

Faculty of Informatics

Faculty of Informatics - Papers

University of Wollongong

Year 2007

Temporal Authorizations Scheme for
XML Documents

J. Wu*

J. Seberry[†]

Y. Mu[‡]

*University of Wollongong, jw91@uow.edu.au

[†]University of Wollongong, jennie@uow.edu.au

[‡]University of Wollongong, ymu@uow.edu.au

This conference paper was originally published as Wu, J, Seberry, J and Mu, Y, Temporal authorizations scheme for XML documents, in Proceedings of the 5th WSEAS Conference on Data Networks, Communications and Computers, 2007.

This paper is posted at Research Online.

<http://ro.uow.edu.au/infopapers/579>

Temporal XML Delegable Authorizations

Jing Wu, Jennifer Seberry, and Yi Mu
School of Information Technology and Computer Science,
University of Wollongong, NSW 2522, Australia
Email: [jw91,jennie,ymu]@uow.edu.au

Abstract

In a large system, XML documents associated with it can be large and complicated. To manage access control in such a large and complicated system is very difficult. Recently, Access Policy Sheet (APS) [6] was introduced to provide a solution to access control for large XML systems. In this paper, we proposed a temporal access control scheme in APS where the propagation of authorization rights is assumed. The authorization policies can be automatically revoked when the associated time expires. We also provide conflict resolutions for our temporal authorization system.

Keywords: XML, Access control, Temporal authorization

1 Introduction

XML has been widely applied to manage Internet information. An XML document system could be very large and complicated and its security protection could be also very difficult. Protecting such an XML system requires a sound access control mechanism, which provides formalisms for specifying, analyzing and evaluating security policies that determine how an access right is granted and delegated among particular users.

Recently, researchers [2, 3, 4] have become increasingly interested in developing authorization models which are flexible and expressive enough so as to handle the specification and enforcement of multiple policies. [1, 5, 9] have provided some ideas developing XML authorization models which are flexible and expressive enough for handling the specification and enforcement of multiple policies. However, those models are very complicated and difficult to realize. In our previous work, Access Policy Sheet (APS) [6], has been introduced. This model provides a simple and dynamic scheme for XML authorization management. However, temporal authorization [8] and authorization propagation have still not been investigated in XML based access control systems including APS. In this paper, we present some new properties in APS including temporal authorization, conflict resolutions and the time dependency which can handle temporal access control policies and policy propagation. In our model,

the authorization can be automatically revoked when the associated time expires. By the hierarchical relationships along with the partial orders defined in APS, we also provide a mechanism for conflict resolutions.

The rest of this paper is organized as follows. In Sections 2,3 and 4, we describe the basic definitions of our model including those of APS. In Section 5, we present the XML Temporal Delegation Authorization. In Section 6, we discuss the conflict resolution. In Section 7, we investigate the conflict resolution while the authorization propagation is assumed. In Section 8, we conclude this paper.

2 Definitions of the vocabularies in DTD

In this section, we define our access control model. We give the definitions of DTD, APS, and associated system components including subjects, objects, authorization rights, and types.

2.0.1 Subject

A subject is active. It could be a user or a processor. A subject has a name and other associated information dependent on the application. We require subjects to be either ordered with a proper order or unordered when the order of subjects are insignificant.

Subject Set. Subject constant poset $(S, >)$: $admin, s_1, s_2, \dots, s_n$ denote ordered subjects with the order of $admin > s_1 > s_2 > \dots > s_n$. We assume that the administrator possesses the highest privilege.

A subject can be defined according to the need. For example, a subject could be described by set of attributes such as name, address, rights, etc. As the simplest example, in DTD, the subject is defined as:

```
<!DOCTYPE subject [
<!ELEMENT subject (users*)>
<!ELEMENT users (name)>
<!ELEMENT name #PCDATA>
]>
```

The attribute to the above subject set contains only the usernames.

2.0.2 Object

Objects are passive. They could be files, programs, tables, etc. Objects are represented by a constant poset $(O, >)$: o_1, o_2, \dots with the order $o_1 > o_2, > \dots$. The object is described as **target** + **path**(V, E), where **target** is an XML document or URL address, **path** is an XPath expression that eventually selects specific portions (object) of the XML document in the XML tree where V is a set of nodes and E is a set of edges. The structure of the objects could be defined in the DTD as follows.

```
<!DOCTYPE object[
<!ELEMENT object (target,path)>
<!ELEMENT target href #PCDATA>
<!ELEMENT path #PCDATA>
]>
```

In our XML access control model, the object are the nodes of the DOM tree. We allow the objects are ordered. The order of object is given by tree inheritance relation.

2.0.3 Access Rights

The order of access right is given by a priority hierarchy relation. In our model, we define the object with the access right of **write** will automatically have the access right of **read**, so we have the partial order: **read** > **write**. That is, if a user has a **write** right on an object, then he will have a **read** right on it as well.

Ordered rights are defined as a constant poset $(A, >)$: a_1, a_2, \dots with the order: $a_1 > a_2 > \dots$. For example, **Read** > **Write** > **Executable**. They are defined in DTD as follows.

```
<!DOCTYPE access_right[
<!ELEMENT access_right (a*) #IMPLIED>
]>
```

2.0.4 Authorization Type

The ordered authorization types are described in DTD:

```
<!DOCTYPE authorization_type[
<!ELEMENT authorization_type
(+|-|*) #REQUIRED>
]>
```

Authorization type is given by the constant set $T = \{+, -, *\}$, where $+$ specifies grant w.r.t. an access right, $-$ is the negation w.r.t. an access right, and $*$ specifies “delegable” w.r.t. an access right.

3 Predicates

In this section, we give the definitions of the predicates of **grant**, **delegate** and **cangrant** for describing the delegatable authorizations.

Definition 1 (*grant*) *grant* is a 5 -tuple predicate $|S| \times |O| \times |T| \times |A| \times |S|$.

grant(s, o, t, a, g) : *grantee* s is granted by grantor g the access right a on object o with authorization type t . We can use XML to define the structure of rule *Grant* In the delegation process, the entity that has been given the access right to delegate by another entity, who has the access right, can perform valid delegations. The following is the definition of our delegable rules. Predicate *grant* in XML is an authorization rule, where the element *grant* with attributes *grantee*; *object*, *authorization_type*; *access_rights*; *grantor* and *status*. Here,

```
grantor ∈ S,
grantee ∈ S,
target+path ∈ O,
authorization_type ∈ {+|-|*},
where +, -, * denote allow, deny, and delegable, respectively.
access_right ∈ {r, w, ...},
status ∈ {true, false}
```

Definition 2 (*cangrant*) *cangrant* (s, o, a): *subject* s has the right to grant access a on object o to other subjects.

cangrant is a 3-tuple predicate $|S| \times |O| \times |A|$, where S is a set of subjects which is a grantor or a grantee; $O = \text{target} + \text{path}(V, E)$, **target** is an XML or URL, **path** is an XPath expression that eventually selects specific portions (object) of the XML document in XML tree where V is a set of nodes and E is a set of edges; A is the set of access rights. Here,

```
subject ∈ S,
target+path ∈ O,
access_right ∈ {read, write, a1, a2, ...}
status ∈ {true, false}
```

Definition 3 (*delegate*) *Delegate* is a 4-tuple predicate $:|S| \times |S| \times |O| \times |A|$ rule, where S is a set of subjects which is a grantor or a grantee; $O = \text{target} + \text{path}(V, E)$, **target** is an XML or DTD, **path** is an XPath expression that eventually selects specific portions (object) of the XML document in XML tree where V is a set of nodes and E is a set of edges; A is the set of access rights. Here,

```
grantor ∈ S,
grantee ∈ S,
target+path ∈ O,
access_right ∈ {read, write, a1, a2, ...},
```

`status` \in {`true`, `false`}

`delegate(g,s,o,a)`: subject g has directly or indirectly granted subject s access a on object o with delegable type.

4 Authorization Rules in APS

Directly using XML to describe access control often shows little advantage when the access control system is complicated (e.g., when authorization delegation and propagation are required). In our model, authorization specifications or rules are provided in an Authorization Policy Sheet (APS) associated with the document/DTD. In APS, the representation of authorizations is described in terms of orders of the objects and subjects and explicit authorization rules.

The APS is separate from the document and DTD and offers great convenience in the administration of access control for system administrators due to its simplicity. The system administrator can manage the system access control by the concise rules given in an APS. The resultant XML sheet can be generated from the corresponding DTD and APS. APS also shows the great advantage due to its convenience in the specification of explicit rights and the implicit rights for XML documents.

The partial orders of the access control components, including subjects, object, types, and rights, are one of the key components in an APS. We will see that they can be used to simplify our access control system by implicit rules in authorization propagations.

The following figure of the XML DOM tree expresses the hierarchy relation of the objects.

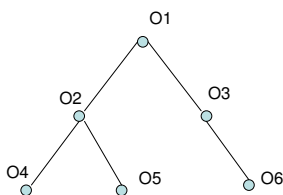


Figure 1: DOM tree, where each node represents an object.

Now let's see how to determine the orders of the objects. According to the structure of the DOM tree. In our model, the root node has the greatest order in any order chain. We can describe the hierarchy relation as following: o_1 is the root of the tree, o_2 and o_3 are the descendant nodes of o_1 , o_4 and o_5 are the children nodes of o_2 and the grandchildren nodes of o_1 , o_6 is the child node of o_3 and the grandchild nodes of o_1 ,

So, we have $o_1 > o_2 > o_4$, $o_1 > o_3 > o_6$, and $o_1 > o_2 > o_5$.

To provide fine grained specification in XML, we utilize the XPath expression to identify the structure of the tree. An Xpath expressions on an XML document tree is a sequence of element name or predefined functions separated by the character: for the one branch of the tree: $o_1-o_2-o_4$, node o_4 can be expressed as `/o1/o2/o4/` and we have the partial order: `/o1/ > /o1/o2/ > /o1/o2/o4/`.

In general, we have the object partial order chain: `/root/ > /root/ child of root/ > ... > /root/child of root/grandchild of root/.../`.

For example:

```

//hospital/ > //hospital/room-info/ >
//hospital/room-info/patient/
  
```

Thus, the order of the ancestor nodes in a DOM tree are always greater than the order of the descendant nodes.

An APS sheet consists of a finite set of rules and orders. A rule consists of *name*, *head* and *attribute*. When the head of a rule is an authorization predicate, the rule is called authorization rule. For a set of rules named r , each rule consists of a predicate and an attribute:

`<p, attribute> <- <condition>`

Here, p_1, p_2, \dots, p_n are a set of predicates and *attribute* denotes the components associated with the predicate. *condition* denotes the condition with respect to the rule of a predicate. Due to the space limit, we will omit the details in this paper and will present it in the full version of the paper.

The structure of rule in DTD is defined as following:

```

<!DOCTYPE rule[
<!ELEMENT rule (predicate+,condition*)>
<!ELEMENT predicate (grant, cangrant)
#IMPLIED>
<!ELEMENT condition (gran, cangrant)*>
]>
  
```

5 Syntax of Temporal Authorizations

Access Policy Sheet (APS) allows an administrator to manage the rules separated from XML documents and DTD. APS consists of a finite set of rules. $R = \{r_1, r_2, \dots, r_n\}$, A rule consists of a set of predicates and conditions associated with the predicates. $r = (P, C)$, where $P = \{p_1, p_2, \dots, p_n\}$ denotes a set of predicates and C denotes a set of conditions. We describe a rule in APS as,

`<p, attributes> <- <coditions>`
 where p is a predicate of arity n in P .

Temporal authorizations enforce a rule to have tenable validity. Formally, we add a time interval Δ_i to a rule, e.g., $r_i = (P, C)(\Delta_i)$, where r_i is the temporal rule which has a temporal argument Δ_i defined as $\Delta_i = (s_i, e_i)$. s_i is the start time and e_i is the end time.

In APS, we define a temporal rule as

```
<p, attributes>
  <- <conditions><time_interval>
```

For example,

```
<garnt, Alice, patient_info.xml,
  //Star_Hospital/operation_info,
  +, read&write, Bob> <-
  <cangrant, Bob,
    *+//Star_Hospital/operation_info,
    read>
  <09/10/05,11/10/05>
```

This rule is applied to predicate **grant**, condition **cangrant** and the validity period of the authorization which contains the start time and the end time. Grantee Alice has the access right **read** and **write** on the Xpath specified object **hospital_info.xml**, with the Xpath **//Star_Hospital/operation_info**. Grantor Bob has the authorization to grant **read** on the all files of the same directory to others. The validity period of the rule is 09/10/05-11/10/05.

According to the structure defined in the associated DTD (omitted), the authorization will be able to be converted to the standard XML as follows.

```
<rule>
  <grant>
    <subject>
      <grantee>
        <name> Alice</name>
      </grantee>
    </subject>
    <object >
      <target>hospital_info.xml</target>
      <path>//nurse/operation_info</path>
    </object>
    <authorization_type>
      + </authorization_type>
    <access_right> read </access_right>
    <access_right> write</access_right>
    <subject>
      <grantor>
        <name> Bob</name>
      </grantor>
    </subject>
  </grant>
  <condition>
    <cangrant>
      <subject>
        <grantee>
          <name> Alice </name>
        </grantee>
      </subject>
    </object >
```

```
<target>hospital_info.xml
  </target>
  <path>//nurse/operation_info
    </path>
</object>
  <access_right> read </access_right>
</cangrant>
<validity>
  <start> 09-10-05 </start>
  <end> 11-10-05 </end>
</validity>
</rule>
```

6 Conflict Resolution

Since both positive and negative authorizations can co-exist in our system, conflict resolution among rules must be considered. Also, since we allow implicit authorization (authorization via authorization propagation), implicit conflicts make the problem more complicated. The policy of “the most specific subject(or object) takes precedence” which is proposed by E. Bertino [1] cannot be adapted to our model. In fact, according to the inherence hierarchy relation in the partial order chain, we can get the policy of the “the least specific takes precedence”. The basic idea of solving conflicts is outlined as follows.

6.1 General Rules

Using delegation relation. According to the delegation relation, if subject **s** delegates subject **s'** directly or indirectly an authorization on object **o** and access right **a**, then, when a conflict w.r.t **o** and **a** occurs, the authorization from **s** (i.e. **s** is the grantor) will always override the one from **s'**.

From authorization type inheritance. For two conflicting authorizations w.r.t. a subject, an object, and an access right, according to the inheritance hierarchy of authorization type $- > + > *$, the authorization with a lower partial order will override the one with a higher type order.

From subject inheritance. For two conflicting authorizations w.r.t. an object, a right, and an authorization type, according to the inheritance hierarchy of subjects, if $s > s'$, then the authorization w.r.t. **s** will override the inherited one with **s'**.

From object inheritance. For two conflicting authorizations w.r.t. a right, a grantor, and a grantee, we consider the object inheritance relation. According to the inheritance hierarchy of objects, if $o > o'$, then the authorization on **o** will override the inherited one on **o'**.

From access right inheritance. For two conflicting authorizations w.r.t. a grantor, a grantee, and an object, according to the inheritance hierarchy of access rights, if $a > a'$, then the authorization on **a** will

override the inherited one on \mathbf{a}' .

6.2 The Scheme for Temporal Authorizations

According to the definition of temporal authorization in the preceding section, we denote by $r_i(\Delta_i)$ the temporal authorization, where $\Delta_i = (s_i, e_i)$ is the validity period of rule r_i , s_i is the start time, and e_i is the end time. Our conflict resolution is twofold: resolution without the time factor (given in the preceding section) and resolution with the consideration of the time factor. For example, for two conflict authorizations $r_i(\Delta_i)$ and $r_j(\Delta_j)$, we consider the conflict resolution of the r_i and r_j omitting the time factor first, using the methods we discussed in the preceding section. Then, we consider the time factors Δ_i and Δ_j . We denote by ϕ the time overlap of two temporal authorizations, i.e., $\phi = \Delta_i \cap \Delta_j$, by \Rightarrow an override operator, and by \Leftrightarrow a conflict operator.

Suppose we have two rules $r_i(\Delta_i) \Leftrightarrow r_j(\Delta_j)$, where $\Delta_i = (s_i, e_i)$, $\Delta_j = (s_j, e_j)$. If $\phi \neq 0$, we have the following cases.

When $s_i = s_j$ and $e_i = e_j$, then we have

$$\begin{cases} r_i \Rightarrow r_j : r_i(s_i, e_i) \\ r_j \Rightarrow r_i : r_j(s_j, e_j) \end{cases}$$

which means two conflict authorizations completely overlap. If $r_i \Rightarrow r_j$, then we have $r_i(s_i, e_i)$. If $r_j \Rightarrow r_i$, then we have $r_j(s_j, e_j)$. The following cases are self-explanatory; therefore, we omit the explanation.

In the case of $s_i \neq s_j, e_i = e_j$, we have

$$\begin{cases} r_i \Rightarrow r_j : r_i(s_i, e_i), r_j(s_j, s_i) \\ r_j \Rightarrow r_i : r_j(s_j, e_j) \end{cases}$$

In the case of $s_i = s_j, e_i \neq e_j$, we have

$$\begin{cases} r_i \Rightarrow r_j : r_i(s_i, e_i) \\ r_j \Rightarrow r_i : r_j(s_j, e_j), r_i(e_j, e_i) \end{cases}$$

In the case of $s_i \neq s_j, e_i \neq e_j$, we have

$$\begin{cases} r_i \Rightarrow r_j : r_i(s_i, e_i) \\ r_j \Rightarrow r_i : r_j(s_j, e_j), r_i(s_i, s_j), r_i(e_j, e_i) \end{cases}$$

When $s_i > s_j, e_i > e_j$, we have

$$\begin{cases} r_i \Rightarrow r_j : r_i(s_i, e_i), r_j(e_i, e_j) \\ r_j \Rightarrow r_i : r_i(s_i, s_j), r_i(s_j, e_j) \end{cases}$$

If there is no time overlap between two conflict authorizations, then each rule is taken separately without any influence from the other.

$$\begin{cases} r_i \Leftrightarrow r_j \\ \Delta_i \cap \Delta_j = 0 \end{cases} : r_i(\Delta_i), r_j(\Delta_j)$$

7 Conflict Resolution with Authorization Propagation

Authorization propagation is implemented in terms of the partial order of subjects, objects and access rights. As defined in [4], we allow the implicit rules derived from the explicit rules in terms of partial ordering.

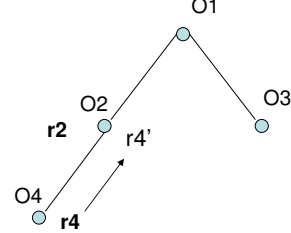


Figure 2: Rule propagation in a DOM tree.

Figure 2 shows an XML DOM tree, where object $\mathbf{o1}$ is the root of the tree, $\mathbf{o4}$ and $\mathbf{o2}$ are the nodes on one branch of the tree. $\mathbf{o4}$ is the descent node of $\mathbf{o2}$. According to the partial order rule of the propagation on the objects we introduced before, there exists a partial order relation $\mathbf{o2} > \mathbf{o4}$.

Suppose that in APS there are two authorization rules r_4 and r_2 actively applied to $\mathbf{o4}$ and $\mathbf{o2}$, respectively. With authorization propagation, r_4' is derived from r_4 . The authorization rule w.r.t. $\mathbf{o2}$ is then affected by r_4' . If r_4 is in conflict with r_2 , then r_4' is also in conflict with r_2 . The conflict resolution discussed in Section 4.1 is still applicable.

We now discuss it further by considering the time factor. Suppose that the above authorizations are temporal rules, $r_4(\Delta_4)$ and $r_2(\Delta_2)$. For $\mathbf{o2}$, $r_4'(\Delta_4)$ derived from $r_4(\Delta_4)$ is also applied. Consequently, we have

$$\begin{cases} r_4(\Delta_4) \Leftrightarrow r_2(\Delta_2) \\ \Delta_4 \cap \Delta_2 = 0 \end{cases} : r_4'(\Delta_4) \not\Rightarrow r_2(\Delta_2)$$

and

$$\begin{cases} r_4(\Delta_4) \Leftrightarrow r_2(\Delta_2) \\ \Delta_4 \cap \Delta_2 \neq 0 \end{cases} :$$

$$\begin{cases} r_4'(\Delta_4) \not\Rightarrow r_2(\Delta_2); \text{ (no overlap)} \\ r_4'(\Delta_4) \Leftrightarrow r_2(\Delta_2); \text{ (overlap)} \end{cases}$$

Here, we give an example. Suppose that in XML document `medical.xml`, there are two nodes in the DOM tree, `//hospital/room-info/` and its child node `//hospital/room-info/patient/`. According to the hierarchy relation of the objects, we have a partial order relation:

`//hospital/room-info/ > //hospital/room-info/patient/.`

Assume that $r_1(\Delta_1)$ and $r_2(\Delta_2)$ are two temporal rules w.r.t. the above objects respectively.

$r_1(\Delta_1)$:
`<grant, Alice,medical.xml,
 //hospital/room-info/,+,read,Bob>
 (01/04/05,01/07/05)`

$r_2(\Delta_2)$:
`<grant, Alice,medical.xml,
 //hospital/room-info/patient/,-,read,Bob>
 (01/06/05,01/12/05)`

Through the propagation, $r'_2(\Delta_2)$ is derived from $r_2(\Delta_2)$. That is, $r'_2(\Delta_2)$:

`<grant, Alice, medical.xml+
 //hospital/room-info/,-,read,Bob>
 (01/06/05,01/12/05)`

Obviously, for the same object

`medical.xml+//hospital/room-info/,`

$r'_2(\Delta_2)$ is in conflict with $r_1(\Delta_1)$. $\Delta_1 = (s_1, e_1) = (01/04/05, 01/07/05)$ and $\Delta_2 = (s_2, e_2) = (01/06/05, 01/12/05)$ follow the case, $s_2 > s_1$ and $e_2 > e_1$. According to the conflict resolution we analyzed in section 4.2, r_1 will override r'_2 in the period of 01/04/05 to 31/05/05 and r'_2 will override r_1 in the period of /01/06/05 to 01/12/05.

8 Conclusion

Access control in XML documents is very complicated and difficult to manage. Access Policy Sheet (APS) provide us with a concise methodology to resolve the problem. In this paper, we proposed a temporal APS which can handle time related authorizations including conflict resolution, authorization delegation, authorization propagation. Based on our previous work , our new model can be used to satisfy the temporal requirement for XML authorizations.

This paper presents a compositional formal framework for the specification of our temporal access control policies. With a temporal intervals of validity, the authorization would automatically revoked when the associated temporal interval expires. By the hierarchical relationships along with the partial orders defined in APS, we had given the conflict resolutions in several conditions.

Our contribution makes APS be able to handle more complicated XML access control systems and our scheme exhibits significant simplicity to the XML system management.

References

- [1] E.Bertino, P.A. Bonatti and E. Ferrari, A temporal role-based access control model, *ACM Transactions on Information and System Security*,2001,4(3):pp191-233.
- [2] Li Qin, Vijayalakshami Atluri, Concept-Level Access Control for the Semantic Web, *ACM workshop on XML Security*, October ,2003.
- [3] Designing a Distributed Access Control Processor for Network Services on the web. *ACM workshop*, Nov,22, 2002.
- [4] D. Samarati, E.Bertino and S. Jajodia, An Authorization Model for a Distributed Hypertext System, *IEEE Trans On Knowledge and Data Engineering*,8(4), pp 555-562 ,1996.
- [5] Chun Ruan, Vijay Varadharajan and Yan Zhang, A Logic Model for Temporal Authorization Delegation with Negation, *Springer-Verlag,ISC 2003, LNCS 2851*, pp 310-324
- [6] Jing Wu, Yi Mu, Jennifer Seberry, and Chun Ruan, Access Policy Sheet for Access Control in Fine-Grained XML, *The First IFIP Workshop on Trusted and Autonomic Ubiquitous and Embedded Systems (TAUES 2005)*, Lecture Notes in Computer Science, Springer Verlag, 2005, pp. 149-161.
- [7] Chun Ruan and Vijay Varadharajan and Y. Zhang, Logic-based reasoning on delegatable authorizations, *In proc. of the 13th international Symposium on Security and Privacy, IEEE Computer Society Press,1997*, pp 31-42.
- [8] E.Bertino, C.Bettini,E. Ferrari and P. Samarati, An access control model supporting periodicity constraints and temporal reasoning, *ACM Transactions on Database Systems*,1998,23(3): pp231-285.
- [9] Francois Siewe, Antonio Cau, Hussein Zedan, A Compositional Framework for Access Control Policies Enforcement, *FMSE'03, October 30,2003, Washington, DC,USA*,pp 161-172.