Access Policy Sheet for Access Control in Fine-Grained XML

Jing Wu¹, Yi Mu¹, Jennifer Seberry¹, and Chun Ruan²

¹ School of Information Technology and Computer Science,
University of Wollongong, Wollongong, NSW 2522, Australia
{jw91, ymu, jennie}@uow.edu.au

² School of Computing and IT, University of Western Sydney, Penrith, Australia
chun@cit.uws.edu.au

Abstract. We propose an access control scheme for developing authorization rules for XML documents, allowing flexible data granularity and authorization propagation. To simplify the complex access control policies in XML, we introduce a new tool: *Authorization Policy Sheet* (APS). Complex access control rules can be easily described in an APS. The administrator of a system can easily manage the access control of the system. With aid of Data Type Definitions (DTD), the policies given in an APS can be converted into a standard XML code that can be implemented in a normal XML environment.

1 Introduction

Using eXtensible Markup Language has brought profound changes in the nature of information and the usability of the Web. XML can conveniently represent document structure and contents and offers great control over information granularity via transformation and query languages. As XML becomes a widespread data encoding format for Web applications, the Web resource protection must follow to withstand increasing threats from Internet hackers.

One important mechanism to protect Web contents is Access Control. An access control service is needed when some people want to block or allow access to an entire XML instance, while others would like to control access at the tag level. Developing an access control model, and related mechanism, in terms of XML is an important step. With the rapid development of web environments XML data access control has been intensively studied (e.g., [2, 8, 4, 5, 3, 1, 7]). However, these existing research works on XML do not offer more advanced access control features such as authorization delegation.

The popular mechanism in XML based access control takes advantage of Data Type Definitions (DTD) (e.g., [1]). With a DTD based approach, the access control rules are defined in DTD. A DTD based approach is sometimes employed along with a customized XML access control language where the rules are described using the language [6]. The merit of this kind of scheme is that the system administrator can enter/update access control rules much more conveniently.

In practice, access control components come with an order. For example, an order could be: **super_user** > **group_member** > **others**, where **super_user** holds the highest privilege and **others** hold the least privilege. In objects, an order can be associated with the depth the objects reside in a tree or a directory. Access control in such systems implies the propagation of authorization in terms of the associated order. In this paper, we present a novel framework for fine grain XML access control that includes delegation, in terms of the partial order of access control components. With our scheme, the complicated access control task in XML documents becomes easy, since we propose a novel access control “spreadsheet” tool for describing rules. We call it *Authorization Policy Sheet* (APS). Using an APS with the associated DTD forms a normal XML code that is understandable to a normal XML environment.

The rest of the paper is organized as follows. In Section 2, we describe our access control model and define the major components in our model. In Section 3, we present the authorization policy sheet (APS), which forms the foundation to our model. In Section 4, we provide the definitions of predicates. The final section is the conclusion.

2 Basic Definitions

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.

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.

Here is the example of the subject hierarchy with three subjects (**Admin**, **Alice**, **Bob**), described in a separate sheet:

```

<subject>
  <users>
    <name> Admin </name>
  </users>
  <users>
    <name> Alice </name>
  </users>
  <users>
    <name> Bob </name>
  </users>
</subject>

```

We have omitted “partial order” of the associated subjects such as Admin > Bob > Alice as this will be presented later on.

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>
	]>	

Here is the example of the object hierarchy described in a separate XML sheet:

```

<object>
  <object>
    <target> hospital.xml </target>
    <path> //doctor/operation_info </path>
  </object>
  <object>
    <target> hospital.xml </target>
    <path> //doctor/personnel_info/alice </path>
  </object>
</object>

```

Access Rights. Ordered rights are defined as 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>
]>
```

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

The ordered authorization types are described in DTD:

```
<!DOCTYPE authorization_type[
  <!ELEMENT authorization_type (n|p|d) #REQUIRED>
]>
```

3 Authorization Policy Sheet

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.

3.1 Rules

An APS sheet consists of a finite set of rules. 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:

```
<rule:r>
  <p1, attribute> <- <condition>
  <p2, attribute> <- <condition>
  <p3, attribute> <- <condition>

  <pn, attribute> <- <condition>
</rule:r>
```

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)*>
]>
```

A rule in XML is defined as following:

```
<rule:r>
  <predicate>
    <attribute>
      <attr>
    </attribute>
    ...
    <attribute>
      <attr>
    </attribute>
    <condition>
      ...
    </condition>
  </predicate>
  ...
</rule:r>
```

3.2 Partial Order

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. In an APS, the partial orders are respectively defined in the form: $(s_1 > s_2 > s_2 > \dots)$, $(o_1 > o_2 > o_3 > \dots)$, $(t_1 > t_2 > t_3 > \dots)$, $(a_1 > a_2 > a_3 > \dots)$.

4 Predicates

Predicates form the essential part of an APS. In our system, there is a set of predicates: $P = \{p_1, p_2, \dots, p_n\}$. Every predicate is constructed in the form $\langle p_i, x_1, \dots, x_n \rangle$. $x_1, \dots, x_n$ are terms associated with the predicate. We utilize following predicates in an APS.

4.1 grant

Definition 1. (*grant*) *grant* is a 6-tuple predicate $S \times O \times T \times A \times S \times F$: $\langle \text{grant}, s, o, t, a, g, f \rangle$, where subject $s \in S$ is granted by grantor $g \in G$ the access right $a \in A$ on object $o \in O$ with the type $t \in T$. It determines whether a subject is granted an access right over an object. In an APS, this rule reads:

```
<grant grantee=" ", target+path=" ", authorization_type=" ",
      access_right=" ", grantor=" ", status=" ">
```

Predicate **grant** in APS is an authorization rule, where the element **grant** has attributes **grantee**; **target + 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, **authorization_type**, **access_rights**, **grantor**, and **status**. **status** is a flag indicating whether or not the rule is effective.

For example,

The merit of **grant** in ASP is obvious. The **grant** is also defined in DTD as follows.

```
<!DOCTYPE grant[
<!ELEMENT grant (subject,object,authorization_type,
                  access_right,subject,status)>
<!ELEMENT subject (grantee)>
<!ELEMENT grantee (name)>
<!ELEMENT name #PCDATA>

<!ELEMENT object (target,path)>
<!ELEMENT target href #PCDATA>
<!ELEMENT path #PCDATA>

<!ELEMENT authorization_type (n|p|d) #REQUIRED>
<!ELEMENT access_right (a*) #IMPLIED>
<!ELEMENT a #PCDATA>

<!ELEMENT subject (grantor)>
<!ELEMENT grantor (name)>
<!ELEMENT name #PCDATA>

<!ELEMENT status (true|false) #REQUIRED>
]>
```

The **grant** rule defined in ASP and DTD is converted into a standard form of XML. Here is an example of **grant** in APS and XML:

– rule in APS:

```
<rule:hospital>
  <grant, grantee="Alice",
    target+path="hospital_info.xml
                + //hospital/operation_info/",
    authorization_type="p",
    access_right="Read",
    grantor="Bob",
    status="True">
</rule:hospital>
```

– rule converted to XML:

```
<rule:hospital>
  <grant>
    <subject>
      <grantee>
        <name> Alice </name>
      </grantee>
    </subject>
    <object>
      <target>hospital_info.xml </target>
      <path>//hospital/operation_info</path>
    </object>
    <type> p </type>
    <access_right> Read </access_right>
    <subject>
      <grantor>
        <name> Bob </name>
      </grantor>
    </subject>
    <status> True </status>
  </grant>
</rule:hospital>
```

which reads that the grantee Alice is granted by the grantor Bob the access right read on Xpath specified object

hospital_info.xml + //hospital/operation_info/
with the authorization type p and the status is set to True.

4.2 cangrant

Differing from **grant**, the predicate **cangrant** represents the capability of a subject in granting a right with respect to an object to another subject. Formally, we define it as follow.

Definition 2. (*cangrant*) *cangrant* is a 4-tuple predicate $S \times O \times A \times F: < \text{cangrant}, s, o, a, f >$ where $s \in S$ is the grantor; $o \in O = \text{target} + \text{path}(V, E)$, *target* is an XML or DTD, *path* is an XPath expression that eventually selects specific portions (object) of the XML document in XML tree, V is a set of nodes and E is a set of edges; $a \in A$ is an access right; and $f \in F$ is the status of the rule. *cangrant* determines whether a subject can grant an access right over an object.

The definition above states that Subject s has the right to grant access right a on object o to other subjects.

In APS, we define the *cangrant* as

```
<cangrant subject=" ", target+path=" ", access_right=" ",
                                status=" ">
```

In DTD, *cangrant* is defined as follows:

```
<!DOCTYPE cangrant[
<!ELEMENT cangrant (subject,object, access_right,
                    status)>
<!ELEMENT subject (grantee)>
<!ELEMENT grantee (name)>
<!ELEMENT name #PCDATA>

<!ELEMENT object (target,path)>
<!ELEMENT target href #PCDATA>
<!ELEMENT path #PCDATA>

<!ELEMENT access_right (a*) #IMPLIED>

<!ELEMENT status (true|false) #REQUIRED >
]>
```

The following is an example of *cangrant* in XML.

– rule in APS:

```
<rule:hospital>
  <cangrant, subject="Alice",
    target+path="hospital_info.xml
                + //hospital/operation_info",
    access_right="Read",
    status="True">
</rule:hospital>
```

– rule can be converted XML (omitted).

This rule states that Alice can grant the right *read* with respect to the object *//hospital_info + //hospital/operation_info* to other subjects.

4.3 Delegation

We define the *delegation* right d , which allows a subject who holds an access right to grant the right to another subject. A subject can grant other subjects an access right a over object o if the subject has an associated **cangrant** right and s is the security administrator **admin**, or s has been granted a over o with delegable type d . Clearly, the type d is a flag indicating whether or not the access right can be further granted to another subject by the holder of the access right.

We also assume that if subject s receives a delegable authorization directly or indirectly from another subject s' on some object o and access right a , then s cannot grant s' authorization on the same o and a later on.

For example, Alice is granted an access right **Read|Write** of type **p|d** over object **hospital_info.xml + //hospital/operation_info/**, then we have

– rule in APS:

```
<rule:hospital>
  <grant, grantee="Alice",
    target+path="hospital_info.xml
                + //hospital/operation_info/",
    authorization_type="p|d",
    access_right="Read|Write",
    grantor="Bob",
    status="True">
</rule:hospital>
```

This rule implies that Alice can grant the access right with respect to o and a to another subject. In other words, the list of rules can be updated whenever Alice takes the action.

With rule:hospital and **cangrant** right,

```
<cangrant,subject="Alice",
  target+path="hospital_info + *",
  access_right="Read",
  status="True">
```

A new rule can be generated by Alice (notice that **Write** access is forbidden):

```
<rule:hospital2>
  <grant, grantee="Cindy",
    target+path="hospital_info.xml
                + //hospital/operation_info/",
    authorization_type="p",
    access_right="Read",
    grantor="Alice",
    status="True">
</rule:hospital2>
```

where we have assumed that **d** denotes the non-inherited delegation. This means that the delegatee Bob cannot further delegate the right to other parties.

5 Conflict Resolution

Our model supports different types of access rights, authorizations can be in conflict where a user is granted two different types of access rights. Thus a proper conflict resolution policy is needed. We solve conflicts as follows. First, trace the delegation relation path explicitly. When a conflict occurs, we will see if the two grantors fall into a delegation path. If they do, then let the authorization with the grantor as the predecessor in the path override the other one. In other words, along the delegation path, the predecessors' grants have higher priority than the successors' grants. This policy can support well-controlled delegation. Second, if the conflicts can not be solved by the above policy, we will use Negative-takes-precedence based policy to resolve the conflicts. That is, we will resolve the conflicts according to their types, and the priority sequence is $n > p > d$. This policy favours security.

In Conclusion, we have presented an access control model that supports fine-grained XML. Our model has the following main merits. simplicity in access control management. Our access control model makes the complex access control management much more effective, simple and elegant. Due to lack of space, we will present the propagation of authorization in the full version of the paper.

References

1. E. Bertino and E. Ferrari. Secure and selective dissemination of XML documents. *ACM Transaction on Information and System*, 5(3), 2002.
2. S. Castano, M. Fugini, G. Martella, and P. Samarati. *Database Security*. Addison-Wesley, 1995.
3. E. Damiani, S. D. C. di Vimercati, E. Fernandez-Medina, and P. Samarati. An access control system for svgdocuments. In *Proceedings of the Sixteenth Annual IFIPWG 11.3 Working Conference on Data and Application Security*, pages 121–135, 2000.
4. E. Damiani, S. Vimercati, S. Paraboschi, and P. Samarati. Design and implementation of an access processor for XML documents. In *Proceedings of the 9th international WWW conference*, pages 59–75, 2000.
5. E. Damiani, S. Vimercati, S. Paraboschi, and P. Samarati. Securing XML document. In *Proceedings of International Conference on Extending Database Technology (EDBT2000)*, pages 121–135, 2000.
6. A. Gabillon and E. Bruno. Regulating access to xml documents. In *Proceedings of the fifteenth annual working conference on Database and application security*, pages 299–314, 2001.
7. M. Kudo and S. Hada. XML document security based on provisional authorizations. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 87–96, Athens, Greece, 2000.
8. C.-H. Lim, S. Park, and S. H. Son. Access control of XML documents considering update operations. In *Proceedings of ACM Workshop on XML Security*, pages 49–59, 2003.