A Scalable XML Access Control System

Yue Wang Kian-Lee Tan
Department of Computer Science
National University of Singapore
Science Drive 2, Singapore 117543
{wangyue,tankl}@comp.nus.edu.sg

ABSTRACT

This paper presents the design of a scalable XML access control system. The system has the following features. First, it can regulate access control at a fine graularity (e.g., at the tag level). Second, it stores XML documents as tables in relational databases. Third, it is efficient compared to existing systems as it only examines the relevant data. Fourth, it is scalable as it can handle very large XML documents that may not fit into the main memory. Finally, it provides very fast initial response time.

Keywords

XML, access control, relational database, scalable.

1. INTRODUCTION

In recent years,there is an increasing amount of information being distributed and shared in XML format over coporate Intranets and the global Internet. As a result, it becomes critical to define and enforce access restrictions on XML documents to ensure that only authorized users have access to the information. Clearly, the simple method of protecting an entire document at the file level is unattractive (since it will limit the dissemination of information) and unnecessary (as different users may be allowed to access different portions of the document).

One promising access control model proposed in the literature is to enforce access restrictions directly on the structure and content of XML documents [1, 2]. In this way, information in XML format can be protected at a finer level of graularity than the whole document, e.g., at the tag level. As an example, a user may be blocked from accessing information tagged by `XLink` and `XPointer`, while another may be allowed to access the entire document.

The design of an access control systems based on this model were reported in [1, 3]. The systems essentially represent XML documents as object trees, according to the Document Object Model (DOM) Level 1 specification [6]. DOM provides an object-oriented Application Program Interface (API) for HTML and XML documents. To enforce security, the DOM tree is repeatedly traversed to mark out nodes that should be denied access. This step is done based on the authorization rules for the user. Finally, the DOM tree is traversed to prune away the nodes that are marked, and the remaining DOM structure represents information that can be accessed by the user. This architecture has two main problems. First, it requires the entire DOM tree to be memory resident. This problem is exacerbated by the fact that a DOM tree is typically much larger (about 5 to 10 times) than its XML document. Thus, the scheme is not scalable for large XML documents. Second, the initial response time is long. This is because all the steps must be completed before any answers are returned to the user. Third, the scheme is inefficient. This is because the DOM tree has to be repeatedly traversed and its nodes labeled. More importantly, a lot of unnecessary information is also loaded (in the sense that the system loads the entire XML document and then prune away information that should not be disclosed.). To overcome these limitations, novel design of access control systems is necessary.

2. XENA: AN XML SECURITY ENFORCEMENT ARCHITECTURE

In this paper, we present the design of an access control system, XENA (Xml sEcurity eNforcement Arctecture), that we are currently implementing at the National University of Singapore.

XENA has several characteristics that are desirable for a secure system for XML documents:

1. From the user (end user or admininstrator) point of view, there is only one “language” and one format. In other words, a query is specified either in a XML query language (e.g., XML-QL [4]) or by an URL (if an XML document is to be accessed). Similarly, the access control administrator specifies his access control policies in an XML format. We note that the access control can be defined on both XML instances and XML schema. While we have adopted the access control model in [1], we have implemented the model so that it operates on XML schema (which is expected to replace DTD very soon). Thus, in XENA, every XML document has its own XML Access Sheet (XAS) that is specified in XML.

2. XML documents are stored as tables in relational databases. This allows us to tap into the strengths of relational databases. In particular, by storing XML documents as tables, we only need to bring in the necessary data – attributes that are not needed can be projected out, and tuples that are not needed can be also be filtered out easily. In addition, the iterator model of evaluating queries allow answers to be returned as soon as they are produced. This facilitates fast initial response time.

   To store XML documents as tables, we need to map the content of XML documents into tuples of tables. In this work, we have adopted the XStorM mapping strategy [5]. XStorM employs a data-mining strategy to identify relations and their attributes. Note that a single XML file can be mapped into multiple tables. In other words, we may need to join multiple tables to produce the original XML document. Since our focus is not on the mapping strategy here, and due to space constraint, interested readers are referred to [5] for further details.

3. The system is scalable. Because operations at the backup (XML storage) are performed on tables, the memory requirement is independent of the size of the XML documents. As such, XENA can handle very large XML documents.

3. ACCESS CONTROL IN XENA

Figure 1 shows the architecture of XENA. Initially, all XML documents are mapped into relational tables by the **XML-Relation Transformer** module. The process also maintains the mapping information in the metadata database, including the correspondences between the XML document objects and the relation names, and the XML document attributes and the relation attributes. New documents are preprocessed and added in a similar manner.

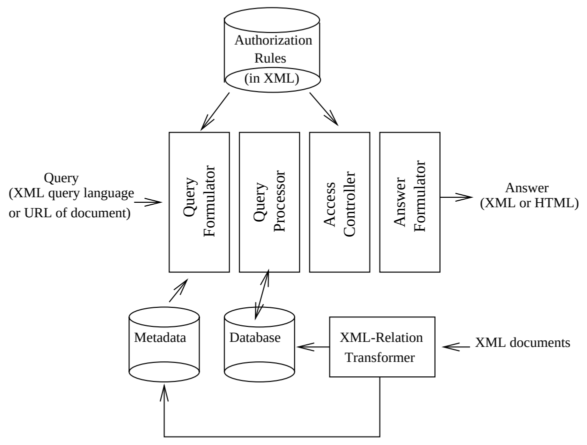

Figure 1: Architecture of XENA.

When a query arrives, the **Query Formulator** parsed the query and maps it into an internal query. Th module essentially examines the authorization rules and the metadata corresponding to the requested documents, and produces a query that will retrieve only those information that can be accessed by the requester. For example, if the requester is not allowed to see a certain attribute, then this attribute will be filtered out (by not specifying it in the target list) so that it is not necessary to access it from the database. Similarly, those tuples that the requester are not allowed to access are also filtered out (by introducing a selection predicate to prune them away).

The internal query is then evaluated by the **Query Processor**. This module accesses the data from the database according to the internal query, i.e., only the relevant information will be accessed.

Note that the requester may still not be able to view all the retrieved data. This is because there may be special cases that cannot be handled by the Query Formulator. For example, the requester may be allowed to view attributes A, B and C. However, (s)he is not allowed to look at those tuples whose B values are say 5, 14, 27 and 39. Note that while we can introduce selection predicates on B, this may be very tedious if the list of exceptions are not small and does not follow any pattern. As such, it may be easier to allow them to be accessed first, and then prune away later. This is exactly the task of the **Access Controller** module. It basically applies the set of authorization rules that cannot be enforced by the Query Formulator.

Finally, the **Answer Formulator** formulates the output in a form (e.g., XML or HTML) for the requester. We note that XENA only allows the users to view what (s)he is allowed to access and nothing else.

4. CURRENT STATUS AND PRELIMINARY RESULTS

We have implemented and evaluated most of the components independently. We expect the full system to be integrated within the next few months. Our preliminary experience with the various components have been good. For example, XStorM shows that storing XML documents as tables can cut down the total response time significantly compared to accessing the entirety of the XML docucments. In particular, as soon as one answer is retrieved, it can be returned to the requester. The access control model can also correctly grant and deny accesses.

Acknowledgements

This work is partially supported by a research grant funded by the National University of Singapore and the National Science and Technology Board. We would like to thank the authors of [1] for providing us with their source code. This allows us to reuse a large part of the code in our implementation while we convert to XML schema (instead of DTD).

5. REFERENCES

[1] E. Damiani, S. Vimercati, S. Paraboschi, and P. Samarati. Design and implementation of an access control processor for xml documents. In *Proceedings of the 9th International WWW Conference*, Amsterdam, May 2000.

[2] E. Damiani, S. Vimercati, S. Paraboschi, and P. Samarati. Securing xml documents. In *Proceedings of the 2000 International Conference on Extending Database Technology (EDBT'2000)*, Konstanz, Germany, March 2000.

[3] E. Damiani, S. Vimercati, S. Paraboschi, and P. Samarati. Xml access control systems: A component-based approach. In *Proceedings of the 14th IFIP 11.3 Working Conference in Database Security*, Amsterdam, August 2000.

[4] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. Xml-ql: A query language for xml. In *Proceedings of the Query Languages Workshop (QL'98), http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/*, 1998.

[5] W.Q. Wang, M.L. Lee, B.C. Ooi, and K.L. Tan. Xstorm: A scalable storage mapping scheme for xml data (Poster). In *WWW'10*, 2001.

[6] World Wide Web Consortium (W3C). Document object model (dom) level 1 specification verison 1.0. In *http://www.w3.org/TR/REC-DOM-Level-1*, 1998.