University of Wollongong
Research Online

Faculty of Informatics - Papers (Archive)

Faculty of Engineering and Information Sciences

2005

Access Policy Sheet for Access Control in Fine-Grained XML

J. Wu
University of Wollongong, jw91@uow.eduau

Y. Mu
University of Wollongong, ymu@uow.edu.au

Jennifer Seberry
University of Wollongong, jennifer_seberry@uow.edu.au

C. Ruan
University of Western Sydney

Publication Details

This article was originally published as Wu, J, Mu, Y, Seberry, J and Ruan, C, Access Policy Sheet for Access Control in Fine-Grained XML, Proceedings of The First IFIP workshop on Trusted and Autonomic Ubiquitous and Embedded Systems (TAUES 2005), Nagasaki - Japan, 6-9 December 2005. Also available at Lecture Notes in Computer Science, 3813, 1273-1282. Copyright Springer-Verlag 2005.

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

Access Policy Sheet for Access Control in Fine-Grained XML

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.

Disciplines

Physical Sciences and Mathematics

Publication Details

This article was originally published as Wu, J, Mu, Y, Seberry, J and Ruan, C, Access Policy Sheet for Access Control in Fine-Grained XML, Proceedings of The First IFIP workshop on Trusted and Autonomic Ubiquitous and Embedded Systems (TAUES 2005), Nagasaki - Japan, 6-9 December 2005. Also available at Lecture Notes in Computer Science, 3813, 1273-1282. Copyright Springer-Verlag 2005.

This conference paper is available at Research Online: http://ro.uow.edu.au/infopapers/276

Access Policy Sheet for Access Control in Fine-Grained XML

Jing Wu[1], and Yi Mu[1], Jennifer Seberry[1], and Chun Ruan[2]

[1] School of Information Technology and Computer Science
University of Wollongong, Wollongong, NSW 2522, Australia
Emai: {wsusilo,ymu}uow.edu.au
[2] School of Computing and IT, University of Western Sydney
Penrith, Australia Email: chuncit.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 and propagation.

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 and propagation of authorization, in terms of the partial order of access control components. With our scheme, the complicated assess 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. In Section 4, we introduce the authorization propagation. 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.

2.1 Document Type Definition (DTD)

A Document Type Declaration can be attached to XML file and specifies the rules or Document Type Definition (DTD) that XML file may follow. A DTD consists of two parts: the element declarations and the attributions. The element declarations part specifies the structure of the elements contained in the document. The attribute list declarations part specifies the list of its attributes, in terms of names, types, optionally clause, and default values.

2.2 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 $(S, >)$: $admin, s_1, s_2, \cdots, s_n$ denote ordered subjects with the order of $admin > s_1 > s_2 > \cdots > s_n$. 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 $(O,>)$: $o_1, o_2, \cdots$ with the order $o_1 > o_2, > ...$ 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 access rights are defined as constant poset $(A, >)$: $a_1, a_2, \cdots$ with the order: $a_1 > a_2 > ...$ 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: the access right is forbidden.
- p: specifies the access right is granted.
- d: specifies the access right is delegable.

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

1. Orders of the objects and subjects,
2. 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, `p1, p2,...,pn` 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, ...,attribute>
            ...
        <condition>
          ...
        </condition>
</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:

```
s1 > s2 > s2 > ...
o1 > o2 > o3 > ...
t1 > t2 > t3 > ...
a1 > a2 > a3 > ...
```

4 Predicates

Predicates form the essential part of an APS. In our system, there is a set of predicates:

$$P = \{p_1, p_2, ..., p_n\}.$$

Every predicate is constructed in the form $< p_i, x_1, ..., x_n >$. $x_1, ..., x_n$ 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* $S \times O \times T \times A \times S \times F$*:* $< grant, s, o, t, a, g, f >$*, 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* $S \times O \times A \times F$*:* $<$ *cangrant*, $s, o, a, f >$ *where* $s \in S$ *is the grantor;* $o \in O =$ `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,* 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` converted to XML:

```
<rule:hospital>
  <cangrant>
      <subject>
          <grantee>
              <name> Alice </name>
          </grantee>
      </subject>
      <object>
          <target> * </target>
          <path>//hospital/operation_info</path>
      </object>
      <access_right> Read </access_right>
      <status> True </status>
  </cangrant>
</rule:hospital>
```

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. To allow further delegation, we use the flag `d+`. If we replace `d` with `d+`, then we have

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

Thus, the access right is delegated to `Cindy`, who can then grant it to another party.

4.4 Authorization Propagation

Our model supports the implicit authorizations by permitting rules propagation. A rule based authorization specification allows implicit authorizations to be derived from the given authorization set. In APS, we give an explicit authorization set and they will derive implicit rules automatically by propagation. Using authorization propagation can greatly reduce the size of an authorization set.

Let $S_1, S_2, \cdots, S_n$ be n sets of subjects and $O_1, O_2, \cdots, O_m$ be m sets of objects. As we have described earlier about the partial order of our system, subjects in S_i, objects in O_i, and authorization rights satisfy the partial order rule. The authorization propagation is based on the partial order rule.

In a set S_i, the access right held by a subject with a lower order propagates to the subjects with a higher order. For instance, in S_i, Alice has the order of 2, Bob has the order of 3, and the administrator has the order of 1; that is, Administrator $>$ Bob $>$ Alice. If Alice can read the object o_1, then by authorization propagation Bob and administrator can also read o_1. If Bob can read the object o_2, then Alice cannot read o_2 unless the administrator or Bob grants the read access right to her. The administrator possesses the access right of Alice and Bob by propagation.

For an object set O_i, the authorization associated with objects with a lower order propagates to the object with a higher order. For instance, the descendant of a node can be read by an user, then the node can also be read by this user. Assuming `//operation/files/patient` is the object set with the partial order `operation` $>$ `files` $>$ `patient`. If a subject who can read on `patient` can also read `files` and `operation` by propagation.

For an access right set A with the partial order $a_1 > a_2 > \cdots$, if a subject holds the access right with a higher order implies that it has the access rights with a lower order. For instance, for $a_1 =$ `Write` and $a_2 =$ `Read`, if the subject s_1 has `Write` access to an object o_1, the then s_1 also has the `Read` access to o_1.

Definition 3. (Propagation) *Let* (s, o, a) *be a 3-tuple identifier which indicates what the corresponding rule is about in terms of subject, object and access right. If there exists a relation of* $s' > s, o' > o, a' > a$ *the rule with respect to* (s', o', a') *could replace the rule with respect to (s,o,a). We say constant c is minimal in a rule r with respect to the partial order "* $>$ *", if no constant* c' *in r exists such that* $c > c'$*. We use* c_{min} *to define the lowest order in a partial order chain. Let* S=$\{s_1 > s_2 > \cdots > s_{min}\}$*,* O=$\{o_1 > o_2 > \ldots > o_{min}\}$*,* A=$\{a_1 > a_2 > \ldots > a_{min}\}$*,* $E = \{(s_i, s_j, a_k)\}$*,* $I = \{(s_{i'}, o_{j'}, a_{k'}),\ i' = 1 \ldots i-1; j' = 1 \ldots j-1; k' = 1 \ldots k-1;\}$*.* s_i, o_i *and* a_k *are the authorization vocabularies in the partial order chain.*

Definition 4. (Implicit Rule) *An implicit rule is defined with a rule and the associated partial order of subjects, objects, access rights, and the propagation rules given above, can be generated automatically.*

In the following, we provide an example of propagation by using the sets:

- Subjects: `Admin` > `Bob` > `Alice`
- Objects: `hospital_info.xml + //hospital/operation_info/`
- Access rights: `Read` > `Write`

The explicit rule in APS reads:

```
<rule:hospital3>
    <grant, grantee="Alice",
    target+path="hospital_info.xml + //hospital/operation_info/",
    authorization_type="p",
    access_right="Write",
    grantor="Bob",
    status="True">
 </rule:hospital3>
```

The implicit rules associated with `rule:hospital3` and the partial orders are automatically generated and described as follows:

```
<rule:hospital3>
    <grant, grantee="Alice",
     target+path="hospital_info.xml + //hospital/operation_info/",
     authorization_type="p",
     access_right="Write",
     grantor="Bob",
     status="True">
    <grant, grantee="Alice",
     target+path="hospital_info.xml + //hospital/operation_info/",
     authorization_type="p",
     access_right="Read",
     grantor="Bob",
     status="True">
```

```
    <grant, grantee="Admin",
     target+path="hospital_info.xml + //hospital/operation_info/",
     authorization_type="p",
     access_right="Write",
     grantor="Bob",
     status="True">
    <grant, grantee="Admin",
    target+path="hospital_info.xml + //hospital/operation_info/",
    authorization_type="p",
    access_right="Read",
    grantor="Bob",
    status="True">
</rule:hospital3>
```

Observe that Alice is also granted the Read access right and Admin is granted with both Write and Read access rights.

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 $\mathtt{n} > \mathtt{p} > \mathtt{d}$. This policy favours security.

6 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. By introducing APS, the security administrator needs only update the rules in the APS. These rules are defined with conciseness, so updating them is easy. The rules in an APS can automatically be converted into XML which is readable to the normal XML environment.
- Conciseness due to authorization propagation. We utilize partial-order rules in APS which allow implicit rules. Implicit rules are obtained with the partial order of associated subjects, objects, and access rights and with normal/explicit rules. Implicit rules do not need to be described in the APS. They can be converted into rules in XML automatically.

Our access control model makes the complex access control management much more effective,simple and elegant.

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*. Addision-Wesley, 1995.
3. E. Damiani, S. D. C. di Vimercati, E. Fernndez-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. Paraboaschi, 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. Paraboaschi, 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.