 3)if the evaluation result of step 2 is positive, Status registration register the status of this user and the target document; 4)Execution executes the requested action. In figure 6, Users profile stores the user's information such as user's identifier, password, all his assigned roles,et; Roles base stores the information about roles, such as the hierarchical Relationships, the SSD relationships and DSD relationships([1]); Policy base stores the policy file policy.xml; Status.xml stores the status of users and files.

In this system, we follow two base rules:

Separate Roles of Administration from Roles of users

Roles of Administration are those who are in charge of role's definition, assignment, authorization and so on. They are approximately equivalent to DBA in DBMS. To separate roles of administration from roles of common users also means to separate data about access control(such as roles.xml, policy.xml) from common data(such as scores.xml, department.xml). This is a common rule of access control.

Centrally management of roles and policies

Role System maps naturally to the enterprise personnel structure and there are hierarchical Relationships, SSD and DSD relationships([1]) among roles. So it is nature to manage roles and policies centrally to make the system less error-prone. Especially when RBAC is applied on XML properties. Because XML is mainly applied in net environment, centrally management of roles and policies can help to keep consistency of data, and at the same time reduce the burden of keeping the data of each local website synchronal.

6.2 Algorithm

request::=<uid,role,action,object>.

1: user u submits a request R(uid,r,act,target).

2: evaluate whether u has the right to submit request with role of r(Role evaluation):

2.1):read Users.xml to check whether r or sub-roles of r(roles inheriting from r) is assigned to u, if false, refuese the request, else go to 2.2;

2.2):read roles.xml and status.xml to check whether u is using other roles which have DTD relationships with r, if true, refuse the request, else go to 3;

3: read policy.xml to find out the matching policy to this request(Access evaluation).

3.1):Let dtd=getDTD(target)   childAction=getChildAction(act)

3.2): get those policies where role=r,URI=target or URI=dtd, action[@name]=act or action[@name]=childAction(the left of the equal mark are values of tags in policy.xml and the right are values in request R or the returned value of 3.1). Let the results set is P{ p1,p2…pn} (pi is a piece of policy). If P is null, refuse the request and report this unmatching request to Administrators;

3.3):Check the condition of pI, If true, go to 3.4, else check the condition of pi+1 till pn;

3.4):Label the target tags with the sign value of properties of pI, if the value of propagation is recursive, lable all the sub-tags of current tag with the same sing value; if the target tags or its sub-tags have been labled by other policy and conflict arises, follow the rules below:

a) If priority value of one policy is hard and the other is soft, choose the sign value of which priority is hard.

b) If fails, choose the sign value of which level value is instance.

c) If both fail, choose deny and report this un resolved conflict.

3.5)Enforce 3.3 till pn,

4: Register the status of u and target document(Status registration).

5:Execute the requested action.

7. Conclusion and future work

With the increasingly widely use of XML, the security issue of XML is getting more and more attention. Access control on XML properties is a relatively new issue, while RBAC is a mature access control method. This paper discusses the new problems when RBAC is applied on XML properties and carries out an extended access control method——the Role-Based and Inheritable-Action Multiple-level Access Control, and system architecture and algorithm.

Our new method has great flexibility to satisfy more complex access control requirement. Our future work is: research when our model is applied to E-commerce what problems will arise and how to resolve these problems.

8. References

1. David F. Ferraiolo ,Ravi Sandhu, Serban Gavrila , A Proposed Standard for Role-Based Access Control, National Institute of Standards and Technology, December 18, 2000

2. Ernesto Damiani, Sabrina De Capitani di Vimercati , Stefano Paraboschi, Pierangela Samarati, Design and Implementation of an Access Control Processor for XML Documents

3. Elena Ferrari , Access Control Policies for XML Document Sources, University of Milan, Italy.

4. Hao He, Raymond K. Wong, Australia, A role-Based Access Control Model For XML Repositories", Proc. of WISE'2000, June, 2000, Hong Kong.

5. AlphaWorks XML Security Suite, http://www.alphaWorks.ibm.com/tech/XMLsecuritysuite.

6. World Wide Web Consortium (W3C). Document Object Model (DOM) Level 2 Specification Version 1.0., September Working Draft 1999. http://www.w3.org/TR/WD-DOM-Level-2.

7. World Wide Web Consortium (W3C). XML Path Language (XPath), November 1999. http://www.w3.org/TR/xpath.

8. XQL,http://www.texcel.no/whitepapers/xql-design.html