Securely Updating XML

Ernesto Damiani[1], Majirus Fansi[2], Alban Gabillon[2], and Stefania Marrara[1]

[1] Università degli Studi di Milano, Dipartimento di Tecnologie dell'Informazione
via Bramante 65 26013 Crema (CR), Italy
{damiani, marrara}@dti.unimi.it
[2] Université de Pau et des Pays de l'Adour, IUT des Pays de l'Adour
40004 Mont-de-Marsan, France
janvier-majirus.fansi@etud.univ-pau.fr, alban.gabillon@univ-pau.fr

Abstract. We study the problem of updating XML repository through security views. Users are provided with the view of the repository schema they are entitled to see. They write update requests over their view using the XUpdate language. Each request is processed in two rewriting steps. First, the XPath expression selecting the nodes to update from the view is rewritten to another expression that only selects nodes the user is permitted to see. Second the XUpdate query is refined according to the write privileges held by the user.

1 Introduction

In the last few years, the *eXtensible Markup Language* (XML)[21] has become the format of choice for data interchange. XML-based systems are now widely deployed in a number of application fields. This success has triggered a growing interest in XML security, and several schemes for XML access control have been proposed (e.g., [6, 8–10, 13, 17, 18]. However, most of the existing proposals mainly focus on the read privilege. Few of them consider write privileges.

In [7], we devised a Deterministic Finite Automaton (DFA) based technique to rewrite user query over a virtual view Dv to an equivalent one over the database. The user is provided with the view Sv of the repository schema she is entitled to see. The schema Sv is as if the user virtual view Dv were materialized then it would be valid w.r.t. Sv. As the view is not materialized, whenever the user writes an XPath query against Dv, the query is rewritten to an equivalent one over the database by the automaton.

The goal of this paper is to exploit the work done in [7] to securely handle XUpdate [14] commands over views. Namely, the authorization designer annotates the repository schema with some write attributes (*insert, update, delete*). The annotated schema is afterward translated into a Deterministic Finite Automaton (*for updates*). Rewriting an XUpdate request into a safe one is done in two steps:

- Whenever a user sends an XUpdate request over her view, we first rewrite the expression selecting the nodes to be updated according to the principles described in [7](see Section 2). This step is necessary since the user should not be able to update nodes she is not entitled to see.

- Then, we rewrite the XUpdate command over the DFA for updates in order to obtain a safe query (i.e a query updating the nodes the user is permitted to update).

The rest of the paper is organized as follows. Section 2 reviews the principles of our approach [7] for rewriting a user query over view to an equivalent one over the base repository. In section 3, we present our approach for securely handling update queries over view. Section 4 compares our proposal to related work. Finally section 5 gives a conclusion and discusses future work.

2 Securely Querying XML

In this section we briefly review the principles of our approach for rewriting potentially unsafe user queries into safe ones. An *unsafe query* is a query which, if executed over the repository as is, can return nodes that the querist is not allowed to see. The reader is invited to refer to [7] for a complete presentation of the rewriting procedure. Our technique is based on *Deterministic Finite State Automata* (DFA) and consists of four steps:

Step 1: The authorization designer inserts the authorization attributes in the XML Schema of the document collection creating the annotated schema. These attributes include *access, condition* and *dirty.* Attribute *access* specifies the rights of the user on the node according to the authorization language. The value of this attribute is either *allow* or *deny.* Attribute *condition* contains a list of predicates that have to evaluate to true for access to be granted. Finally, attribute *dirty* indicates that some descendants of the current node could be unauthorized.

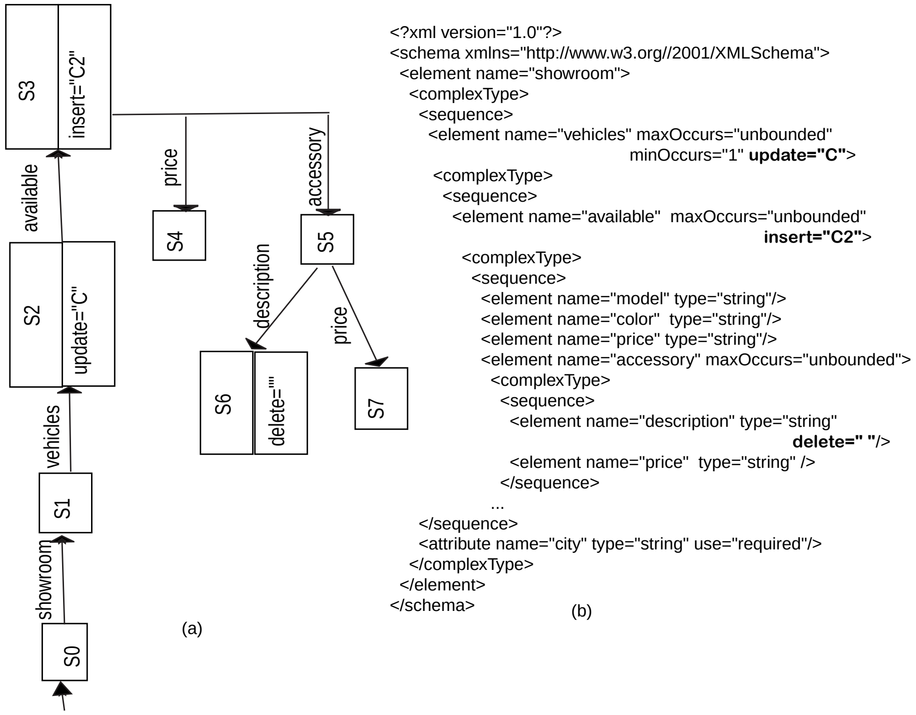
Step 2: The XML schema is transformed according to the policy that applies to each role. According to her role, the user is provided with the view of the schema (in short Sv) she is entitled to see. Then she writes her query using information available on Sv. Hereafter, unless stated otherwise, the term *view* will refer to the view of the schema and not to the view of a source document.

Step 3: The annotated schema is translated into an automaton which represents the structure of Sv. Each state within Sv contains some security attributes that will further serve us while rewriting the user request.

Step 4: Finally, the user query is rewritten using the finite state automaton (see [7] for the procedure).

We employ the answer-as-tree model, i.e., we return to the user the subtree rooted at the node targeted by the rewritten expression. As a consequence, we use the XPath function *except* to discard any forbidden node which is descendant of the target node.

In [7], we proved that our procedure is correct and efficient. The reader is then invited to refer to that paper for details.



(a)

```
<?xml version="1.0"?>
<schema xmlns="http://www.w3.org//2001/XMLSchema">
  <element name="showroom">
    <complexType>
      <sequence>
        <element name="vehicles" maxOccurs="unbounded"
                                          minOccurs="1" update="C">
         <complexType>
           <sequence>
             <element name="available"  maxOccurs="unbounded"
                                                      insert="C2">
               <complexType>
                 <sequence>
                   <element name="model" type="string"/>
                   <element name="color"  type="string"/>
                   <element name="price" type="string"/>
                   <element name="accessory" maxOccurs="unbounded">
                     <complexType>
                       <sequence>
                         <element name="description" type="string"
                                                         delete=" "/>
                         <element name="price"  type="string" />
                       </sequence>
                     ...
        </sequence>
        <attribute name="city" type="string" use="required"/>
      </complexType>
    </element>
</schema>
```

(b)

Fig. 1. automaton for updates (a) and write privilege annotated schema (b)

3 Updating XML

Updating XML data can still be considered a research issue (e.g. see [2, 3, 14, 19, 20]); however at least some building blocks of a data manipulation language for XML are now firmly in place. XUpdate is an XML-based host language for instructions tailored for update tasks. In other words, it expresses updates as well-formed XML documents; specifically, each update is represented by an `xupdate:modifications` element. XUpdate has now over a dozen implementations; this relative success is due to the fact that it is easy to understand and simple to implement. XUpdate operations have a required `select` attribute. The value of this attribute is a XPath expression which selects the nodes to update, referred to as *context* nodes. Besides updating XML content, XUpdate operations can create and delete entire XML fragments. An example XUpdate instruction is `<remove select='//vehicles'>`. This command is to remove from a document every fragment which root is an element named `vehicles`. The interested reader can refer to [14] for a complete description of the XUpdate Language. In our model, we consider the following write privileges: `DELETE, INSERT`, and `UPDATE`. The semantics of these privileges can be stated as follows:

– if user s holds the INSERT privilege on node n then user s has the right to add a new sub-tree to node n.
– if user s holds the UPDATE privilege on node n then user s has the right to update node n (i.e., change the values of its immediate children of type text).
– if user s holds the DELETE privilege on node n then user s has the right to delete the sub-tree of which node n is the root.

Below, for each XUpdate operation we list the write privilege that user s should hold.

Creating node operations There are three XUpdate instructions for creating XML fragments: insert-before, insert-after and append.

Insert-before inserts a given fragment as the preceding sibling of every context node, and insert-after inserts it as the following sibling of every context node. The operation append allows a node to be created and appended as a child of every context node.

– insert-before/insert-after: user s needs the INSERT privilege on the parent node of every context node.
– Operation append: user s needs the INSERT privilege on every context node.

Update operations There are two XUpdate instructions for updating XML nodes: update and rename. Operation update can be used to update the content of existing nodes. Operation rename allows an attribute or element node to be renamed after its creation.

– update: if context nodes are elements, then user s needs the UPDATE privilege on the content (text node) of every context node. If context nodes are attributes, then user s needs the UPDATE privilege on every context node.
– rename: user s needs the UPDATE privilege on every context node.

Renaming an attribute or updating its value requires the UPDATE privilege on the context node. This choice is consistent with the XPath data model, where an attribute node encapsulates both the attribute and its value. On the contrary, renaming an element requires the UPDATE privilege on the context node and updating its content requires the UPDATE privilege on the content node itself (i.e., the text child of the context node).

Delete operations There is one XUpdate instruction for deleting XML fragments: remove. Operation remove deletes all sub-trees having a context node as the root. For this operation, user s needs the DELETE privilege on every context node.

3.1 Securing update operations

Simply considering the write privileges held by a subject is not sufficient to make XML updates secure. The reason for this can be best understood by considering an analogy with SQL. Let us consider *user_A* who is the owner of an Employee database table and who has granted to *user_B* the UPDATE privilege on it. As a result, user_B is not permitted to see user_A's Employee table

```
   SQL> SELECT * FROM user_A.employee;
ERROR ORA-01031: insufficient privilege
```

but user_B is permitted to update it:

```
   SQL> UPDATE user_A.employee SET salary=salary+100 WHERE salary
> 3000; 2 rows updated
```

The simple example above shows that although user_B was not permitted to see user_A's employee table, she was able to learn, through an update command, that there were two employees with a salary greater than 3000. This is due to the fact that the `WHERE` clause did perform a `read` operation on `Employee` despite the fact that user_B did not hold the `SELECT` privilege on that table. In XUpdate operations the `select` attribute plays the same role as the `WHERE` clause in a SQL `UPDATE/DELETE` command. Therefore, in order to avoid the inference problem caused by write operations performing read action, we rewrite the XPath expression selecting context nodes according to the principles described in Section 2. Securely controlling an XUpdate operation is then done in two steps.

1. The XPath expression selecting the context nodes is rewritten according to the read privileges held by the user submitting the XUpdate operation. This step is described in section 2. It corresponds to the work presented in [7] and uses the DFA for queries. However, when rewriting the XPath expression, we use the *answer-as-nodes* technique which stipulates that the XPath expression should return the target nodes rather than the entire sub-trees rooted at them. Consequently, we spare the operation *except* that eliminates forbidden nodes within the sub-tree rooted at the target node.
2. The XUpdate operation should succeed for the context nodes on which user s holds the proper write privilege and fail for the others. In order to implement this principle, we rewrite a second time the XPath expression selecting the context nodes, so that only the nodes on which user s holds the proper write privilege are selected.

For rewriting the XPath expression according to the write privileges held by the user, we use the following technique: the policy author inserts for each user class (role), the authorization attributes in the XML Schema of the document collection creating the annotated schema. These attributes include *insert, update* and *delete*. The value of these attributes is either empty or equal to a list of predicates stating under which conditions the operation should be performed. A sample annotated schema is shown in Figure 1(b). The annotated schema is afterwards translated into a deterministic finite automaton for updates (see Figure 1(a)). The automaton traverses its states according to the tokens[3] of the rewritten expression produced by step 1, until the last token gets through.

[3] We call token a step in the path expression, for example `showroom` is the first token in `/showroom/vehicles/available`, while `vehicles` is the second. `/` stands for a `lookahead`.

Then, the automaton transits to the state corresponding to the target node of the expression. At this position, the finite state machine behaves as follows[4]:

Case 1: The operation is `insert-before` or `insert-after`. The automaton backtracks to the previous state, which is the state corresponding to the parent of the context node. Indeed, the user needs the `INSERT` privilege on the parent node of every context node. If the attribute insert is present at that state then its (possibly empty) value is appended to the XPath expression and returned. If the attribute insert is not present then the expression is rejected.

Case 2: The operation is `rename` or `update`. If the attribute `update` is present then its (possibly empty) value is appended to the XPath expression and returned. If the attribute update is not present then the expression is rejected.

Case 3: The operation is `remove`. If the attribute `delete` is present then its (possibly empty) value is appended to the XPath expression and returned. If the attribute `delete` is not present then the expression is rejected.

Case 4: The operation is append. If the attribute `insert` is present then its (possibly empty) value is appended to the XPath expression and returned. If the attribute `insert` is not present then the expression is rejected.

It is worthwhile making a few observations about the operation `remove`. Let us consider a `remove` operation on a node `n`. When the user removes node `n` then she actually deletes the sub-tree rooted on node `n`. Some of the nodes which belong to that sub-tree may not be visible (i.e. the user may not be permitted to see them). Shall we reject the operation if some nodes of the deleted sub-tree do not belong to the user's view? On one hand, this would preserve the integrity of data the user is not permitted to see. On the other hand, it would reveal to the user the existence of data she is not entitled to see. In fact there is no definite answer to this question. This is typically a case of conflict between confidentiality and integrity. Here, we prefer to emphasize the confidentiality, and the command is accepted.

4 Related Work

The problem of secure updates for XML documents was first addressed in [15]. Authors in [15] extend the model presented in [6] with write operations and suggest a technique for managing access controls in a web environment which emphasizes the integrity of the documents. The model in [15] requires provision for view materialization. In general view-based enforcement schemes suffer from high maintenance and storage costs, especially for large XML repository.

The problem of updating databases through views was first reported by Codd [5] and has been widely studied ever since. Especially in the area of relational DataBase Management Systems (DBMS). Recently, proposals have appeared regarding updating XML Views (eg. [1, 11, 4]). For Instance, [1] focuses on translating XML view updates to relational view updates and delegating the problem to the relational DBMS; however, as pointed out in [4] most commercial DBMSs

[4] Here, for the sake of simplicity, we consider only commands and privileges addressing element nodes.

only have limited view-update capability. A step forward is done in [11]. Authors in [11] propose a solution based on schema level analysis for determining whether an update over XML views is translatable and for finding the translation (if one exists), while considering the constraints in the XML schema. This work is similar to our approach, but is too limited as the authors consider only deletion of single element node. Authors in [4] investigate the view update problem for XML views published from relational data. Given the XML view of a relational database, authors in [4] need to propagate updates of the XML view to the original relational tables, without compromising the integrity of neither the XML nor the relational data. [4] present two main disadvantages. First, the views are materialized. Second, there is no efficient algorithm for insertions in relational views as the operation is NP-complete. On the contrary, our approach is based on our efficient technique (see section 2) for translating query expressed on views to equivalent query on data source.

Another approach [16] proposes an infrastructure and related algorithms for cooperative updates of XML documents. Key components of the proposed system are a set of XML-based languages for specifying access control policies and the path that the document must follow during its update. Such path can be fully specified before the update process begins or can be dynamically modified by properly authorized subjects while being transmitted from users to users. Thus [16] is designed for cooperative contexts where documents have to transit between many subjects; while our approach is designed for centralized XML repository.

5 Conclusion

In this paper, we describe a DFA based approach for securely handling XUpdate commands over XML Repository through views. We highlight how our approach improves previous works in the area. Our proposal leaves space for further work. Experiments and proof of correctness remain work to be done. Updating an XML document may invalidate it regarding the repository schema. In [12], authors develop a mechanism aiming to determine if an update would invalidate the document. We are investigating the possibility to leverage the essence of [12] in order to verify, prior to the execution of a query, that the document will remain valid afterward.

Acknowledgments: This work was supported in part by the Italian Basic Research Fund (FIRB) within the contract n. RBNE05FKZ2_004, TEKNE, by the European Union within the PRIME Project in the FP6/IST Programme under contract IST-2002-507591 and by funding from the French ministry for research under "ACI Sécurité Informatique 2003 - 2006. projet CASC". Majirus Fansi holds a Ph.D scholarship granted by the "Conseil Général des Landes". The authors wish to thank Sabrina De Capitani di Vimercati for valuable suggestions.

References

1. V. P. Braganholo, S. B. Davidson, and C. A. Heuser. From xml view updates to relational view updates: old solutions to a new problem. *In Proc. of the 30th VLDB Conference*, 2004.

2. E. Bruno, J. L. Matre, and E. Murisasco. Extending xquery with transformation operators. *In Proc. of the 2003 ACM Symposium on Document Engineering (DocEng 2003)*, 2003.
3. D. Chamberlin, D. Florescu, and J. Robie. Xquery update facility. *W3C working draft*, May 2006.
4. B. Choi, G. Cong, W. Fan, and S. D. Viglas. Updating recursive xml views of relations. *In Proc. of ICDE*, 2007.
5. E. F. Codd. Recent investigations in relational data base systems. *In IFIP Congress*, pages 1017–1021, 1974.
6. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Securing xml documents. *In Proc. of the 2000 International Conference on Extending Database Technology (EDBT2000)*, March 2000.
7. E. Damiani, M. Fansi, A. Gabillon, and S. Marrara. A general approach to securely querying xml. *In Proc. of the 5th International Workshop on Security in Information Systems (WOSIS 2007)*, June, 2007.
8. S. de Capitani di Vimercati, S. Marrara, and P. Samarati. An access control model for querying xml data. *In Proc. of SWS05 workshop*, 2005.
9. W. Fan, C. Chan, and M. Garofalakis. Secure xml querying with security views. *In Proc. of SIGMOD 2004 Conference*, 2004.
10. A. Gabillon. A formal access control model for xml databases. *In Proc. of the 2005 VLDB Workshop on Secure Data Management (SDM)*, 2005.
11. M. Jiang, L. Wang, M. Mani, and E. Rundensteiner. Updating views over recursive xml. *In Proc. of ICDT Worshop on Emerging Research Opportunities in Web Data Management*, 2007.
12. B. Kane, H. Su, and E. L. Rundensteiner. Consistently updating xml documents using incremental constraint check queries. *In Proc. of ACM WIDM*, 2002.
13. M. Kudo and S. Hada. Xml document security based on provisional authorization. *In Proc. of ACM CCS*, 2000.
14. A. Laux and L. Martin. Xml update language (xupdate). *xml:db working draft, http://xmldb-org.sourceforge.net/xupdate*, 2000.
15. C. Lim, S.Park, and S. H. Son. Access control of xml documents considering update operations. *In Proc. of ACM Workshop on XML Security*, 2003.
16. G. Mella, E. Ferrari, E. Bertino, and Y. Koglin. Controlled and cooperative updates of xml documents in byzantine and failure prone distributed systems. *In Proc. of ACM Transactions on Information and System Security, vol. 2 No. 3*, pages 1–32, 2005.
17. S. Mohan, A. Sengupta, Y. Wu, and J. Klinginsmith. Access control for xml - a dynamic query rewriting approach. *In Proc. of VLDB 2005 Conference*, 2005.
18. M. Murata, A. Tozawa, and M. Kudo. Xml access control using static analysis. *In Proc. of CCS*, 2003.
19. G. M. Sur, J. Hammer, and J. Simeon. updatex-an xquery-based language for processing updates in xml. *In Proc. of the 2004 International Workshop on Programming Language Technologies for XML (PLAN-XML)*, 2004.
20. I. Tatarinov, Z. G. Yves, A. Y. Halevy, and D. S. Weld. Updating xml. *In Proc. of ACM SIGMOD*, 2001.
21. W3C. extensible markup language (xml). *http://www.w3.org/XML/*.