Regulating Access to Semistructured Information on the Web

E. Damiani[1] S. De Capitani di Vimercati[2] S. Paraboschi[3] P. Samarati[1]
(1) Università di Milano, Milano, Italy
(2) Università di Brescia, Brescia, Italy
(3) Politecnico di Milano, Milano, Italy

Abstract The remarkable growth of the World Wide Web in recent years has made it possible to distribute information to users in the form of an unorganized and unstructured collection of documents. While security is an important aspect in such a scenario, access control systems available today result too rigid and limited. We present an approach to specify and enforce access restrictions to Web documents. The approach provides flexible, as it allows to enforce a variety of security policies and requirements at a fine-grained level without affecting the data organization.

1. INTRODUCTION

Internet, the Web in particular, is emerging today as a primary means to acquire and make information available to others. The large amount and diversity of information that needs to be shared calls for the support of flexible models and techniques to manage it. Data models and languages are making considerable steps in this direction, as witnessed by the large body of research and development activities around multimedia and *semistructured* data management [Buneman, 1997, Papakonstantinou et al., 1995, Siméon and Smaga, 1998]. Semistructured data models allow the organization, storing, and manipulation of information with loosely defined or irregular structure thus providing a flexible means for publicizing information on the Web. Communication and sharing of this loosely structured information requires the adoption of a uniform standard to transfer data. The *eXtensible Markup Language* [Bray et al., 1998], recently proposed by the World Wide Web Consortium as a new standard for Web publishing, appears a promising solution to this problem [Goldman et al., 1999]. By maintaining interface specifications separate from the actual data, XML allows applications to exchange data among each other, without requiring prior knowledge of incoming data formats. In this distributed and heterogeneous context, security is clearly a critical problem and the model to express and enforce access restrictions must provide the flexibility and expressiveness needed in the large network infrastructure. Current security solutions (e.g., Apache's access control) support access restrictions traditionally used for operating systems, therefore resulting much limited

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35515-3_53

S. Qing et al. (eds.), *Information Security for Global Information Infrastructures*

in this new context. In particular, specification of authorizations at the file system level does not provide support for different restrictions within a document, at the point that security considerations could affect the data organization itself. The variety and dynamic nature of access restrictions that can arise in such a context makes this a considerable limit. The lack of a flexible and powerful access control model for Web documents is probably due to the inherent limitations of the HTML language traditionally used to publish documents. HTML provides no clean separation between the structure and layout of a document, and access control proposals developed for HTML documents [Samarati et al., 1996] remains therefore limited. XML represents an important opportunity to solve this problem. By exploiting XML data organization it is possible to define access restrictions directly on the structure and content of documents. In [Damiani et al., 2000] we started the investigation of an authorization model to regulate access to information in XML documents. In this paper, we extend the work by supporting different types of authorizations and overriding policies, and present an algorithm for their enforcement. Our model provides flexible as it allows the representation of different security policies and protection requirements.

2. OVERVIEW OF XML

XML [Bray et al., 1998] is a markup language for describing semistructured information. An XML document is composed of a sequence of nested elements, each delimited by a pair of start and end tags. Figure 1 shows an example of XML document including information about a department of an organization, which is assumed to have one `R&D` division composed of the `Security` group. XML documents can be classified into two categories: *well-formed* and *valid*. An XML document is well-formed if it obeys the syntax of XML (e.g., non-empty tags must be properly nested, each non-empty start tag must correspond to an end tag). A well-formed document is valid if it conforms to a proper *Document Type Definition* (DTD). A DTD is a file (external or included directly in the XML document) which contains a formal definition of a particular type of XML document. A DTD includes declarations for elements and attributes. Element declarations in the DTD specify the names of elements, and their sub-elements, possibly associated with a cardinality constraint: "*" indicates zero or more occurrences, "+" indicates one or more occurrences, "?" indicates zero or one occurrence, and no label indicates exactly one occurrence. Attributes represent properties of elements. Attribute declarations specify, for each element, its attribute names, types, and, possibly, default values. Figure 2 shows the DTD for the XML document in Figure 1.

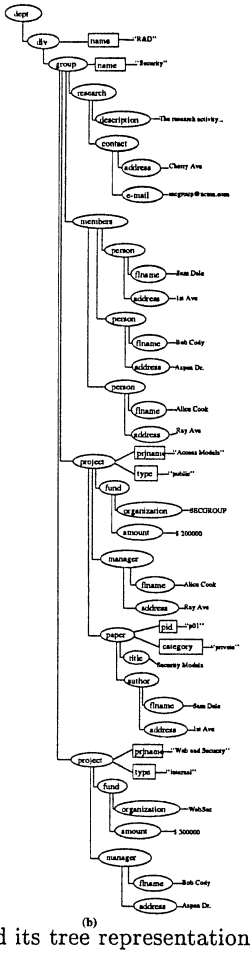
3. PROTECTION REQUIREMENTS

Before introducing our access control system, we discuss two basic features that an authorization model for regulating access to Web documents should support.

Coarse and fine object granularity. It should be possible to specify authorizations at both a coarse as well as at a fine grain object level. The advantage of supporting fine grain specifications is obvious: specifying requirements on individual pieces of information. The support of coarse grain specifications allows instead the factorization of common requirements, which can be stated as one. With respect to a document's content, granularity considerations call for the support of authorizations on individual elements as well as on whole portions of a document. At a higher level, granularity

```
<dept>
  <div name = "R&D">
    <group name = "Security">
      <research>
        <description>
          The research activity is in the area
          of computer security
        </description>
        <contact>
          <address> Cherry Ave </address>
          <e-mail> secgrp@acme.com </email>
        </contact>
      </research>
      <members>
        <person>
          <flname> Sam Dale </flname>
          <address> 1st Ave </address>
        </person>
        <person>
          <flname> Bob Cody </flname>
          <address> Aspen Dr </address>
        </person>
        <person>
          <flname> Alice Cook </flname>
          <address> Ray Ave </address>
        </person>
      </members>
      <project prjname = "Access Models" type = "public">
        <fund>
          <organization> SECGROUP </organization>
          <amount> $ 200000 </amount>
        </fund>
        <manager>
          <flname> Alice Cook </flname>
          <address> Ray Ave </address>
        </manager>
        <paper pid = "p01" category = "private">
          <title> Security Models </title>
          <author>
            <flname> Sam Dale </flname>
            <address> 1st Ave </address>
          </author>
        </paper>
      </project>
      <project prjname = "Web and Security" type = "internal">
        <fund>
          <organization> WebSec </organization>
          <amount> $ 300000 </amount>
        </fund>
        <manager>
          <flname> Bob Cody </flname>
          <address> Aspen Dr </address>
        </manager>
      </project>
  </div>
</dept>
```

(a)



(b)

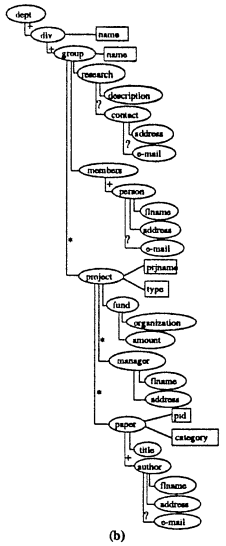
Figure 1 An example of XML document (a) and its tree representation (b)

considerations require the support of authorizations on single documents as well as on sets of documents, identified with reference to their schema (DTD).

Exception support. The use of authorizations at a coarse granularity level would result limited without the support of exceptions. For instance, a requirement stating that all instances of a DTD, but a specific document, can be accessed would need to be translated in several authorizations: one for each instance of the DTD, but the one that must be protected. The support of both permissions and denials, and the consequent exception policy, allows instead the same requirement to be represented with two authorizations: a positive authorization on the DTD, and a negative authorization on the specific document to be protected. An analogous reasoning applies to exceptions within documents (e.g., the whole document but a particular element/attribute within it can be released). It is important to note that when dealing with exceptions, the exception policy has to be flexible. In particular, if a built-in policy is assumed, it should be possible to reverse it [Jajodia et al., 1997].

```
<!ELEMENT dept (div)+>
<!ELEMENT div (group)+>
<!ELEMENT group (research,members,project*)>
<!ELEMENT research (description,contact?)>
<!ELEMENT description (#PCDATA)*>
<!ELEMENT contact (address,e-mail?)>
<!ELEMENT members (person)+>
<!ELEMENT person (flname,address,e-mail?)>
<!ELEMENT flname (#PCDATA)>
<!ELEMENT address (#PCDATA)*>
<!ELEMENT e-mail (#PCDATA)>
<!ELEMENT project (fund,manager*,paper*)>
<!ELEMENT fund (organization,amount)>
<!ELEMENT organization (#PCDATA)>
<!ELEMENT amount (#PCDATA)>
<!ELEMENT manager (flname,address)>
<!ELEMENT paper (title,author+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (flname,address,e-mail?)>
<!ATTLIST div   name CDATA #REQUIRED>
<!ATTLIST group   name CDATA #REQUIRED>
<!ATTLIST project   prjname CDATA #REQUIRED
  type CDATA #REQUIRED>
<!ATTLIST paper   pid ID #REQUIRED
  category (public|private) #REQUIRED>
```

(a)



(b)

Figure 2 An example of DTD (a) and its corresponding graphical representation (b)

The next section illustrates the components of authorizations in our model and how they address the requirements above.

4. ACCESS AUTHORIZATIONS

Access authorizations determines the accesses that the system should allow (or deny). In our model, each authorization is a 5-tuple ⟨*subject*, *object*, *action*, *sign*, *type*⟩, where *subject* is the subject for which the authorization is intended, *object* is the object to which the authorization refers, *action* indicates the operation authorized/denied by the authorization (for simplicity, we consider action to be the `read` access), *sign* indicates whether the authorization states a permission ('+') or a denial ('−'), and *type* regulates whether the authorization propagates to other objects and how it interplays with other authorizations (exception policy). We now discuss the subject, object, and type fields.

Subject. Our model allows authorizations to be referred to specific user identities (e.g., `Tom`) as well as groups (e.g., `Managers` or `Public`) defined on them. Moreover, in consideration of the fact that we operate in a distributed context, authorizations can refer to hosts, that is, location addresses from which requests may (or may not) originate. Location addresses can be specified in addition or in alternative to identities and can make use of patterns denoting sets of addresses (e.g., `130.89.*` denotes all the machines in subnetwork `130.89`).

Object. Our model supports different levels of granularity. The schema vs instance specifications are easily supported by referring authorizations to either the URI of the DTD (schema level) or of the document (instance level). Reference to the finer element and attribute grains is supported through *path expressions*, which are specified in the XPath language [XPa, 1999]. A path expression identifies one or more elements/attributes within a document though the specification of a sequence of labels $l_1/l_2/\ldots/l_n$, where each l_i is either an element or a predefined function (e.g.,

child is a function returning the children of a node), and the last label can be either an element or an attribute prefixed by @. The name of a function and its argument are separated by a double colon '::'. A path expression may also specify conditions that elements need to satisfy to be identified by the path expression. Conditions are distinguished from navigation specifications by enclosing them within square brackets. For instance, with respect to the DTD in Figure 2, path expression "/dept/div/group/project[./@type="public"]/child::*[position() ≥2]" identifies the second to the last child of public projects.

Type. The type field defines how the authorizations must be treated with respect to propagation at finer granules and overriding (exception support). By default, we assume that authorizations specified on a DTD apply to all its instances, but may be overridden by authorizations explicitly specified on the document, according to the principle that the *more specific authorization takes precedence* [Lunt, 1989]. This propagation/overriding policy is intuitive and natural, and we can expect it to apply in most cases. We can however imagine scenarios where this policy may need to be bypassed (reversed). For instance, one may wish to specify authorizations at the schema level that do not allow exceptions (*hard* statements). Analogously, one may wish to specify authorizations at the instance level that behave as default rules, in case no schema level statement has been made, but are not intended to override them otherwise (*soft* statements). A further option associated with authorizations regulates whether the authorization on an element (i.e., an element that satisfies the path expression) applies only to the element itself, that is, to its direct attributes, or to all its content, that is, recursively to its subelements. In the first case the authorization is *local*; in the second case it is *recursive*.

The combination of the options above, introduces eight authorization types: LDH (Local DTD Hard), RDH (Recursive DTD Hard), LD (Local DTD), RD (Recursive DTD), L (Local), R (Recursive), LS (Local Soft), and RS (Recursive Soft). Types are used by the access control system according to the following principles: (1) DTD level authorizations propagate to all instances of the DTD; (2) recursive authorizations on a node recursively propagate to all its subnodes; (3) local authorizations on a node propagate to its direct attributes; (4) propagation is stopped whenever a more specific authorization is found on the node; and (5) instance level authorizations, unless declared as soft, take precedence over DTD level authorizations, unless declared as hard. Intuitively, principles 1 through 3 regulate what is propagated, principle 4 determines how it is propagated, and principle 5 determines how defined/propagated authorizations are used to produce the final decision.

Example 4.1 Consider the XML document and DTD illustrated in Figure 1 and Figure 2, respectively. Some examples of authorizations are as follows.
Organization's policy specified at the DTD level. It refers to the organization in its entirety and applies to all its departments.

- The name of divisions and groups is publicly accessible.
  ⟨(Public,*),/dept/div//@name,read,+,LD⟩
- The research activity and members of any group in any division is publicly accessible.
  ⟨(Public,*),/dept/div/group/child::*[position()≤ 2],read,+,RD⟩
- Access to "private" papers *must* be explicitly forbidden to nonmembers of the department, no instance-level exceptions allowed (hard authorization).
  ⟨(NonMembers,*),/dept/div//paper[./@category = "private"],read,−,RDH⟩

Department's policy specified on document in Figure 1. It may complement or override the policy stated by the organization.

- Managers connected from machines in network `130.*` can access projects of the `Security` group. Everybody else is explicitly forbidden to access "internal" projects.
  `⟨(Manager,130.*),/dept/div[./@name="R&D"]/group[./@name="Security"]/project,read,+,R⟩`
  `⟨(Public,*),/dept/div[./@name="R&D"]/group[./@name="Security"]/project[./@type="internal"],read,−,R⟩`
- The name of "public" projects is publicly accessible, unless the organization's policy states otherwise (soft authorization).
  `⟨(Public,*),div/group/project[./@type="public"]/@prjname,read,+,LS⟩`
- Users connected from host `130.89.56.8` and members of the `Security` group can read the name of "internal" security projects.
  `⟨(Security,130.89.56.8),/dept/div[./@name="R&D"]/group[./@name="Security"]/project[./@type="internal"]/@prjname,read,+,L⟩`

5. AUTHORIZATION ENFORCEMENT

The access control system mediates all the requests to documents and evaluates them against the authorizations. For each request, it produces the view of the document composed only of the information that the requester can access. The expressiveness of the language and the fine granularity at which authorizations can be specified make the access control enforcement far from being trivial. In particular, each access request may imply the consideration of several authorizations, each applying to specific elements/attributes of the documents. The elements/attributes to which an authorization applies are determined by the path expression associated with the authorization; the type of the authorization regulates instead whether it recursively applies to the element's content. It is therefore important that the access control mechanism enforces authorizations in an efficient way. For this reason, our system first evaluates all authorizations at once and translates them into labels ('+' for permission, '−' for denial, 'ε' for no specification) associated with element and attribute nodes of the tree structure of the documents, and then enforces propagation/overriding of the labeling by navigating through the tree. The consideration of label 'ε' is particularly important in the enforcement of the propagation and overriding policies (see Section 4). Indeed, authorizations propagate down (from an element to its content, and from a DTD to a document) to a node only if *not overridden by a more specific authorization*, meaning if neither a '+' nor a '−' is already associated with the node. Intuitively, the propagation of authorizations from a node labeled b to a node labeled a can be expressed by assigning to a the result returned by statement "**if** $a = \varepsilon$ **then** b".

By interpreting labels as values of a three-value logic [Rescher, 1969], we can conveniently express such a statement as a simple logic formula. To this end, we first need to map '+', '−', and 'ε' in the logic. The only condition that such mapping must satisfy is that 'ε' must be mapped to 0 (`false`). To understand the reason for this, think of `false` as "no statement" has been made. Signs '+' and '−' must then be mapped to the other two values, namely 1 (`true`), and $\frac{1}{2}$ (`indeterminate`); whatever choice would do. Here, we map '+' to 1 and '−' to $\frac{1}{2}$. It is easy now to see that, with the defined mapping, statement "**if** $a = \varepsilon$ **then** b" is equivalent to formula "$a \vee (\neg a \wedge b)$". In the following, we denote such a formula as $a \oplus b$.

INPUT: A requester *rq*, the tree T of the requested document, and the set of authorizations associated with the document.
OUTPUT: The tree corresponding to the document view to be returned to *rq*.

1. Determine the set A_{rq} of authorizations specified on the document applicable to *rq*
2. **For each** authorization $a \in A_{rq}$ **do**
   Evaluate *object*(a) on T to determine the set $N_a \subseteq T$ of nodes to which a applies
   For each $n \in N_a$ **do**
   If *sign*(a)= '+' **then** ENQUEUE(*subject*(a),$Label_n$[*type*(a)].*Allowed*)
   else ENQUEUE(*subject*(a),$Label_n$[*type*(a)].*Denied*)
   For each node n in T and type $t \in$ {LDH,RDH,L,R,LD,RD,LS,RS} **do**
   Determine $Label_n[t].sign$ by composing $Label_n[t].Allowed$ and $Label_n[t].Denied$ according to the policy
3. Traversing tree T in preorder:
   For each node n **do**
   Let p be the parent of n
   case n of
   attribute: $Label_n := Label_n \vec{\oplus} Label_p$
   element: $Label_n := Label_n \vec{\oplus}$ **mask_local**($Label_p$)
   $final_n := \epsilon$
   For t in [LDH, RDH, L, R, LD, RD, LS, RS] **do** $final_n := final_n \oplus Label_n[t].sign$
4. Traversing tree T in postorder:
   Delete all leaf nodes n such that $final_n \neq$ '+'

Figure 3 Access control algorithm

Figure 3 illustrates the complete access control process. It is composed of four basic steps that we now describe in more details.

Step 1: Authorization retrieval. It consists in finding, among the authorizations associated with the document, specified either at the instance or schema level, those that apply to the requester. Authorizations that apply to a requester *rq*=(*user,location*) are those authorizations specified for a subject (*ug,host*) such that *user* is equal to, or is a member of, *ug* and *location* is a machine within *host*.

Step 2: Initial labeling. The initial labeling step evaluates the authorizations retrieved in step 1 and translates them into labels associated with each node in the tree. Although the analysis of all the authorizations should eventually produce a unique sign on each element and attribute of the document, in the process itself it is convenient to associate with each node more than one sign; one sign for each possible authorization type. For this reason, our tree labeling process starts by associating with each node n a vector $Label_n$ of eight elements, one for each type $t \in$ {LDH,RDH,L,R,LD,RD,LS,RS}, whose content reflects the authorizations specified on the node. Again, note that several authorizations, possibly of different sign, may exist for each of such types, either specified for the same or for different subjects. For instance, a (user) requester may belong to two different groups, and one of such group may be allowed access to a node while the other may be denied the same access. The presence of both authorizations is completely legitimate and it is not to be considered an inconsistency; it represents (on the given user) a conflict that must be resolved according to some conflict resolution policy. Different policies can be applied to this purpose [Damiani et al., 2000, Jajodia et al., 1997, Lunt, 1989]. For instance, a possible policy is the intuitive "most specific subject takes precedence" together with the "denials take precedence" according to which authorizations specified for a group (location pattern, resp.) are overridden by authorizations specified for its members (location subpatterns, resp.) and, if conflicts

remain unsolved (incomparable subjects) the denial prevails. To make our mechanism largely applicable and adaptable to different policies that can be considered, we evaluate authorizations independently of the conflict resolution policy for subjects. To this purpose, as a result of parsing authorizations, we associate with each vector component $Label_n[t]$ two lists of subjects: $Label_n[t].Denied$ and $Label_n[t].Allowed$, composed of all those subjects for which there is a negative, positive respectively, authorization of type t that applies to n. The sign ('+' or '−') to be then considered applicable to node n for each type t, stored in $Label_n[t].sign$, is then obtained by combining the two lists according to the conflict resolution policy. $Label_n[t].sign = \varepsilon$', in the case where no authorization of type t applies to n. While our mechanism implements the policy discussed above, the application of a different policy simply requires a different combination of the two lists.

Step 3: Label propagation. The third step of the algorithm consists in propagating the access permissions/denials associated with each node down to its descendants (elements and attributes), according to their types (see Section 4). Since propagation from a node may affect its indirect descendants in a recursive way, the natural way of enforcing the process is by considering the nodes according to a preorder visit of the tree, therefore labeling a node only after the labeling of its parent. Propagation makes use of operator $\oplus$. Propagation of labels from node p to a child attribute n is obtained, for each t, by assigning to $Label_n[t].sign$ the result of $Label_n[t].sign \oplus Label_p[t].sign$. Intuitively, $Label_n[t].sign$ remains unchanged if different from 'ε'; it takes the value of $Label_p[t].sign$, otherwise. Propagation of labels in case n is an element is enforced in an analogous way with the difference that signs of "local" labels (`LDH`, `L`, `LD`, and `LS`) are not propagated. This is obtained by combining $Label_n$ with a masked $Label_p$, (function **mask_local**), where the sign of `LDH`,`L`,`LD`,`LS` is set to 'ε'. Note that the algorithm enforces propagation of labels with reference to vectors and uses operator $\vec{\oplus}$, defined as component-wise application of $\oplus$ to the sign fields of the vectors. After the propagation is completed, the final sign associated with each node n, denoted $final_n$, can be determined as the result of operation $\oplus$ between the elements of the label vector considered in their priority order (from left to right). Intuitively, this is the same as taking the first non null (i.e., different from 'ε') value in the vector. It is easy to see that this process of propagating labels and determining the final signs correctly implements principles 1 and 5 discussed in Section 4.

Step 4: View computation. The final step of the access control algorithm computes the document's view to be returned to the requester. As a result of the labeling process, the value of $final_n$ for each node n contains the sign, if any, reflecting whether the requester can ('+') or cannot ('−') access the node. Remember that, even after propagation, the value of $final_n$ can still be 'ε', in the case where no authorizations have been specified nor can be derived for n. Value 'ε' can be interpreted either as a negation or as a permission, corresponding to the enforcement of the *closed* and the *open* policy, respectively [Jajodia et al., 1997]. Here we assume the closed policy. Accordingly, the requester is allowed to access all the elements and attributes whose label is positive. To preserve the structure of the document, the returned view will also include start and end tags of elements with a negative or undefined label, which have a descendant with a positive label. With respect to implementation, the view is obtained by pruning from the original document tree all the subtrees containing only nodes labeled negative or undefined. By exploiting the tree structure, the algorithm enforces such a pruning by visiting the tree in postorder and removing all leaf nodes n with $final_n \neq$ '+' (note that a node can be a leaf because all its descendants have

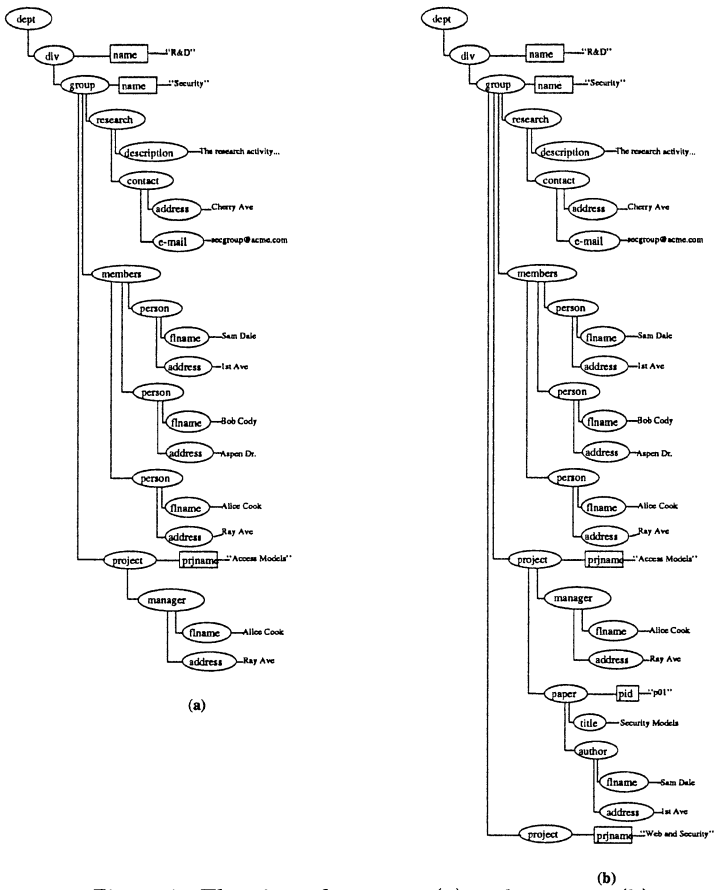


Figure 4 The view of user `Tom` (a) and user `Sam` (b)

already been deleted in the process). The computed pruned document, although guaranteed to be well-formed, may not be valid with respect to the original DTD (e.g., a required attribute is deleted in the pruning process). To avoid this problem, a *loosening* transformation is applied to the DTD. Loosening a DTD simply means to define as *optional* all the elements and attributes marked as *required* in the original DTD. From a security point of view, the DTD loosening prevents users from detecting whether information was hidden by the security enforcement or simply missing in the original document.

Example 5.1 Consider the set of authorizations illustrated in Section 4 and suppose that users `Tom` and `Sam` connected from hosts `130.100.50.8` and `130.89.56.8`, respectively, submit a request to read the document in Figure 1. We now assume that `Tom` belongs to group `NonMembers` (he does not belong to any other group), and `Sam` is a member of the `Security` group, which is a subgroup of `DeptMembers`. `Sam` does not belong to any other group (in particular, he does not belong to `Manager`). Figure 4 illustrates the views returned to `Tom` and `Sam` by the access control algorithm.

6. CONCLUSIONS

We have presented an access control system for specifying and enforcing protection requirements on Web documents. Our system not only takes into account the possible semistructured form of documents, but exploits it to provide flexibility and expressiveness in the access control model itself. Basing on the XML standard, our access control system results of easy integration with existing technology.

Acknowledgments

This work was supported in part by the European Community within the FASTER Project in the Fifth (EC) Framework Programme under contract IST-1999-11791 and by the Italian MURST within the INTERDATA and DATA-X projects.

References

[XPa, 1999] (1999). *XML Path Language (XPath) Version 1.0.* World Wide Web Consortium (W3C). http://www.w3.org/TR/PR-xpath19991008.

[Bray et al., 1998] Bray, T., Paoli, J., and Sperberg-McQueen, C. (1998). *Extensible Markup Language (XML) 1.0.* World Wide Web Consortium (W3C). http://www.w3.org/TR/REC-xml.

[Buneman, 1997] Buneman, P. (1997). Semistructured Data. In *1997 Symposium on Principles of Database Systems (PODS97)*, pages 117–121, Tucson, Arizona.

[Damiani et al., 2000] Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., and Samarati, P. (2000). Securing XML Documents. In *VII Conference on Extending Database Technology (EDBT2000)*, Konstanz, Germany.

[Goldman et al., 1999] Goldman, R., McHugh, J., and Widom, J. (1999). From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. In *Proc. of the 2nd International Workshop on the Web and Databases (WebDB '99)*, Philadelphia, Pennsylvania.

[Jajodia et al., 1997] Jajodia, S., Samarati, P., and Subrahmanian, V. (1997). A Logical Language for Expressing Authorizations. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 31–42, Oakland, CA.

[Lunt, 1989] Lunt, T. (1989). Access Control Policies for Database Systems. In Landwehr, C., editor, *Database Security, II: Status and Prospects*, pages 41–52. North-Holland, Amsterdam.

[Papakonstantinou et al., 1995] Papakonstantinou, Y., Garcia-Molina, H., and Widom, J. (1995). Object Exchange Across Heterogeneous Information Sources. In *ICDE*, pages 251–260, Taipei, Taiwan.

[Rescher, 1969] Rescher, N. (1969). *Many Valued Logics.* Mc Graw-Hill, New York.

[Samarati et al., 1996] Samarati, P., Bertino, E., and Jajodia, S. (1996). An Authorization Model for a Distributed Hypertext System. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):555–562.

[Siméon and Smaga, 1998] Siméon, J. and Smaga, K. (1998). Your Mediators Need Data Conversion! In *Proc. of the ACM SIGMOD'98 International Conference on Management of Data*, Seattle, Washington.