International Journal of Latest Research in Science and Technology
Volume 4, Issue 6: Page No.10-15, November-December 2015
https://www.mnkpublication.com/journal/ijlrst/index.php

ISSN (Online):2278-5299

ANALYZING XPATH REFERENCES IN XML ACCESS CONTROL

Qassim AlMahmoud[1], Ayman Nayef Ahmad Alhalaybeh[2]
College of Science and Arts, Tanumah
Department of Computer Science
King Khalid University
Kingdom of Saudi Arabia
Email:qassimalmahmoud@gmail.com
aiman68@hotmail.com

Abstract*- The trade-off between certain attributes such as efficiency and cost is always present when it comes to developing applications. Different approaches have been proposed that address the problem of protecting XML documents from unauthorized access especially at the granularity level. Some approaches deploy documents into memory to deliver fast runtime results, and some rather to process access requests statically by labeling each node within XML documents to avoid multiple checks when decisions are made. Web services and e-commerce applications are definitely increasing in day by day bases, access control approaches are highly encouraged to consider these high demands when developing any model. We propose an XML access control model that Eliminates the need of accessing XML documents in databases by analyzing references to XML objects. Giving an access control policy, even with the increase of documents size or access requests, we show that our model closely costs the same. Case studies are left for future work to insure the completeness of our reference contexts.

Keywords :- XML, XPath, Security, Policy, Granularity.

I. INTRODUCTION

Sharing Extensible Markup Language (XML) documents is becoming widely available within e-commerce applications. The need of access control at a fine-grained levels has pushed developers and researchers to aggressive battles in choosing between cost and efficiency. XML access control granularity is the smallest amount of nodes that can be independently controlled. Therefore; the XML Path (XPath) [1] is the common language to refer to these nodes, and to be used as the object in access control requests.

Despite the fact that XPath expressions are only used as references to actual XML nodes in XML access control policies, current models unfortunately deployed the full contents for dealing with policy constraints. In these regards; current proposals can be categorized into two different solutions. First; there are some models that dealt with XML documents in the main memory to process access control decisions in a timely manner. And second; other approaches labeled nodes in XML documents with their constraints statically to make decisions before deploying the requested data.

Security must not be afterthought, it must be first thought of [2]. Providing means to ease developing access control models takes care of this issue. Access control development should be as simple as the conditional statement (if Boolean then decision). We propose a method that eliminates data deployments in XML access controls. The used XPath references in access requests and policy constraints will be compared to make access decisions.

One problem is that there are different ways to express selecting nodes in XPath. Analyzing XPath expressions is necessary to achieve 100% access control. As will be seen later, one may use the full syntax to request an access to a node, where the policy uses the abbreviated syntax to prevent this access. Matching these two expressions will return false which results in allowing the access to the requested data. From developer's point of view, the analysis will provide the terms; algorithms optimization, time efficiency, and logical development.

The rest of the paper is organized as follows. After we give a general overview on access control and XML access control mechanisms in Section II, we present previous work in Section III. We then define our mechanism based on using reference check between objects, issues, and common functions in using XPath expressions in Section IV. Experimental analysis is discussed in Section V, where future work and conclusion are presented in Section VI.

II. ACCESS CONTROL

The problem of access control is not relatively new, it has been studied within different applications such as Web Services (WS), Relational Database Management Systems (RDBMS), XML documents, and so on. In general, access control security systems enforce protection by calling documents called policies. A Policy contains set of rules that the system must meet before allowing users to access any critical information. Figure 1 is an example that demonstrates a policy for a random system.

Publication History
Manuscript Received : 3 November 2015
Manuscript Accepted : 20 November 2015
Revision Received : 10 December 2015
Manuscript Published : 31 December 2015

Rules are also referred to as constraints. An access control constraint is generally on the form (Subject, Object, Effect, Action) [3] where the Subject is the user or role to whom access is applied to, the Object is the reference expression that denotes the set of files concerned by the rule, the Effect specifies whether the operation is allowed or denied, and the

1) A lecturer can read any student files.
2) A lecturer can only update his students' grades.
3) A student can only read his own file.
4) A student can not update any student files.

Figure 1. Example of a policy

1) (Lecturer, "//student", +, Read).
2) (Lecturer, "//student[class/lecturerId=userName]/grades", +, Update).
3) (Student, "//student[id=userName]/info", +, Read).
4) (Student, "//student", -, Update).

Figure 2. Expressing Figure 1 example by access control policy

Action specifies the operation under concern. The example illustrated in figure 1 can be expressed by access control policy shown in figure 2.

Remember that these examples are general and each security system may express and implement more complicated cases. Our model represents an access control that restricts access to XML documents by enforcing policies specified in section II-A. More complex cases that were implemented in previous work as demonstrated in section III can also be extended using our approach.

A. XML Access Control

The major difference between XML access control and previous applications of access control is that XML always deals with fine-grained constraints where access is restricted to the smallest unit contained in the document. Indeed, a clean model for dynamic access control with granularity control is needed to allow XML documents to link against arbitrary XML chunks [4]. Using XPath expressions allows a finer level of granularity where applications control access to the tag level of XML documents.

As will be seen in section III, many proposals cover access control to XML documents. Our proposal is not intended to change the way constraints are handled in these approaches, but it suggests a way to use references to objects to improve the efficiency of their behaviors, and to reduce the cost associated with results when decisions are made. The following summarizes some specifications that are covered by these proposals:

- Two-valued versus three-valued policies: Some approaches consider the Permit/Deny decisions while others consider Permit/Deny/Undetermined decisions when a request does not match any rule within the policy.

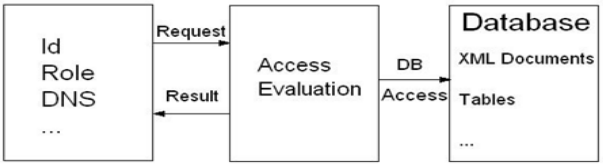

Figure 3. General architecture of In-Memory approach

Handling domain constraints: Most of these proposals accept User IDs and Roles as subjects within rules, and some extend subjects to domain names (DNS), internet protocols (IP), using the wild card (*), and so on.

- Support policies specified in other XML languages: It is important for a system to support policies ex- pressed in rich policy languages such as the Extensible Access Control Markup Language (XACML) with features combining algorithms [5].

B. The XML Path Language (XPath)

XPath is the result of an effort to provide a common syntax and semantics for functionality shared between XSL Transformations (XSLT) and XPointer [1]. In XML access control mechanisms, XPath expressions are used in selecting requests' resources as well as specifying rules' objects. The constraint; (**Role:Doctor,//record/patient,+,Read**); for example indicates that the record Patient in an XML document can be read by a doctor. XPath analysis will be provided in Section IV.

III. PREVIOUS WORK

Traditional access control models, such as access matrix, mandatory access control (MAC), discretionary access control (DAC), and role-based access control (RBAC), have been proposed to meet different application requirements for long [6], [7], [8].

In the past ten years, XML access control has reached its peak of development. Proposals for XML security battled in providing finer-grained access levels, in providing logical definitions of deny, permit, and undetermined decisions, in supporting other XML languages such as the Extensible Stylesheet Language Transformation (XSLT) and XACML, in processing access control queries efficiently, etc.

Access control models can be categorized into two different types in regards to efficiency and cost. These are illustrated in the following two sections.

A. In-Memory Processing

Where XML trees are placed in-memory in query processing enabling quick runtime processing beside the flexibility of visiting nodes as many times as needed. Figure 3 shows the general architecture of In-Memory approach.

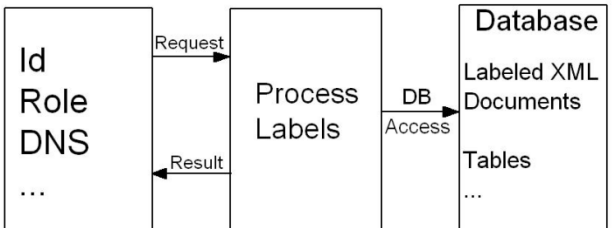

Figure 4. General architecture of In-Document approach

A model [4] that covers access control attributes by using the Java DOM interface to evaluate request against policies was proposed. Provisional authorization [9] provided XML with element-wise access control mechanism namely XML Access Control Language (XACL). By exploiting the opportunities offered by XML [10], a fine-grained access control defined a model for restricting access to Web documents that takes into consideration the semi-structured organization of data and their semantics.

OASIS Extensible Access Control Markup Language (XACML) [11] uses a Policy Decision Point (PDP) that evaluates statement requests in accordance to the semantics specified in the given policy. XACML project was also continued [12] aiming in creating portable and standard way of describing access control entities beside providing a mechanism that offers much finer granular control than simply denying or granting access.

Other approaches used similar mechanisms to deliver access control for XML documents. Role Based Access Control (RBAC) for example is applied on XML properties and carried out an extended access control method [13]. XML-Based Declarative Access Control [14] presents an engine (Xplorer) to interpret control rules and provide se-cure searching and browsing for XML repositories. Access Control for XML Documents [15] proposed a flexible and powerful model that can effectively protect XML documents from unauthorized attempts and malicious damages. And a Distributed Push-Based XML Access Control Model for Better Scalability [16] presented a scaling strategy that distributes the increased system workloads to different servers.

B. In-Document Processing

Labeling XML nodes; Static Method; with policies' contents is another way of evaluating access requests. The rationale for this approach is defining an XML markup for a set of security elements describing the protection requirements of XML documents. The security markup can be used to provide both instance and schema-level authorizations at the granularity of XML elements. Figure 4 shows the general architecture of In-Document approach.
XML access control using static analysis [17] uses automata to read the marked up nodes in databases for representing and comparing queries with access control.

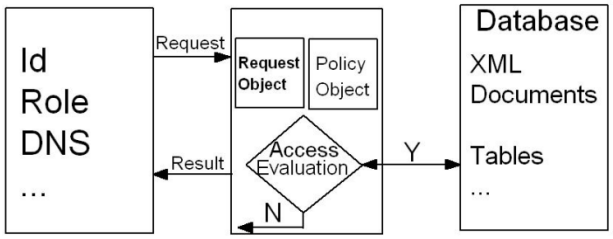

Figure 5. General architecture of Reference Check Model

policies and schemas. Active XML (AXML) is a declarative framework that harnesses the emerging standards for security integration [18] which is based on embedding calls to web services, parts of the XML data are given explicitly while other parts consist of calls to web services by placing them within the <sc> ...</sc> elements. Proposals [19], [5], [20] encrypted parts of documents to be retrieved by users who have public keys along with indexing methods to efficiently support multiple level security model.

Tree based access control mechanism for XML databases [21] introduced a tree structure; Policy Matching Tree (PMT); that pre-processes policies into trees to be matched with access requests. This approach performs significantly better than the previous approaches, it did not however cover the different expressions that XPath uses where the same node may be selected in multiple ways as will be seen in Section IV. Another issue is the limitation of its use by treating access control policies as a whole structure in constructing tress from objects, effects, and actions for each user.

IV. REFERENCE CHECK IN XML ACCESS CONTROL

We propose a mechanism that checks object references used in access requests against object references used in policy constraints eliminating database prefetching and labeling to make access decisions. Previous approaches fully deploy the actual data to make access decisions despite the fact that only references are used within requests and policy rules. Figure 5 shows the general architecture of our proposed model.

To achieve full security coverage, our model is based on analyzing the reference language used in requests and constraints. In our case, we will produce a function that recognize the full syntax and the abbreviated syntax in XPath, and convert them to a set of common patterns.

XPath uses different syntax to express selecting the same node in a tree. For example; the full syntax record::child::patient and the abbreviated syntax record/patient result in selecting group of the Patient children in a Record node within XML documents. More information and examples will also be useful by reading the XPath tutorial in http://www.w3schools.com/Xpath/.

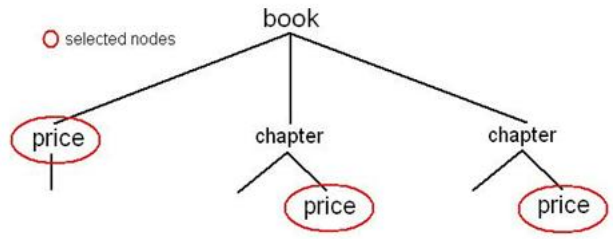

Figure 6. Selecting any node example

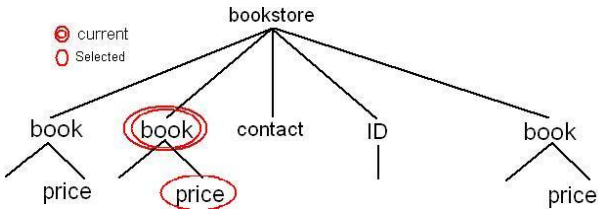

Figure 7. Selecting current node example

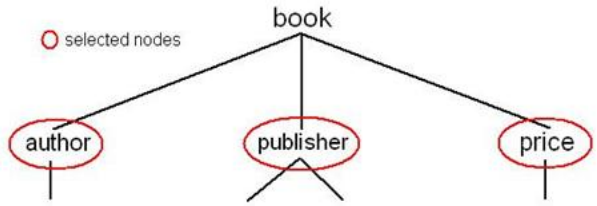

Figure 8. Selecting all nodes example

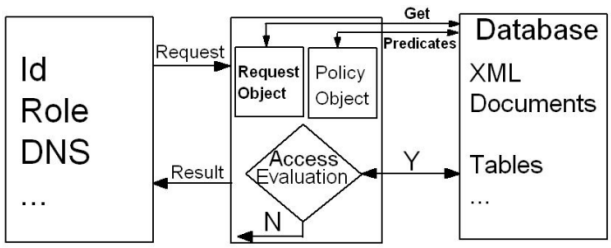

Figure 9. General architecture of Reference Check with Predicates

A. Expressions Conversion in XPath

XPath expressions that are mostly used will be analyzed enabling their conversion to sets of patterns. The general mechanism will be provided.

1) The Absolute Location (NodeName) and (/): The simplest, easiest, and straight forward expression input is the absolute expression where the first axis starts from the root of the tree. The pattern will be constructed by writing node names simultaneously with navigating through the XPath expression. For instance; /bookstore/book and book-store::child::book produce the set [“/bookstore/book”].

2) The Any Node (descendent) and (//): Descendent and double slash // are used in XPath to select any node name under the specified path. For example; book::descendent::price and /book//price are used to select the price children, children’s children, etc of the book root. Figure 6 shows the selected nodes to be returned. To implement this notation, the wild card (*) will be used to indicate that any pattern in this location is a match. The set, [“/book/*/price”] is produced.

Note that the wild card located in the set is different from the wild card located in the original XPath expression which will be seen in the next sections. The (*) input indicates all nodes in XPath, where the (*) output indicates any string at the same location.

3) The Current Node (.): Starting the full path with the relative axis and the abbreviated path that starts with the dot symbol (.) are equivalent, they denote navigating from the current node. The two expressions child::price and ./price select the child price of the current node. The above expressions result in producing the set [“./price”]. Figure 7 assumes the Book node is the current node, and it shows the selected node.

4) The All Nodes (sibling) and (*): The (sibling) axis and the wild card (*) are used in XPath expressing that all nodes are to be selected in an XML document. Implementing all nodes would use the same technique that is used in any node selection. Figure 8 is an example of the two expressions sibling::book and ./book/*. The set [“./book/*”] is produced.

5) The Parent Node (..): The axis (Parent) and the expression (..) indicate selecting the parent of the named node. The two expressions parent::price and ../price select price’s parent, and the set [“../price”] is produced.

6) The Multiple Expressions (|): XPath expressions may use the symbol (|) to select multiple nodes that are not related. The expression sibling::book | book::descendent::price and the expression book/* | book//price produce the set [“/book/*”,“/book/*/price”]

7) Predicates and Conditions: Predicates are used to find a specific node, a node that contains a specific value, or a node that conforms to a giving condition. XPath places predicates and conditions in between the square brackets ([]) to satisfy this requirement. In our model, predicates are the only type that direct the control to access the actual XML documents in databases to get values based on the specified condition. While this may cause a delay, the obtained values are too little comparing with the other approaches that have been studied. Figure 9 extends the giving graph shown in Figure 5 by adding the predicates access to both of the request and policy objects.

Our XPath analysis covered most of the used expressions when selecting nodes in XML trees. To achieve full security coverage and to avoid permission leakage, we need more

case studies to analyze, and to insure there is no expressions are left without being converted. In our future work, these

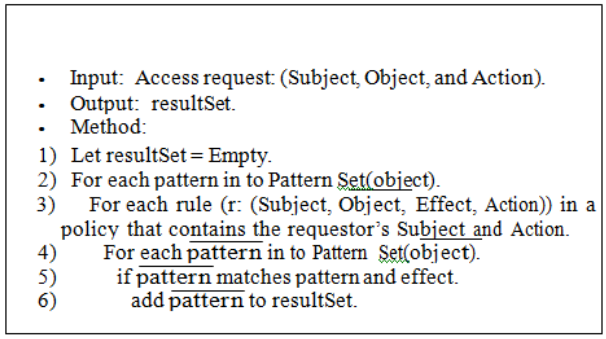

Figure 10. Access control algorithm

case studies will be collected, and our system will be tested in a realtime infrastructure to achieve these requirements.

B. Access Control Algorithm

When access is requested to an object by a user, the process passes certain information to the access control model in order to evaluate that access permission or denial. Access control algorithm is illustrated in figure 10.

The object contained in the input is an XPath expression that will be converted to a set of patterns as illustrated in our XPath analysis.

The rule r in a policy is a tuple (Subject, Object, Effect, Action) where Object is an XPath expression that the rule is specifying, Effect is a Boolean value; True indicates that Subject is allowed to perform Action on Object where false indicates otherwise.

Processing the request results on creating a new set of patterns; resultSet; that the subject is allowed to access. Please note that any implementation may refine its outputs depending on the returned patterns. One may say if resultSet is empty, the subject is not allowed to proceed with his request. And another may return new XML document that has the allowed nodes by calling prune(XML-document) method which deletes the nodes that are not allowed to be accessed from the original document.

V. PERFORM ANCE ANALYSIS

Based on our understanding to the reviewed literature, we constructed three simple models to be compared with our XPath Reference Check. The first is using Policy Matching Tree (PMT) where references only use the absolute XPath expressions to achieve security results. The second is using in-memory approach where the full XML documents were deployed. And the third is using in-document mechanism where user names and constraints are written within our documents.

Our experiment was done using Pentium 2.53GHz and 1GB of RAM. It is divided into the following four stages to prove that our model closely cost the same even when policy and document sizes increase.

Table I
AVERAGE PROCESSING TIME PER REQUEST (MS)

Stage	Ref. Check	PMT	In-Mem	In-Doc
1	0.0071	0.0062	90%	
2	0.0076	0.0077	95%	
3	0.0082	0.0086	87%	
4	0.0084	0.0083	93%	

1) We used a policy consisting of five rules, XML document with four levels and a total of 614 nodes, ten requests without using predicates, and ten requests using predicates.
2) We used a policy consisting of five rules, XML document with seven levels and a total of 3060 nodes, ten requests without using predicates, and ten requests using predicates.
3) We used a policy consisting of fifty rules, XML document with four levels and a total of 614 nodes, ten requests without using predicates, and ten requests using predicates.
4) We used a policy consisting of fifty rules, XML document with seven levels and a total of 3060 nodes, ten requests without using predicates, and ten requests using predicates.

Table I shows the average request's processing time in milliseconds for each stage. PMT performance was as good as our model, it did not however detect most of the expressions where the full and abbreviated syntax were used.

VI. CONCLUSION

This paper introduced another access control mechanism by using reference check between requests and constraints objects eliminating the expensive runtime needed to access documents in databases. We focus on these references' analysis by converting them to sets of common patterns. From developers point of view, these analysis will provide the terms; algorithms optimization, time efficiency, and logical development.

Another advantage is that the mechanism can be used within other access control models such as RBAC and RDBMS. We plan to gain more skills into dealing with these controls, and analyze the references used to cache and control the required objects.

More aspects are also left for future work as this paper could not cover. Case studies and expressions analysis for example will be conducted to insure accurate decisions, and to achieve full security coverage. Finally, policy analysis to overcome repeated objects and conflicts when policies are updated will enable the integration of our model into realtime systems.

REFERENCES

[1] J.Clark and S.DeRose, "Xml path language (xpath) version 1.0," November 1999.[Online]. Available: http://www.w3.org/TR/xpath

[2] C. Steel, R. Nagappan, and R. Lai, Core security patterns best practices and strategies for J2EE, Web services and identity management. Upper Saddle River, NJ, USA: Prentice Hall Professional Technical Reference, 2005.

[3] F. Irini and M. Sebastian, "Formalizing xml access control for update operations," SACMAT07, pp. 169–174, 2007.

[4] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati, "Design and implementation of an access control processor for xml documents," Computer Networks, vol. 33, no. 1-6, pp. 59–75, 2000.

[5] P. Rao, D. Lin, E. Bertino, N. Li, and J. Lobo, "An algebra for fine-grained integration of xacml policies," Purdue University, West Lafayette, IN, USA, Tech. Rep., 2008.

[6] N. Li and M. V. Tripunitara, "Security analysis in role-based access control," ACM Trans. Inf. Syst. Secur., vol. 9, no. 4, pp. 391–420, 2006.

[7] X. Zhang, R. Sandhu, and F. Parisi-Presicce, "Safety analysis of usage control authorization models." New York, NY, USA: ACM, 2006.

[8] S. Jha, N. Li, M. Tripunitara, Q. Wang, and W. Winsborough, "Towards formal verification of role-based access control policies," Dependable and Secure Computing, IEEE Trans-actions, vol. 5, no. 4, pp. 242–255, 2008.

[9] M. Kudo and S. Hada, "Xml document security based on provisional authorization," in CCS '00: Proceedings of the 7th ACM conference on Computer and communications security. New York, NY, USA: ACM, 2000, pp. 87–96.

[10] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati, "A fine-grained access control system for xml documents," ACM Trans. Inf. Syst. Secur., vol. 5, no. 2, pp.
[11] 169–202, 2002.

[12] S. Godik and T. Moses, "Oasis extensible access control 2 markup language (xacml)," cs-xacml-specification-1.0, 2008.

[13] M.Verma,"Xml security:Control in- formation access with xacml," 2004, http://www.ibm.com/developerworks/xml/library/x-xacml/.

[14] X. Meng and D. Luo, "An extended role based access control method for xml documents," Wuhan University Journal of Natural Sciences, vol. 9, no. 5, pp. 740–744, 2004.

[15] R.Steele,W Gardner, T.S.Dillon, and A.Erradi,"Xml-based declarative access control,"vol.3381,pp.310–319,2005, http://www.springerlink.com/content/66rq8tuwhkfdgm1q/.

[16] Y. Xiao, B. Luo, and D. Lee, "Access control for xml document," vol. 5027, pp. 621–630, 2008.

[17] W. Halboob, A. Mamat, and R. Mahmud, "A distributed push-based xml access control model for better scalability," Distributed Framework and Applications, pp. 20–26, 2008.

[18] M. Murata, A. Tozawa, M. Kudo, and S. Hada, "Xml access control using static analysis," ACM Trans. Inf. Syst. Secur., vol. 9, no. 3, pp. 292–324, 2006.

[19] S. Abiteboul, O. Benjelloun, and T. Milo, "The active xml project: an overview," vol. 17, no. 5, pp. 1019–1040, 2007.

[20] H.-K. Ko, M.-J. Kim, and S. Lee, "On the efficiency of secure xml broadcasting," Information Sciences, vol. 177, pp. 5505–5521, 2007.

[21] Y.Xiao,B.Luo,and D. Lee, "Security-conscious xml indexing," vol.4443,pp. 949–954,2007, http://www.springerlink.com/content/372088578m01883u/.

[22] N.Qi and M.Kudo,"Tree-based access control mechanism for xml databases," 2005, http://www.ieice.org/d˜ e/DEWS/DEWS2005/procs/papers/5A- o1.pdf.