Feature

Controlling Access to XML Documents

Access control techniques for XML provide a simple way to protect confidential information at the same granularity level provided by XML schemata.

Ernesto Damiani
University of Milan

Sabrina De Capitani di Vimercati
University of Brescia

Stefano Paraboschi
Polytechnic of Milan

Pierangela Samarati
University of Milan

The widespread adoption of the eXtensible Markup Language has profoundly changed the nature of the information and user interaction styles available on the Web. Unlike HTML, XML can represent both document structure and content, offering increased control of information granularity through transformation and query languages. The original XML 1.0 specification used Document Type Definitions (DTDs) to describe document structure; recently, the World Wide Web Consortium (W3C) issued the XML Schema Recommendation, which lets users define XML vocabularies and complete typespaces. Starting from a document-oriented standard, XML is evolving toward a universal format for information interchange.

This evolution is fostering a new generation of applications that can synthesize and exchange XML information tailored to user needs. XML's new features, however, envision a complete paradigm shift from HTML that raises new security concerns in the WWW community. Initially, many practitioners assumed that XML documents would automatically benefit from the security standards already in place to deliver HTML pages via HTTP. Later, researchers began investigating XML-specific security measures that would address its richer data model and finer granularity control; work is under way to develop standard XML-based formats for the resources to be protected, their associated metadata, and the policies expressing authorizations (see the sidebar, "The XACML Standardization Effort"). For instance, the adoption of XML for medical records requires tailoring information from XML data sources to the different needs of physicians and patients, preserving confidentiality and avoiding unnecessary duplications. In the same scenario, access control policies themselves should be easy to process and interchange and should check for compliance with externally defined regulations.

In this article, we describe our approach to these problems and the design guidelines that led to our current implementation of an access control system for XML information. (See the sidebar, "Related Work in XML Access Control," as well.)

The Role of Encryption

Cryptography has given the Web a general-purpose infrastructure for secure communication. What is its potential role in providing fine-grained security to XML documents? Some commercial products are available (for example, AlphaWorks' XML Security Suite; alphaworks.ibm.com/tech/xmlsecuritysuite) providing fine-grained security features, such as element-wise encryption and digital signatures. DataChannel (www.datachannel.com) has proposed a coarser solution; their Server product links XML authentication to directory systems, supporting both Windows NT and Lightweight Directory Access Protocol 3 directories. However, encryption-based approaches unequally split security responsibilities between the connection protocol, the XML content, and the application processing the document; in parallel, the need for access control standardization for XML data is receiving growing recognition. Moreover, some encryption-based techniques leave encrypted private information in the hands of unauthorized users, a design choice that might well prove unwise over time. Recently, a W3C initiative began dealing with XML Encryption, focusing on how to make XML content discernible only to the intended recipients, and opaque to all others. XML Encryption focuses on how to encrypt XML documents at the granularity of elements. While there are many applications for such a specification, including the protection of payment and transaction information, the XML Encryption initiative is explicitly *not* aimed at controlling access to XML information.

XACML Standardization Effort

The research described in this article, together with that described by Michiharu Kudo and Satoshi Hada,[1] underlies a recent standardization effort on the use of XML-based languages to express and interchange access control policies.

This effort is organized under a new technical committee within the Organization for the Advancement of Structured Information Standards, a nonprofit, international consortium that creates interoperable industry specifications based on public standards such as XML and SGML (http://www.oasis.org). The OASIS eXtensible Access Control Markup Language (XACML) technical committee has been organized to define a standard core schema and corresponding namespace for the expression of authorization policies in XML against objects that are themselves identified in XML. The schema will be capable of representing the functionality of most policy representation mechanisms available at the time of adoption. It should also be extensible to address custom application requirements and other functions or features not yet included.

The TC charter includes such issues as fine-grained control, the requestor's nature, the protocol over which the request is made, content introspection, and the types of activities authorized. Initial TC members include Baltimore Technologies, CrossLogix, Hewlett-Packard, IBM, Jamcracker, Oblix, Reuters, Sun Microsystems, and webMethods. Other companies and individuals are encouraged to participate in the XACML work by joining OASIS.

Reference

1. M. Kudo and S. Hada, "XML Document Security Based on Provisional Authorization, *Proc. 7th ACM Conf. Computer and Communication Security*, ACM Press, New York, 2000, pp. 87–96.

XML Access Control: An Outline

Our approach exploits XML's own capabilities, using XML markup to describe access authorizations to XML elements. The hierarchical structure of XML documents lets you intuitively specify and authorization's definition: authorizations, when stated for an element, can propagate to the other elements or attributes included in it, unless a more specific authorization is stated for them. The main features of our fine-grained authorizations include

- *Authorization signs*: Authorizations can be positive, granting access, or negative, denying access, to an XML element or attribute. The possibility of specifying negative authorizations, while increasing our model's expressive power, introduces potential conflicts among authorizations. Our approach solves such conflicts by giving priority to authorizations specified on more specific subjects or objects, and denying the access (*denial takes precedence* policy) for unresolved conflicts.
- *Authorization levels*: You can add security markup to XML documents and XML Schemas, to provide document- and schema-level authorizations with the granularity of XML elements and attributes. Schema and document-level authorizations have complementary roles in increasing access control flexibility. Intuitively, as XML Schemas specify structure and content of entire document classes, schema-level security markup lets you quickly and effectively state authorizations that apply to XML elements regardless of the specific document under consideration (as with a `<CONFIDENTIAL>` tag used consistently in a set of documents). Schema-level authorizations can serve to implement corporate-wide access control policies on document classes. Document-level security markup lets you tailor security requirements for each document, as is required when documents complying with the same XML Schema contain information with different protection requirements (for example, a `<CURRENTPROJECTS>` tag in the resumes of researchers pursuing both classified and public projects).
- *Authorization strength*: Intuitively, document-level authorizations usually take precedence

Related Work in XML Access Control

Because HTML tagging is intended primarily to render Web pages, access control mechanisms currently available for Web sites tend to be coarse-grained. For instance, the Apache Web server (www.apache.org) lets you specify access control lists through a configuration file (access.conf) containing a list of users, hosts (IP addresses), or host–user pairs that are specifically allowed or forbidden connection to the server. Users are identified by user and group names and passwords, which are specified through Unix-style password files. By specifying a configuration file for each directory on the Web server's disk, you can define authorizations on a directory basis. The specification of authorizations at the level of single files (that is, Web pages) is possible but a little awkward, whereas it is not possible on portions of files. This limitation forces protection requirements to affect data organization at the file system level.

John Linn and Magnus Nyström propose a model for authorizations that reference portions of a file.[1] However, the model does not support semantic context similar to that provided by XML and so remains limited. The Enterprise Integrated Technologic Secure HTTP scheme (www.homeport.org/~adam/shttp.html) represents authorizations within HTML documents through security-related tags. Every document can include security (meta)tags describing its access authorizations. Again, because of HTML limitations, even this proposal cannot take information semantics into account.

Recently, several research groups proposed access control techniques expressed through XML-based languages or aimed at XML data.[2,3] We developed the approach here from work presented at the 2000 International Conference on Extending Database Technology (EDBT2000).[4]

A related line of research was pursued by The Tokyo IBM Research Labs by developing a processor that lets users to specify and enforce fine-grained authorization policies.[5] They have proposed XrML (www.xrml.org/), a commercial product designed to serve as an open architecture for digital content rights management.

Our model presents similarities with some access control models for object-oriented DBMSs; in particular, authorizations can propagate along the document structure and conflict resolution must consider authorization strength.[6] However, the object-oriented context differs significantly from XML, which was born as a textual representation format:

- In XML, the main form of relationship between nodes is in the form of node containment (which is similar but not identical to part-of composition in object-oriented models);
- XML does not offer inheritance, polymorphism, or typing of references.

You must consider all these aspects when designing an access control model for XML, making it inconvenient to apply an object-oriented access control model to XML information.

References

1. J. Linn and M. Nyström, "Attribute Certification: An Enabling Technology for Delegation and Role-Based Controls in Distributed Environments," *Proc. 4th ACM Workshop on Role-based Access Control,* ACM Press, New York, 1999, pp. 121–130.
4. E. Damiani et al., "Securing XML documents," *Proc. 2000 Int'l Conf. Extending Database Technology* (EDBT2000), Springer Verlag, Berlin, 2000, pp. 121–135.
2. E. Bertino, S. Castano, and E. Ferrari, "Securing XML Documents with Author-X," *IEEE Internet Computing*, Vol. 5, No. 3, May–June, 2001, pp. 21–31.
3. A. Gabillon and E. Bruno, "Regulating Access to XML Documents," *Proc. 15th Ann. IFIP WG 11.3 Working Conf. on Database and Application Security,* Kluwer, 2001, pp. 311–328.
5. M. Kudo and S. Hada, "XML Document Security Based on Provisional Authorization," *Proc. 7th ACM Conf. Computer and Communication Security* (CCS 2000), ACM Press, New York, 2000, pp 87–96.
6. S. Jajodia et al., "Flexible Support for Multiple Access Control Policies," *ACM Trans. Database Systems*, 2001; to appear.

over schema-level ones. When this behavior is not adequate, you can either define document-level authorizations as soft (giving them a lower priority than schema-level ones) or define schema-level authorizations as hard (giving them a higher priority). You would use soft document-level authorizations when the document owner agrees to accept the policy stated at the schema level, if existing. In turn, you would use hard schema-level authorizations when your organization wants to implement a system-wide security policy, which must not be reverted by authorizations at the specific document level.

- *Propagation policies*: Whatever its sign or level, an authorization can be either local (it applies only to the current element and its attributes) or recursive (it applies recursively to the current element and its subelements).

Authorizations must be specified for the different classes of requesters, as identified by user-group name and physical location. As you would obviously do not want to list all potential requesters individually, our model uses the well-known technique of defining a subject hierarchy of requesters based on their identifications. Authorizations stated for a given subject automatically apply to all subjects below it in the subject hierarchy. Every time you request access to XML data, the joint enforcement of the authorizations that apply to them at the schema and document level will produce a custom view on the data, including only the information that the particular requesters are entitled to see. This approach, while powerful enough to define sophisticated access to XML data, makes the design of a server-side Access Control Processor (ACP) for XML data sources rather straightforward.

The specification of the access control system we've just outlined requires a detailed definition of objects (the resources against which authorizations must be specified) and subjects (the system's users).

Identifying Authorization Objects via Path Expressions

Our model identifies the objects to which fine-grained access authorizations apply using XPath (XML Path Language; www.w3.org/TR/xpath) expressions, which return a set of nodes within a document. XPath is a W3C standard well known to potential users and supported by several tools that can be easily reused to produce a functioning system. A simple example of XPath expression is a sequence of element names separated by a slash; for instance, path expression `/catalog/category/merchant` denotes the nodes of the merchant element, which are children of category elements, which are children of catalog elements. Path expressions can terminate with an attribute name (prefixed with the special character `@`), add conditions in the navigation, and also use special functions. Overall, XPath is a powerful language and its capabilities match quite well the access control model's needs.

Dealing with Roles

The application scenario we describe in this article envisions an access control server dealing with locally defined users and groups. While such an approach might well suffice for a number of Intranet-based applications, Internet services often require access control decisions not to be identity-based. On the global Net, the way users can exercise an access right (which we characterize as a *role*) tends to be more important than a particular user's identity. For instance, browsing an on-line library catalog might require that a user subscribe to the library (user's identity can be relevant only for accounting purposes or not at all). We use roles to deal with access requests coming from previously unknown parties.

Our model readily extends to cover this eventuality by considering roles as an additional component characterizing authorization subjects. It lets you model subjects requesting access as a 4-tuple ⟨user-id, IP-address, sym-address, role-id⟩, where role-id is a set of roles activated by attaching appropriate accreditation (or attribute) certificates with a request.[1,2] An authorization's subject can be either a user, group, or role, possibly restricted with respect to a request's originating location.

References

1. J. Linn and M. Nyström, "Attribute Certification: An Enabling Technology for Delegation and Role-Based Controls in Distributed Environments," *Proc. 4th ACM Workshop on Role-based Access Control,* ACM Press, New York, 1999, pp. 121–130.
2. J.S. Park, R. Sandhu, and G.-J. Ahn, "Rolle-Based Access Control on the Web," *ACM Trans Information and System Security,* Vol. 4, No. 1, Feb. 2001, pp. 37–71.

Identifying Authorization Subjects

Usually, most approaches identify authorization subjects by their identity or the location from which their requests originate. In turn, locations can be expressed through IP addresses (`150.100.30.8`, for example) or symbolic names (such as `tweety.admin.com`). Our model combines all these features. In it, we characterize subjects requesting access by a triple ⟨`user-id, IP-address, sym-address`⟩, where `user-id` is the login name the user used in connecting to the server, `IP-address` is the client machine's address, and `sym-address` is the machine's name.

Our model lets you consider remote identities trusted by the server (using a Certification Authority or any other secure infrastructure) as well. So that authorizations can apply to sets of users or machines, our model also supports groups and location patterns. *Groups* are sets of users defined at the server; they need not be disjoint and can be nested. On the other hand, a *location pattern* is an expression identifying a set of physical locations, referencing either their symbolic names or IP addresses. Our model uses the wild card character * to specify *patterns*. For instance, `151.100.*.*` denotes all the machines belonging to network 151.100. Similarly, `*.it` denotes all the machines in the Italy domain. (See the sidebar, "Dealing with Roles.")

For each XML data source, users and groups linked by their membership relationship, IP addresses with patterns, and symbolic names with patterns form three distinct hierarchies. As the same user can belong to more than one group, the user-group hierarchy is a direct acyclic graph (DAG), while IP addresses' and, as you would expect, symbolic names' hierarchies are trees. Figure 1 shows an example of such hierarchies. You can define a general authorization subject hierarchy ASH by combining user and group, symbolic names, and IP address hierarchies as follows: a subject s_j is dominated by another subject s_i (that is, $s_j \leq s_i$) if each of s_j's components (namely user-group, IP address, and symbolic name) are dominated by the corresponding component of s_i.

While our model conceptually identifies authorization subjects by triples of the general hierarchy, it can detect relationships between address (and symbolic names) patterns in a straightforward manner; therefore, only the usual user-group hierarchy need be explicitly defined and stored at the server. It can specify access authorizations for any of ASH's elements, propagating the authorizations to all subjects that lie below it in the hierarchy. More formally, authorizations specified for subject $s_j \in$ ASH are applicable to all subjects s_i such that $s_i \leq s_j$.

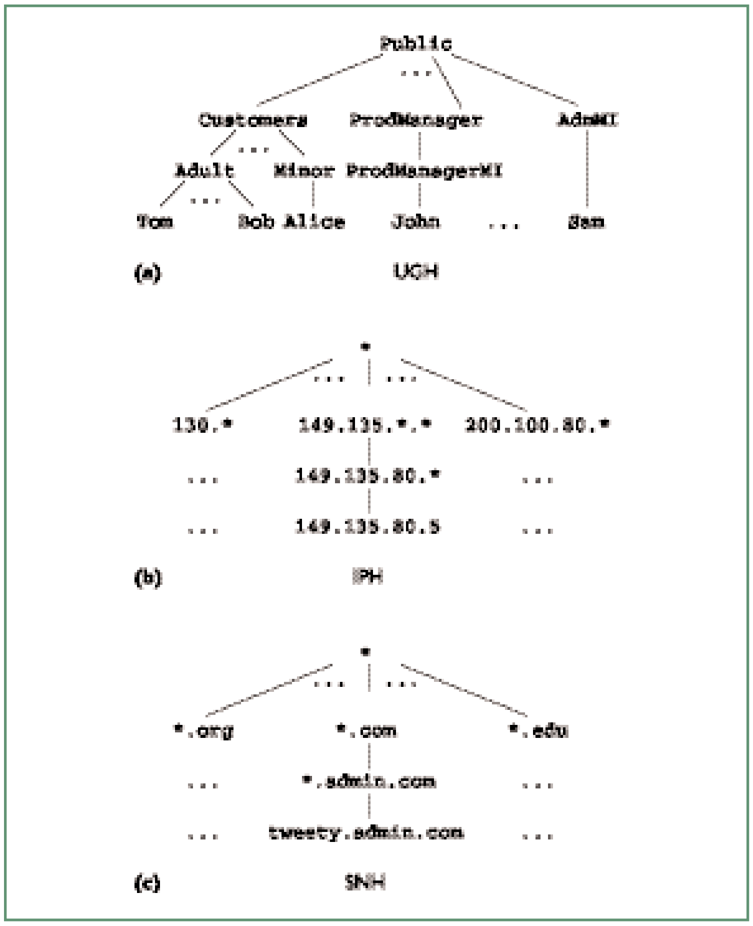


Figure 1. Sample hierarchies: (a) user-group, (b) IP, and (c) symbolic name.

A Closer Look at Fine-Grained Authorizations

To illustrate our model's expressiveness, consider the following example. An electronic company, called OnlineMall, wants to create an online catalog so potential customers can search and browse the catalog and purchase items. The catalog is encoded in XML and its structure is defined by the XML Schema in Figure 2a.

OnlineMall collects information about products and services of several merchants such as MyItems. For accounting purposes, besides catalog information, each merchant maintains profiles of its customers. Customer profiles, described by the XML Schema in Figure 2B, include such personal customer information as name, address, date of birth, and sex, and general information such as age, preferences, and hobbies.

Here, we present some examples of protection requirements for the XML document depicted in Figure 3, complying with the XML Schema in Figure 2b, which is stored at the Web site of merchant MyItems. The letters between square brackets in the list identify the authorizations in Figure 4. Note that the horizontal line between authorizations [e] and [f] separates schema-level authorizations ([a] through [e]) from document-level authorizations ([f] through [k]).

OnlineMall's Policy. Specified at the schema level—applicable to all merchants:

[a] Catalog information is public.

[b] Personal information about merchants cannot be accessed by customers. (This authorization forbids access only to the textual content of `pinfo`'s children, so that customers can still see the element names.)

[c] Information about tobacco and wine products can be accessed only by customers who are not minors.

[d-e] Information about customers cannot be accessed without the customer's consent.

MyItems' Policy. Specified at the document level by the MyItems merchant to complement or override the OnlineMall's policy:

[f] Information about customers cannot be accessed, unless otherwise stated at the schema level.

[g] Information about `name` and `address` of all customers can be accessed by members of the `AdmMI` group connected from network `130.*`.

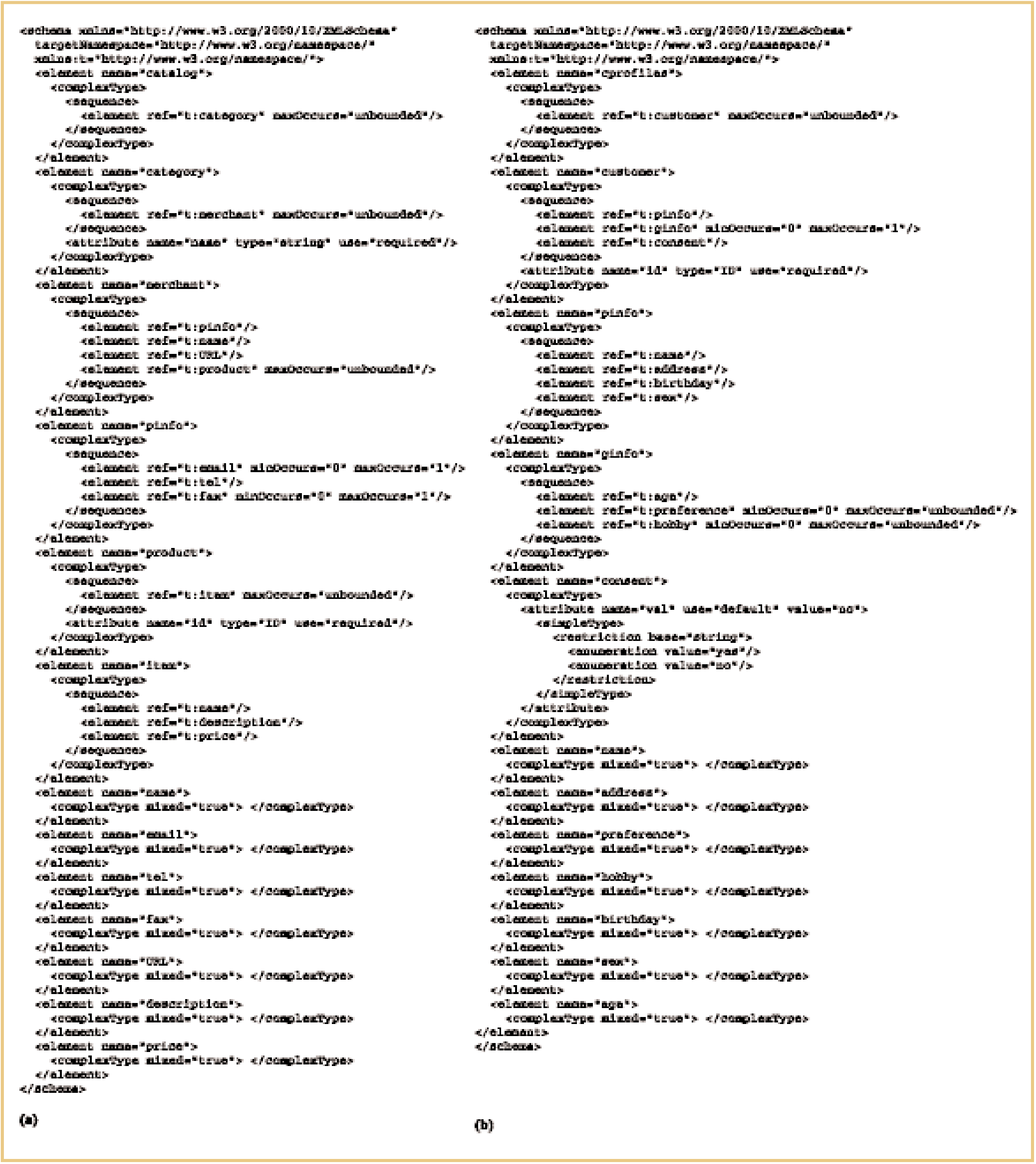
[h] Customer identifiers can be accessed by the administrative staff.

[i] General information about customers can be accessed by members of `ProdManager` group.

[j] Information about birth date and sex of all customers can be accessed by members of `ProdManagerMI` group when connected from hosts with domain `*.it`.

Computing the Requester's View on XML Documents

In our model, you compute each subject's view on each XML document by combining local and propagated authorizations at schema and document levels. This computation's first stage is a simple labeling procedure. To explain the procedure's operation, we employ a widely used graphical representation of XML documents.[1] It represents XML documents as labeled trees containing a node for each attribute and element. Elements are represented as circles and attributes as squares. There is an arc between an element and an element or attribute belonging to it.



```
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
  targetNamespace="http://www.w3.org/namespace/"
  xmlns:t="http://www.w3.org/namespace/">
  <element name="catalog">
    <complexType>
      <sequence>
        <element ref="t:category" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
  </element>
  <element name="category">
    <complexType>
      <sequence>
        <element ref="t:merchant" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="name" type="string" use="required"/>
    </complexType>
  </element>
  <element name="merchant">
    <complexType>
      <sequence>
        <element ref="t:pinfo"/>
        <element ref="t:name"/>
        <element ref="t:URL"/>
        <element ref="t:product" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
  </element>
  <element name="pinfo">
    <complexType>
      <sequence>
        <element ref="t:email" minOccurs="0" maxOccurs="1"/>
        <element ref="t:tel"/>
        <element ref="t:fax" minOccurs="0" maxOccurs="1"/>
      </sequence>
    </complexType>
  </element>
  <element name="product">
    <complexType>
      <sequence>
        <element ref="t:item" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="id" type="ID" use="required"/>
    </complexType>
  </element>
  <element name="item">
    <complexType>
      <sequence>
        <element ref="t:name"/>
        <element ref="t:description"/>
        <element ref="t:price"/>
      </sequence>
    </complexType>
  </element>
  <element name="name">
    <complexType mixed="true"> </complexType>
  </element>
  <element name="email">
    <complexType mixed="true"> </complexType>
  </element>
  <element name="tel">
    <complexType mixed="true"> </complexType>
  </element>
  <element name="fax">
    <complexType mixed="true"> </complexType>
  </element>
  <element name="URL">
    <complexType mixed="true"> </complexType>
  </element>
  <element name="description">
    <complexType mixed="true"> </complexType>
  </element>
  <element name="price">
    <complexType mixed="true"> </complexType>
  </element>
</schema>

(a)
```

```
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
  targetNamespace="http://www.w3.org/namespace/"
  xmlns:t="http://www.w3.org/namespace/">
  <element name="cprofiles">
    <complexType>
      <sequence>
        <element ref="t:customer" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
  </element>
  <element name="customer">
    <complexType>
      <sequence>
        <element ref="t:pinfo"/>
        <element ref="t:ginfo" minOccurs="0" maxOccurs="1"/>
        <element ref="t:consent"/>
      </sequence>
      <attribute name="id" type="ID" use="required"/>
    </complexType>
  </element>
  <element name="pinfo">
    <complexType>
      <sequence>
        <element ref="t:name"/>
        <element ref="t:address"/>
        <element ref="t:birthday"/>
        <element ref="t:sex"/>
      </sequence>
    </complexType>
  </element>
  <element name="ginfo">
    <complexType>
      <sequence>
        <element ref="t:age"/>
        <element ref="t:preference" minOccurs="0" maxOccurs="unbounded"/>
        <element ref="t:hobby" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
  </element>
  <element name="consent">
    <complexType>
      <attribute name="val" use="default" value="no">
        <simpleType>
          <restriction base="string">
            <enumeration value="yes"/>
            <enumeration value="no"/>
          </restriction>
        </simpleType>
      </attribute>
    </complexType>
  </element>
  <element name="name">
    <complexType mixed="true"> </complexType>
  </element>
  <element name="address">
    <complexType mixed="true"> </complexType>
  </element>
  <element name="preference">
    <complexType mixed="true"> </complexType>
  </element>
  <element name="hobby">
    <complexType mixed="true"> </complexType>
  </element>
  <element name="birthday">
    <complexType mixed="true"> </complexType>
  </element>
  <element name="sex">
    <complexType mixed="true"> </complexType>
  </element>
  <element name="age">
    <complexType mixed="true"> </complexType>
</element>
</schema>

(b)
```

Figure 2. Two examples of XML Schema: (a) catalog structure, (b) customer profiles.

Figure 3 illustrates the tree representation of an XML document complying with the schema in Figure 2b. When you request a document, the analysis of all the authorizations holding for you produces an access decision (*yes* or *no*, depicted as + or –) on each node to which some authorization applies. To obtain this outcome, we associate with each note a list of signs, corresponding to authorizations of different types. Namely, our tree-labeling process associates to each element or attribute node *n* an 8-tuple $\langle \mathtt{LXH}_n; \mathtt{RXH}_n; \mathtt{L}_n; \mathtt{R}_n; \mathtt{LX}_n; \mathtt{RX}_n; \mathtt{LS}_n; \mathtt{RS}_n \rangle$, whose content reflects the authorizations

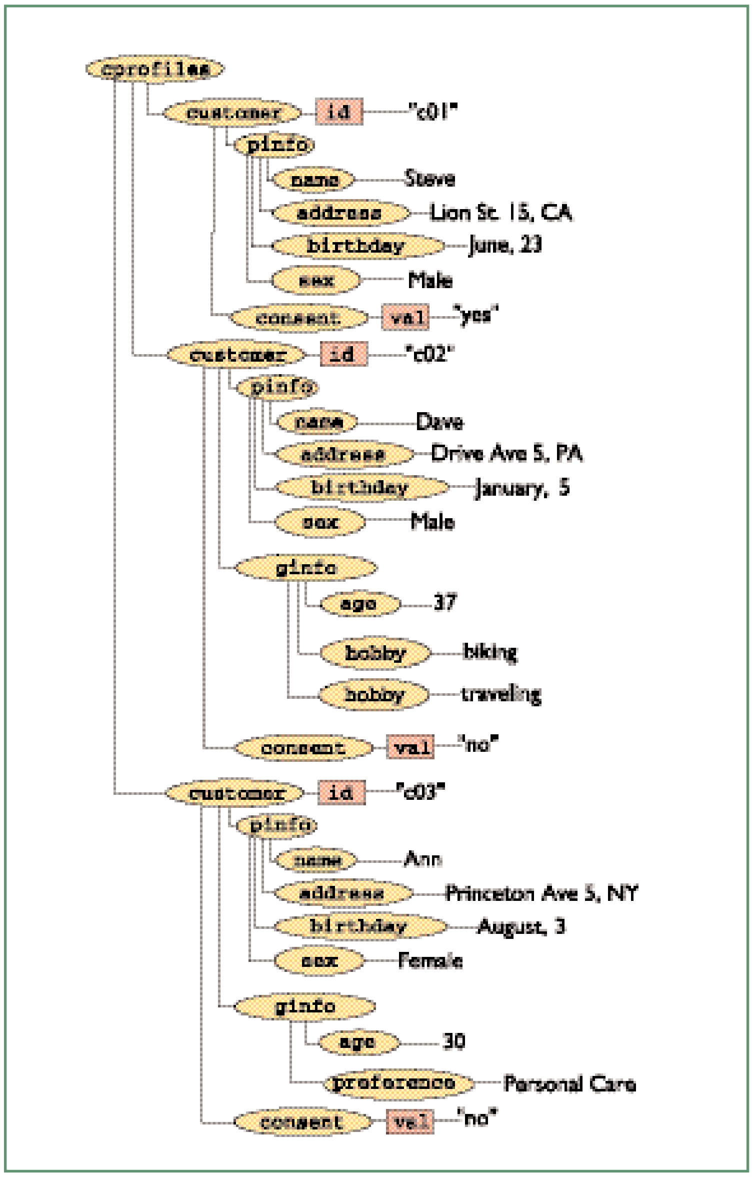


Figure 3. A valid XML document conforming to the XML schema in Figure 2b.

specified on the node. In other words, the 8-tuple elements contain the signs of the authorizations of types Local for the XML Schema Hard; Recursive for the XML Schema Hard; Local; Recursive; Local for the XML Schema; Recursive for the XML Schema Local Soft; and Recursive Soft; holding for each node *n*. The eight types arise from the combination of three properties: propagation (local or recursive), level (document or schema), and strength (normal and soft or hard).

An element's order in the 8-tuple reflects its priority, from the highest to the lowest, in determining the access decision. Each element in the tuple can assume one of three values: + for permission, – for denial, and ε for no authorization. Local authorizations holding for each node propagate to its attributes, while propagating authorizations also propagate to its subelements. You can override authorizations according to a *most specific object takes precedence* principle, which guarantees that authorizations on a node take precedence over those on its ancestors. This principle operates together with the *denial takes precedence* policy we discussed earlier.

Thus, you can obtain a document's labeling by starting from its root and, proceeding downwards with a preorder visit, updating the 8-tuple of a node *n* depending on its values and the values of the 8-tuple of node *p* parent of *n* in the tree. At the visit's end, for each node *n* of the document tree, the authorization valid on *n* will be the not null one with the highest priority. The decision value is set to the null value ε when no authorizations have been specified nor can be derived for *n*. You can interpret value ε as either a negation or a permission, corresponding to the enforcement of the closed or the open policy.[2] We act conservatively, choosing the closed policy.

Document Transformation

After the labeling process, the requester can access all the elements and attributes whose label is positive. For elements with a negative or undefined label that have a descendant with a positive label, start and end tags will also be included in the document portion visible to the requester. You can obtain the document view by pruning from the original document tree all the subtrees containing only nodes labeled negative or undefined. You perform this pruning using a postorder visit on the document removing any leaf labeled – or ε.

The pruned document might not conform with the original schema. This will happen, for instance, when required attributes are deleted because the requester is not entitled to receive them. Typically, this is not a problem and it is sufficient for the XML document to be well-formed. If the document must be validated at the client side, you can apply a loosening transformation to the schema, defining as optional all the elements and attributes marked as required in the original schema. Schema loosening also prevents users from detecting whether information was hidden by the security enforcement or simply missing in the original document.

A Sample Transformation

We now provide a step-by-step example based on the XML document in Figure 3, using the set `Auth` of authorizations depicted in Figure 4 and the

```
[a] <<Public,*,*>, /catalog, read, +, RX>
[b] <<Customers,*,*>, /catalog//pinfo//text(), read, -, RXH>
[c] <<Minors,*,*>, //product[.//description[contains(text(),'tobacco')
         or contains(text(),'wine')]], read, -, R>
[d] <<Public,*,*>, /cprofiles, read, -, RX>
[e] <<Public,*,*>, /cprofiles/customer[./consent[@val='yes']], read, +, RX>
[f] <<Public,*,*>, /cprofiles/, read, -, RS>
[g] <<AdmMI,130.*,*>, /cprofiles//info/node()[position()=1
         or position()=2], read, +, L>
[h] <<AdMI,*,*>, /cprofiles/customer/@id, read, +, L>
[i] <<ProdManager,*,*>, /cprofiles//ginfo, read, +, L>
[j[ <<ProdManagerMI,*,*.it>, /cprofiles//pinfo/node()[position()=3
         or position()=4], read, +, L>
```

Figure 4. Examples of authorizations.

user-group hierarchy shown in Figure 1.

Consider two requests to read the document, the first submitted by user Sam connected from host 130.89.56.8 with symbolic name nf3lab.staff.it and the second from user Trent connected from host 130.100.50.5 with symbolic name u20.staff.it. Trent is a product manager of the FurnitureSupplier merchant, a member of company OnlineMall. According to the authorizations stated by the company and by the MyItems, because Sam is a member of the AdmMI group, he can only access all information about customers who give a positive consent, and names and addresses of all other customers. But, because Trent is a member of OnlineMall's ProdManager group, he can access general information about customers but not personal information.

Figure 5 shows the resulting view of Sam and Trent. Sam view is restricted to administrative information, while Trent view is restricted to general marketing information. Neither Sam nor Trent have a full view on the document.

Designing and Implementing an Access Control Processor for XML

We implemented a prototype of the system to demonstrate how our access control technique can be smoothly integrated with existing XML-based solutions.

As one of our first decisions in implementing the prototype, we decided to use the Java platform. This is a natural choice in the current XML context, where Java-based solutions are quite common. We first focused our prototype design efforts on demonstrating the access control model's internal mechanisms. Then, we refined it using a number of different server-side solutions: simple CGI architectures, Java servlets, and JSP pages. We also integrated the prototype with an XSL processing tool (the Cocoon environment produced by the Apache Software Foundation) and verified the use of an SSL implementation as a transport layer, a solution that would probably be used in secure environments. All these experiments demonstrated the model's applicability and flexibility.

Incidentally, we could have implemented the prototype with XSLT, defining a set of templates that, considering the requesting user and existing authorizations, would transform the XML document to remove the protected parts. The XSLT solution does not require a complex programming environment nor a Java Virtual Machine, whereas XSLT engines are already available in many XML processing environments, without requiring a complex programming environment nor a Java Virtual Machine. However, the XSLT language is designed to describe local XML transformations, so it is quite cumbersome for use in realizing an implementation of our access control model, which requires propagation of authorizations and conflict resolution among positive and negative authorizations. XSLT would be an interesting option for implementing an access control model simpler than the one we describe here.

Use of the System

Our system manages documents internally by representing them as object trees, according to the Document Object Model specification (www.w3org/TR/REC-DOM-Level-1). We chose DOM as the internal data representation format because it provides an object-oriented API for HTML and XML documents.

Our prototype is based on the definition of a set of Java classes describing users, groups, authorizations, and security labels. We do not detail the design and organization of classes, but in this context only provide an example of use of our prototype. The fragment of Java program in Figure 6 uses six parameters: three String variables representing the user name, IP address and symbolic address, and three File parameters representing the user–group hierarchy, the authorizations to apply, and the document requested by the user.

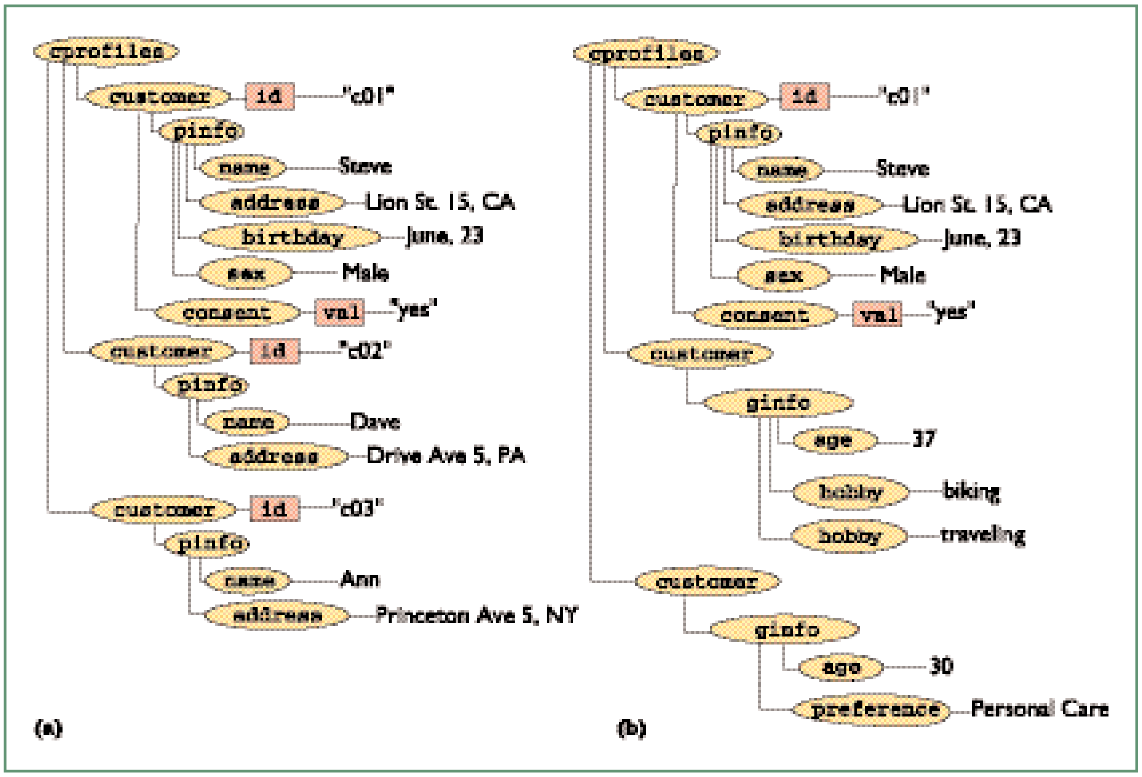


Figure 5. The views of user Sam (a) and user Trent (b) on the document in Figure 3b.

Statement 1 initializes variable `ugRep`, instance of class `UserGroupRepository`, which represents in the internal memory format of the prototype the user-group hierarchy described in the XML file `usersAndGroupsFile`. Variable user is created in statement 2, initialized to the object in `ugRep` identified by the name appearing in parameter `userName`. In statement 3, an instance `authSubj` of class `AuthorizationSubject` is created, using as parameters `symbName`, `ipAddr`, and the user variable created in the previous statement. The three parameters fully characterize the subject with respect to the identification of the authorizations that should be applied.

Statements 4 to 6 build an instance of `DocInfo`, a support class that binds the document with the authorization files applicable to it; the XML document the user wants to access is in file `documentFile` and the authorizations are in XML file `authorizationsFile`. Statement 7 creates the instance factory of class `SecureDocumentFactory`; this class is responsible of creating the pruned document, receiving as parameter the `UserGroupRepository` on which authorization subjects should be evaluated; statement 8 associates variable `factory` with the document and its authorizations, as represented in variable `docInfo`. In statement 9, the ACP is finally invoked, by method `getDocumentForUser`, using as parameter the `authSubj` object; the method returns the pruned document, which is assigned to variable `prunedDocument`.

The ACP follows three steps when it is invoked by method `getDocumentForUser`.

- *Parsing.* The parsing step consists in the syntax check of the requested document with respect to the associated XML Schema and its compilation to obtain an *object-oriented document graph* according to the DOM format. In the prototype this task is realized by Apache Software Foundation's Xalan tool.
- *Tree labeling.* The labeling step involves the propagation of the labeling of the DOM tree

URLs

The XML Access Control Prototype—http://sansone.crema.unimi.it/~xml-sec
XML—http://www.w3.org/TR/REC-xml
XML Schema—http://www.w3.org/TR/xmlschema-2
XPath—http://www.w3.org/TR/xpath
XQuery—http://www.w3.org/TR/xquery
XSLT—http://www.w3.org/TR/xslt
DOM—http://www.w3.org/TR/REC-DOM-Level-1

```
(1) UserGroupRepository ugRep = UserGroupRepository.getInstance(usersAndGroupsFile);
(2) UserGroup user = ugRep.userGroupByName(userName);
(3) AuthorizationSubject authSubj = new AuthorizationSubject(symbName,ipAddr,user);
(4) DocInfo docInfo = new DocInfo();
(5) docInfo.setXmlFile(documentFile);
(6) docInfo.addAuthDocuments(authorizationsFile);
(7) SecureDocumentFactory factory = SecureDocumentFactory.getInstance(ugRep);
(8) factory.setTargetDocument(docInfo);
(9) Document prunedDocument = factory.getDocumentForUser(authSubj);
```

Figure 6. Fragment of Java program using the prototype implementation.

according to the authorizations associated with the document. If the current user satisfies the pattern represented by the authorization's subject, the authorization is added to the nodes described by the authorization's object.

- *Transformation.* The transformation phase is the pruning of the DOM tree, described earlier. A visit on the document tree first evaluates in preorder the label of the current node, considering all authorizations defined on the node and recursive authorizations coming from the parent node. If the node has a negative label and it has an empty set of children, the node is removed from the document, pruning the DOM tree. The resulting DOM structure then returns as result.

Evolving the Prototype

Despite the access model's capabilities, the fine granularity on which accesses can be granted, and the use of nontrivial algorithms, the major source of complexity lies in the interpretation of the XPath expressions that identify the nodes. If all authorization objects are represented by an XPointer, the system characteristically has a linear complexity in the size of the document and the authorizations. In other words, our model does not introduce any fundamental increase in complexity. Moreover, several optimization techniques will speed up access control, including low-level refinements (for example, the use of a compiled language instead of Java), as well as a few high-level strategies.

A first strategy consists in the prelabeling of documents, to associate authorizations with the nodes of XML documents in anticipation of user requests. A further strategy involves the definition of auxiliary structures that permit to reduce the labeling procedure's granularity (for example, if an authorization defined on a node *n* is not overridden by any other authorization on its subnodes, the labeling procedure can deal with the subtree whose root is *n* as a unity).

Yue Wang and Kian-Lee Tan describe an ACP built starting from our prototype, which tries to improve system performance by avoiding the DOM construction and relying on a relational storage to compute access only to the fraction of data that the user is authorized to see.[3] This is an interesting solution, which is applicable in certain contexts (in particular, when authorizations severely restrict the portion of XML information that a user can access) and which can exploit the results of research on relational storage of XML information.

Conclusions

One of the evident features of our proposed system is its richness, which lets you define sophisticated security requirements, but also might require the authorization designer to carefully consider each authorization's implications. Also, the system reads authorizations in an XML format, which the designer might find difficult to manage if the XML representation must be created with a text editor or an unspecialized XML tool. To solve both problems, we implemented a tool in Java offering a graphical interface to the ACP, supporting the definition of user–group hierarchies and authorizations and their rapid interactive evaluation on the XML repository.

References

1. J. Siméon and K. Smaga, "Your Mediators Need Data Conversion," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, ACM Press, New York, 1998, pp. 177–188.
2. S. Jajodia et al. "Flexible Support for Multiple Access Control Policies," *ACM Trans. Database Systems*, 2001; to appear.
3. Y. Wang and K.-L. Tan, "A Scalable XML Access Control System," *Poster Proc. 10th Int'l World Wide Web Conf.*, Elsevier, Dordrecht, The Netherlands, 2001, pp. 150–151.

Ernesto Damiani is an associate professor at the Department of Information Technology of the University of Milan. His research interests include distributed and object-oriented systems, semistructured information processing and soft computing. He holds a Laurea in electrical engineering from the University of Pavia and a PhD in computer science from the University of Milan. He is the vice-chair of ACM SIGAPP and the general chair of the International Conference on Knowledge-Based Engineering Systems.

Sabrina De Capitani di Vimercati is an assistant professor at the Department of Electronics for Automation at the University of Brescia. She received her Laurea and PhD in computer science from the University of Milan. Her research interests are in information security, databases, and information systems. She has been an international fellow in the Computer Science Laboratory at SRI. She is co-recipient of the ACM-PODS'99 Best Newcomer Paper Award. Her Web page is http://www.ing.unibs.it/~decapita.

Stefano Paraboschi is an associate professor at the Department of Electronics and Information at the Milan Politechnic. He received the Laurea in electrical engineering and a PhD in informatics from Milan Politechnic. His main research interests are in databases, with a focus on active databases, data warehouses, and the construction of data-intensive Web sites. He is the coauthor of *Database Systems: Concepts, Languages and Architectures* (McGraw-Hill 1999).

Pierangela Samarati is a professor at the Department of Information Technology at the University of Milan. Her main research interests are in data and application security, information system security, access control policies, models and systems, and information protection in general. She has been a computer scientist in the Computer Science Laboratory at SRI. She is co-author of *Database Security* (Addison-Wesley, 1995) and is co-recipient of the ACM-PODS'99 Best Newcomer Paper Award. Her Web page is http://seclab.crema.unimi.it/~samarati.

Readers may contact the authors at damiani@dti.unimi.it, samarati@dti.unimi.it, decapita@ing.unibs.it, and parabosc@elet.polimi.it.

Put half-page fill here