Adaptive XML Access Control Based on Query Nesting, Modification and Simplification

Stefan Böttcher , Rita Steinmetz
University of Paderborn (Germany)
Computer Science
Fürstenallee 11
D-33102 Paderborn
stb@uni-paderborn.de , rst@uni-paderborn.de

Abstract: Access control is an important aspect in guaranteeing data privacy within XML data sources which are accessed by users with different access rights. The goal of predicative access control for XML data sources is to use XPath expressions to describe that fragment of a given document or database that can be accessed by queries of a certain user. Our approach to access control hides the XML data source within an access control module which implements a combination of two secure query execution plans. The first query execution plan works on copied secure fragments, whereas the second query execution plan transforms a given query into another query that respects all the access rights. For each query the appropriate secure execution plan is determined depending on the query and the user's access rights.

1. Introduction

XML is a widely accepted standard for the storage and representation of semi-structured data that are accessed by multiple parties with different access rights. Whenever some data has to be protected from the access of a part of the users, data distribution and access control become an issue. In general, two main directions to solve this problem can be distinguished. One direction is to keep multiple partially redundant copies of XML data, where each copy is tailored to the access rights of a special user group or user. This approach allows each user group or user to read its own copy that contains only data which the user has access to. Although this approach avoids access violations, it has the well known disadvantages of redundant data like the requirement to write each change of data to multiple copies and the risk of inconsistent updates.

In contrast, the other direction involves that all users or all user applications have to access the same XML data source, even if they have different access rights. Whenever users with different access rights must read (parts of) the same XML data source, access control is a key issue. Access control has to guarantee that no user application reads data that are not contained within the user's access rights.

Previous contributions to the field of access control for XML data sources cover different aspects that range from policies, to user groups, to document location, to access control on fragments of XML documents [13,2,6,1]. Further contributions focus on access control to encrypted XML data security views [15] and on query rewriting to hide partial information from insiders [11]. In comparison, our work focuses on access control for fragments of XML and follows approaches like [7,8] to determine by access rights which fragments of an XML document may be read by which users or user groups.

There are different possibilities how to define access rights, how to detect access violations and how to react on an access violation. While many approaches define access rights on the physical level [18,7], i.e. on the level of documents or on the level of nodes within an XML document, we follow [4] to define access rights on the basis of XPath expressions [20] for the following reasons. Access rights defined on the level of documents are too coarse-grained for many applications, i.e. for applications that share large documents between different groups of users. On the other hand, the declaration of access rights on the level of individual XML document nodes is often too fine-grained in the sense that access right information has to be added to too many individual nodes. The use of access rights defined for sub-trees of an XML document with the option to override the access rights [21] is a compromise, as it allows both a compact description of an access right for a sub-tree, and to define a different access right for individual nodes or sub-trees. However the definition of access rights has to obey the hierarchy given by an actual XML document, i.e. when we want to have the same access rights for a certain subset of nodes that are spread over the XML document, the sub-tree oriented approach to access rights has to define the same access rights for multiple sub-trees. In comparison, we follow the predicative approach of [3,4] to use an XPath expression to describe the subset of nodes for which an access right is granted for the following reason. This is not only more general than the sub-tree approach as each sub-tree, for which an access right is granted, can be represented by an XPath expression selecting exactly this sub-tree. It is also in general more compact to use XPath expressions for the description of access rights, as XPath expressions can select nodes based on content even if they occur widely spread over an XML document.

There are different possibilities how to detect that an operation violates an access right defined by an XPath expression and how to react on such an access violation. Proof-based techniques (e.g.[5,14,16,19]) prove that a query selects a subset of the nodes covered by an access right for arbitrary XML documents that are valid according to a given DTD. As proofs may become very complex for complicated queries, we prefer an approach that is based on queries alone. Inspired by an idea of [17] that preserves database consistency by query modification, we modify the query in such a way that it can not violate the access rights.

When a query of a user application accesses or attempts to access an existing fragment of a given XML document for which the user does not have an access right, we call this an *access violation*. The treatment of access violations has to avoid the following problem, called *information uncovering problem*. It is *not* acceptable that the access control engine signals an access violation to an application, whenever an access violation occurs for the following reason. The access violation signals received as reaction to certain access violations can be used to find out details of the data which should have been protected by the access control module. This is even the case, when only an access violation is signaled but further access to the fragment is prohibited. Therefore, in order to solve the information uncovering problem, it is either possible to signal an access violation independently of the concrete nodes stored in a given XML document, i.e.

based on the formula of the access rights and the formula of the query alone, or it is possible to restrict queries to only that fragment of a given XML fragment for which the access rights exists. The first solution uses a proof to check that a query can not violate an access right and signals an access violation when the proof fails [3,4]. In order to circumvent the complexity of these proofs, we follow the second approach, i.e. we present a solution which restricts the query to that fragment to which an access right exists.

The remainder of the paper is organized as follows. Section 2 presents basic assumptions and characterizes the problem. Section 3 describes the role and the location of the access control module within our overall system architecture. Section 4 describes access control based on copies, whereas Sections 5 and 6 describe our optimized approach to access control based on query modification and simplification of modified queries. Section 6 describes the adaptive decisions made by the access control module, and Section 7 outlines the summary and conclusions.

2. Basic assumptions and problem definition

We assume that a user's access rights on an XML document D are described by an XPath expression A, such that A(D) selects all the nodes of an XML document D for which access is granted to the user with access rights A.

For example, consider the following (extremely simplified) fragment of an XML document

```
<E1>
<E2 t="1"> … </E2>
<E2 t="2"> … </E2>              <!-- this line is not included in the access rights -->
<E2 t="3"> … </E2>
...
 </E1>
```

The access rights could be expressed by the XPath expression

A = /E1 | (/E1/E2 [not t="2"] #)

where "|" is the union operator and "#" is a shortcut for the selection of a sub-tree, i.e. whenever an XPath expression X selects the nodes {n1,…,nk} of a document D, then X# selects the k sub-trees of D with the root nodes n1,…,nk.

Furthermore, consider an example user query

Q1 = / E1 / E2 [2] .

As access to the element E2 with an attribute value of "2" for the attribute t is not granted, the query Q1 must assume that this element does not exist. Instead, the query Q1 must be answered only based on the nodes for which the access rights A exists, i.e. Q1 must return the same answer as a query

Q1A = / E1 / (E2[not t="2"])   [2]

i.e. Q1 shall return the node E2 with the attribute value of "3" for the attribute t .

As a second example query consider a query

Q2 = /E1[./E2/@t="2" and ./E2/@t="3"].

As access to the element E2 with an attribute value of "2" for the attribute t is not granted, also the query Q2 has to assume that this element does not exist.

Note that it is *wrong* to query for the intersection

$Q2 \cap A$ = /E1[./E2/@t="2" and ./E2/@t="3"] ,

i.e. for the elements that are selected by both the query Q2 and the access rights A, for the following reason. As the intersection query Q2∩A returns the element E1, it thereby uncovers information about the existence of an element E2 with the attribute value of "2" for the attribute t. In other words, by returning the answer of the intersection query Q2∩A, we uncover information stored in nodes for which the access rights (A) is not granted. Therefore, in general, the computation of the intersection of an access right and a query does not solve the information uncovering problem.

More generally stated, the access control problem is to answer a query Q in such a way that no information is uncovered, for which an access right A does not exist. In other words, we have to transform an incoming query Q into a query processing algorithm P(A,Q) that accesses only the allowed parts A(D) of the document. The result of P(A,Q) applied to any XML document D, i.e. P(A,Q)(D) must be the same as Q applied to A(D), i.e. P(A,Q)(D)=Q(A(D)). Given the query Q and the access rights A, the problem is to compute a query execution plan P(A,Q) such that P(A,Q)(D)=Q(A(D)) for every document D.

Furthermore, such a query execution plan has to be embedded into an architecture that guarantees that access control can not be circumvented. Finally, the architecture has to prevent that information in the XML database is uncovered by inspecting the communication of other users with the access control module.

3. The access control module within the overall system architecture

Within our overall system architecture (shown in Figure 1 below) an XML database is hidden by an access control module. In other words, user applications do not access the XML database directly. Instead, every user query and every query of a user's application program is submitted to the access control module which operates as the entry and the guard of the XML database. The user program submits each query together with the user ID, the user-specific database access password, and a symmetric key that has been generated at the client's side and that will be used by the database to encrypt the answer to the query. The access control module's public key can be used for secure transferal of query, user ID, password and symmetric key from the client to the access control module. When the access control module receives this encrypted message, it uses its own private key in order to decrypt the message containing the query, the user ID, the user-specific database access password, and the symmetric key for encrypting the answer. Thereafter, the access control module uses the user ID and the user-specific database access password to search in an access right table for the user's access rights or the access rights to be granted to the user's application program. Finally, the access control module uses the access rights in order to rewrite the query in such a way that the query can not violate the access rights. Altogether, XPath queries encrypted by a user program enter the access control module that decrypts the query and generates a secure query execution plan depending on the user's access rights. The encrypted exchange of queries and results prevents that other users read information sent to one user.

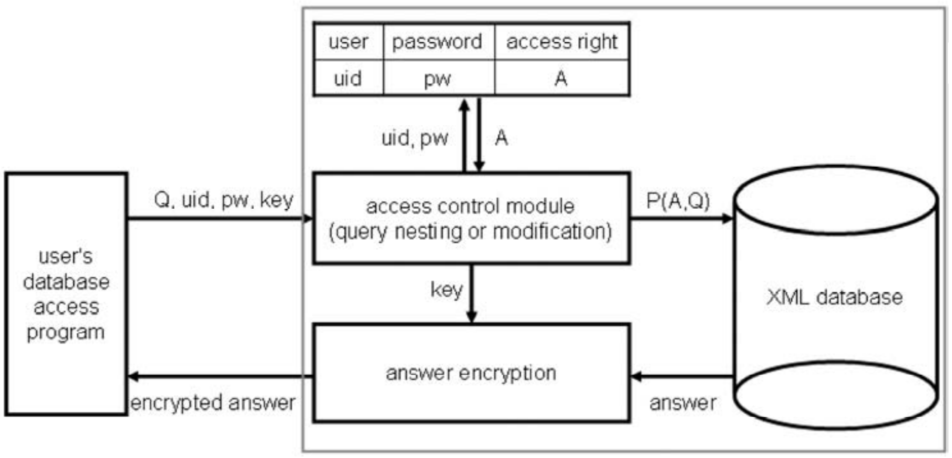

Figure 1: The role of the access control module within the overall system architecture

4. Access control based on a nested application of access rights and query

One solution to the information uncovering problem is to require a nested application of the access rights and the query, i.e. the query has to be applied only to that fragment of a document or XML database D for that an access right is granted. A natural query execution plan P(A,Q) following this approach is to mark or to copy that fragment of D to which access is granted and to execute the query on the marked or copied fragment. In other words, a query Q is translated into the following query execution plan P(A,Q).

C = treeCopyOf(A(D)) ;

return the query result of Q applied to C (instead of Q applied to D) ;

Whenever our query evaluator contains the treeCopyOf operation as an operator, this query execution plan can be expressed shorter as follows. The modified query execution plan P(A,Q), that answers a given query Q by accessing only the parts for which an access right A is granted, can be computed by

P(A,Q) = Q(treeCopyOf(A(D))) .

The operation treeCopyOf(A(D)) executes a query A on a document D and returns the fragment of D that contains all the leaf nodes of D that are selected by A plus all the paths to these selected leaf nodes.

5. The optimized approach to access control

The goal of the optimization is to avoid the computation of C if possible. Instead of computing C, the query Q and the access rights A shall be used to compute a transformed query QA. QA applied to D shall return exactly the same answer as Q applied to C, i.e. to A(D). The advantage of computing QA on a logical basis is that this approach avoids the physical computation (i.e. the construction) of A(D) alone for the purpose of answering the query Q. The advantage is the larger, the smaller the result of Q(A(D)) is compared to the accessible part A(D) of the XML document or XML database D.

In some applications for data streams, as for example an XML dissemination service [9], access control would be desirable. Consider for example the case where the access to certain parts of a data stream is sold. In this case, the above presented approach is not applicable, as it is not feasible to compute treeCopyOf(A(D)), but the next approach can be applied.

Our solution involves the rewriting of the original query Q. In order to obey an access right A, each query Q is rewritten to a query QA by the following query rewrite rule and by subsequent simplifications of the query formula of QA, which are described in the next section.

Query rewrite rule:

Given an XPath expression A that describes an access right,
a query Q is rewritten to a query QA
by restricting each occurrence of an element Ei to A, i.e. each element name Ei occurring in the query Q is replaced with (Ei ∩ A).

When we apply the query rewrite rule to query Q1 given in the example of Section 2 above, the query

Q1 = / E1 / E2 [2]

is rewritten to

Q1A = / (E1 ∩ A) / (E2 ∩ A) [2] .

By replacing A with the XPath expression for the access rights, i.e. with
/E1|(/E1/E2[not t="2"] #), and by simplifying the XPath expression afterwards, we get

```
Q1A = / ( E1 ∩ ( /E1|(/E1/E2[not t="2"] #) ) / ( E2 ∩ ( /E1|(/E1/E2[not t="2"]#) ) )[2]
    = / E1                                  / ( E2[not t="2"] ) [2]
```

Altogether, Q1A does not access the "forbidden" element E2, i.e. the element E2 with an attribute value of "2" for the attribute t. Thereby, Q1A returns the correct result, i.e. the element E2 with an attribute value of "3" for the attribute t.

When we apply the query rewrite rule to the query Q2 in the example of Section 1.2, the query

Q2 = /E1[./E2/@t="2" and ./E2/@t="3"]

is rewritten to

Q2A = / (E1∩A) [./(E2∩A)/@t="2" and ./(E2∩A)/@t="3"].

By replacing A with the XPath expression for the access rights, i.e. with
/E1|(/E1/E2[not t="2"] #), and by simplifying the XPath expression afterwards, we get

```
Q2A = / ( E1∩( /E1|(/E1/E2[not t="2"] #) ) )
      [ ./ (E2∩( /E1|(/E1/E2[not t="2"] #) )) / @t="2"
        and
        ./ (E2∩( /E1|(/E1/E2[not t="2"] #) )) / @t="3"
      ]
```

```
= / E1
        [ ./ ( E2[not t=”2”] ) / @t=“2”
          and
          ./ ( E2[not t=”2”] ) / @t=”3” ]
= / E1 [  false  and  ./  ( E2[not t=”2”] )  / @t=”3” ]
= ∅
```

Note that by returning the empty set, the query result reflects the fact that there is no element E2 with an attribute value of “2” for the attribute t for which an access right exists, i.e. when the access right is regarded, the answer is correct. Furthermore, note that the existence of an element E2 with an attribute value of “2” for the attribute t is not uncovered, because the query Q2A returns exactly the same answer, if this element is deleted from the document. This is the reason why the query rewrite rule prevents queries from uncovering information for which an access right does not exist. In other words: this is the way the query rewrite rule guarantees that access rights are preserved.

6. Simplification rules for XPath queries

The simplification rules are used to reduce the complexity of XPath query expressions after the query rewrite rule has modified the query in such a way that it respects the access rights. Simplification rules are equivalence transformations on XPath expressions that are applied as long as possible.

Let W, X, Y, and Z be XPath expressions, whereas W, X, Y and Z can denote the empty path ε, let E1 and E2 be node name tests, and let F1 and F2 be filter expressions all of which are occurring within a modified query. Further, let ∅ denote the empty node set and A and B denote axis specifiers. Then the access control module applies the following kinds of simplification rules wherever possible by replacing the left hand side of the rule with the right hand side:

1. Rewrite rules that extract absolute XPath expressions

/ X / (/ X / Y)	→ / X / Y
/ (/ X)	→ / X
X / (Y ∩ /Z) / W	→ X / Y / W ∩ /Z / W
X / (Y | /Z) / W	→ X / Y / W | /Z / W

2. Rewrite rules that remove or un-nest “\|”-operators:

X | X	→ X
X / Y | X#	→ X#
X | ∅	→ X
X ∩ (Y | Z)	→ (X ∩ Y) | (X ∩ Z)
(X | Y) [F1]	→ X [F1] | Y [F1]
Y / X [F1] | Y / X# [F1]	→ Y / X# [F1]
Y / X [F1] | Y / X [F2]	→ Y / X [F1 or F2]

3. Rewrite rules that remove or un-nest “∩”-operators:

X ∩ X	→ X
X / Y ∩ X#	→ X / Y
X ∩ ∅	→ ∅

X / A::E1 [F1] $\cap$ Y / B::E2 [F2]	$\rightarrow$	$\varnothing$	if E1 $\neq$ E2
X / Y $\cap$ X	$\rightarrow$	$\varnothing$	if Y $\neq$ ε
X / child::E1 [F1] / Y $\cap$ X / child::E2 [F2] / Z	$\rightarrow$	$\varnothing$	if E1 $\neq$ E2
/ E1 [F1] / Y $\cap$ / E2 [F2] / Z	$\rightarrow$	$\varnothing$	if E1 $\neq$ E2
Y / X [F1] $\cap$ Y / X [F2]	$\rightarrow$	Y / X [F1 and F2]	
Y / X [F1] $\cap$ Y / X# [F2]	$\rightarrow$	Y / X [F1 and F2]	

4. Further rewrite rules to simplify XPath expressions:

X / $\varnothing$ / X	$\rightarrow$	$\varnothing$	
$\varnothing$ #	$\rightarrow$	$\varnothing$	
$\varnothing$	$\rightarrow$	false	if $\varnothing$ is within a filter
X [false]	$\rightarrow$	$\varnothing$	
false and X	$\rightarrow$	false	
true and X	$\rightarrow$	X	
X and X	$\rightarrow$	X	
false or X	$\rightarrow$	X	
true or X	$\rightarrow$	X	
X [not Y] / Y	$\rightarrow$	$\varnothing$	
E1	$\rightarrow$	E1 [true]	

Within each group, there are more similar rules, which we have omitted in order to keep the presentation of the rules simple. For example, there are additional symmetric rules for every rule containing a commutative operator like intersection (“$\cap$”), union (“|”), the logical “and”, and the logical “or”.

Notice that it is not essential that all intersection-operators are omitted by the rewrite rules, because the intersection can be computed based on the data on the server side. Even in the case of data streams the intersection can be computed (see [9, 10] for details). However, the simplification of the generated XPath query will result in a lower computation time on the server.

7. Adaptive access control

We have proposed two solutions for a query execution plan that preserves access rights. The approach presented in Section 4 is based on the nested execution of the access rights and the query, e.g. by computation of a copy of that fragment for which an access right exists and executing the query on that fragment. This approach is useful, when the access rights cover a very small portion of a document, e.g. when a user is allowed to access its own account information only. The other approach presented in Sections 5 and 6 based on rewriting the query and following simplification of the modified query is useful, when an access right covers a large portion of the database. For example, whenever most of the product information is available to most of the customers, but small portions of product information are private or are accessible to only certain user groups, the solution based on query rewriting is preferable, because it avoids copying of large fragments of the product information XML database.

Depending on the amount of data which is covered by the access rights and depending on the complexity of the query to be rewritten, the access control module can switch between both methods of access control. In other words, our approach provides an adaptive access control module which chooses the appropriate access control method, depending on query complexity and depending on the size of the fragment for which access is granted to the user.

8. Summary and conclusions

We have developed an approach to access control which allows us to use a single master copy of XML data or documents that can be accessed by multiple user groups or users. This master copy is not directly accessible by any user or any user application. Instead user applications submit their query via an access module that operates as a guard to the master data source. This architecture allows us to integrate query execution plans into the access module that can not be circumvented by users or user applications. Each query execution plan takes the user ID and the user's password to look up the user's access rights. These access rights are used within the query execution plan to modify the query in such a way that the access limitations can not be violated. Our system distinguishes two types of query execution plans, such that the access control module chooses the more appropriate query execution plan, depending on the access rights and on the query. While one execution plan copies the data for which an access right exists to a temporary data structure upon which the query is executed securely, the second query execution plan is based on query modification. Query modification rewrites an XPath query in such a way that every element occurring in a location step of the XPath query is restricted according to the access rights. The resulting secure query is simplified in such a way that both secure and efficient query processing is guaranteed, i.e. the query avoids access violations, but does not require to copy the fragment of the XML data source to which access is granted. We consider this approach to be especially advantageous, whenever a user or user group has access to large parts of the hidden XML database.
The approach contributed is neither limited to the use of XML data within a single company nor limited to the XML access language XPath. It seems to be possible to extend the results to cross-enterprise XML data exchange and to other XML query languages like XQuery and to XML transformers like XSLT too. Furthermore, our approach to access control is compatible with query optimization on transformed XML data [12], which is used for cross-enterprise XML data exchange. Therefore, we consider it to be a challenging research topic to investigate, how our approach to efficient and secure query processing can be extended to meet the requirements of XML-based cross-enterprise information systems.

References:

[1] Elisa Bertino , Silvana Castano , Elena Ferrari: On specifying security policies for web documents with an XML-based language, Proceedings of the Sixth ACM Symposium on Access control models and technologies, May 2001.
[2] Elisa Bertino, Elena Ferrari: Secure and selective dissemination of XML documents. *TISSEC*, 5(3):290–331, 2002.
[3] S. Böttcher, A. Türling: Access Control and Synchronization in XML Documents. In Proceedings: XML Technologie für das Semantic Web, Berlin 2002, Springer, LNI P-14, 2002.

[4] S. Böttcher, A. Türling. Checking XPath Expressions for Synchronization, Access Control and Reuse of Query Results on Mobile Clients. In Birgitta König-Ries, Michael Klein, Philipp Obreiter (Hrsg.): Database Mechanisms for Mobile Applications. Proceedings of the 2003 Spring Workshop of the Working Group on Mobile Databases and Information Sytems within the German Informatics Society (GI). Karlsruhe, April, 2003.
[5] S. Böttcher, R. Steinmetz: Testing Containment of XPath Expressions in order to Reduce the Data Transfer to Mobile Clients. 7th East European Conference on Advances in Databases and Information Systems, Dresden, September, 2003.
[6] Ernesto Damiani, Sabrina di Virmercati, Stefano Paraboschi, Pierangela Samarati: Securing XML Documents. Proc. of the 7 th Int. Conf. on Extending Database Technology (EDBT), Konstanz, March, 2000. Springer, LNCS, Volume 1777.
[7] Ernesto Damiani, Sabrina di Vimercati, Stefano Paraboschi, Pierangela Samarati: XML Access Control Systems: A Component-Based Approach. DBSec 2000.
[8] Ernesto Damiani, Sabrina di Vimercati, Stefano Paraboschi, Pierangela Samarati. A fine-grained access control system for XML documents. *TISSEC*, 5(2):169–202, 2002.
[9] Yanlei Diao, Shariq Rizvi, Michael J. Franklin: Towards an Internet-Scale XML Dissemination Service. VLDB 2004: 612-623
[10] Yanlei Diao, Mehmet Altinel, Michael J. Franklin, Hao Zhang, Peter M. Fischer: Path sharing and predicate evaluation for high-performance XML filtering. ACM Trans. Database Syst. 28(4): 467-516 (2003)
[11] Wenfei Fan, Chee-Yong Chan, Minos Garofalakis: Secure XML Querying with Security Views. SIGMOD 2004.
[12] S. Groppe, S. Böttcher, G. Birkenheuer. Efficient querying of transformed XML documents. 6th International Conference on Enterprise Information Systems. Porto, Portugal, April 2004.
[13] Michiharu Kudo, Satoshi Hada: XML document security based on provisional authorization, Proceedings of the 7th ACM conference on Computer and communications security, 2000.
[14] Gerome Miklau, Dan Suciu: Containment and Equivalence of XPath Expressions. PODS 2002: 65-76.
[15] G. Miklau and D. Suciu. Controlling access to published data using cryptography. VLDB, Berlin, Germany, 2003.
[16] Neven, F., Schwentick, T.: XPath Containment in the Presence of Disjunction, DTDs, and Variables. ICDT 2003: 315-329.
[17] Stonebraker, M.: Implementation of Integrity Constraints and Views by Query Modification. ACM SIGMOD 1975.
[18] Yue Wang, Kian-Lee Tan: A Scalable XML Access Control System. WWW Posters 2001.
[19] Peter T. Wood: Containment for XPath Fragments under DTD Constraints. ICDT 2003: 300-314.
[20] XPath: XML Path Language (XPath) Version 1.0. W3C Recommendation, November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.
[19] Ting Yu, Divesh Srivastava, Laks V. S. Lakshmanan, H. V. Jagadish: Compressed Accessibility Map: Efficient Access Control for XML. VLDB 2002