RDF Metadata for XML Access Control

Vaibhav Gowadia
gowadia@cse.sc.edu

Csilla Farkas
farkas@cse.sc.edu

Information Security Lab
Department of Computer Science and Engineering
University of South Carolina
Columbia, SC 29205

ABSTRACT

In this paper we present an access control framework that provides flexible security granularity for XML documents. RDF statements are used to represent security objects and to express security policy. The concepts of simple security object and association security object are defined. Our model allows to express and enforce access control on XML trees and their associations. Access control rules, corresponding to $(s, o, \pm a)$ triples, are represented as RDF statements with properties access type, user, and object. A history file is maintained for each user that allows decision-making using temporal data.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration - security, integrity and protection; D.4.6 [**Operating Systems**]: Security and Protection - Access controls

Keywords

Access control, association objects, flexible security granularity, RDF metadata, RXACL, tree extension, XML security

1. INTRODUCTION

Web-based applications increasingly rely on eXtensible Markup Language (XML) [11]. Numerous domains, like health care and electronic commerce, use semi-structured and XML data to provide inter-operation among different systems. Efforts, to view semi-structured and XML data from the database perspective, have emerged recently [7, 8, 9]. It is necessary to develop access control models that can efficiently represent application specific security requirements for XML data.

Several XML access control models have been developed recently [4, 5, 6, 14, 16, 21, 26, 27]. They are based on traditional access control lists and provide extensions to XML syntax. Existing models allow node-level security granularity by assigning access restrictions to the nodes and links of XML documents. However, none of these models provide access control for data associations.

To illustrate the need of access control for data associations we present an example in the medical domain. Assume that XML format is used for storing patient records. The DTD for patients' health records is shown in Figure 1. Alice, who is an intern at the hospital, needs limited access to the database. Her duties involve two main tasks. First, Alice needs to contact the patients for collecting feedback about their treatments, thus Alice is allowed to read <name> and <PhoneNumber> elements. Second, Alice needs to prepare statistical reports based on Age, Race and Diagnosis of the patients. For this, Alice is needs to access both contact and diagnosis information of all patients. However, Alice should not be able to access data about the name and diagnosis of the patients together. The functionality requirements of Alice's work and the security restrictions cannot be both satisfied using traditional access control list-based methods. It is possible to combine results from multiple queries to disclose disallowed data. For example, Alice may use unique identifiers, like database keys, to associate data from different queries and disclose patient diagnosis information.

Existing XML access control models do not consider data associations. For example, XACML [28, 27] allows use of conditions, obligations or provisions for accessing data. Although, XACML could be extended with additional security modules for keeping track of history and combining answers to express association-based restrictions, this approach would be cumbersome. Access control models that provide intuitive and efficient representation of data associations need to be developed.

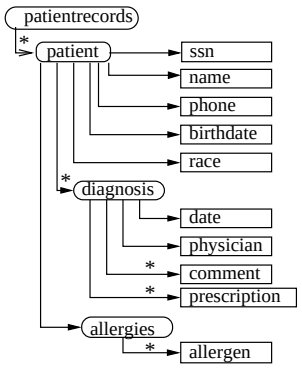
In this paper we present an access control model that provides flexible access control granularity by allowing security classification of XML nodes and subtrees (simple security objects), and associations among nodes (association security objects). Intuitively, an association security object is an XML subtree that is disallowed to be accessed by a user, while all of its constituent elements are permitted separately. We assume that the answers to XML data requests form a forest of trees. These trees may be merged based on the XML key constraints that hold on the original XML document. To detect violations of security policy we check whether the merged trees contain any disallowed object.

We use Resource Description Framework (RDF) [10] statements to express access control requirements. Use of RDF adds semantic meaning to the statements, increasing the

ACM Workshop on XML Security, October 31, 2003, Fairfax VA, USA.

```
<!DOCTYPE patientrecords[
  <!ELEMENT patientrecords(patient*)>
  <!ELEMENT patient(ssn,name,phone,age,race,diagnosis*)>
  <!ELEMENT ssn(#PCDATA)>
  <!ELEMENT name(#PCDATA)>
  <!ELEMENT phone(#PCDATA)>
  <!ELEMENT birthdate(#PCDATA)>
  <!ELEMENT race(#PCDATA)>
  <!ELEMENT diagnosis(date,physician,comment*,presecription*)>
  <!ELEMENT date(#PCDATA)>
  <!ELEMENT physician(#PCDATA)>
  <!ELEMENT comment(#PCDATA)>
  <!ELEMENT prescription(#PCDATA)>
  <!ELEMENT allergies(allergen*)>
  <!ELEMENT allergen(#PCDATA)>
]>
```

(a)



(b)

Figure 1: (a) An example of a DTD (b) Tree representation of DTD

flexibility and expressiveness of our model. In particular, it provides an intuitive method to express association-based constraints. Our ongoing work extends the current results with conflict resolution and limited inference control.

The organization of the paper is as follows. In Section 2 we discuss the proposed security architecture. Section 3 contains the specifications of the RDF based XML Access Control Language(RXACL). This section also defines the association security object. In Section 4 we describe the algorithms for security check and tree extension (partial merging). In Section 5 we give a brief overview of related work. Finally in section 6 we conclude and suggest future research work.

2. RDF-BASED XML ACCESS CONTROL ARCHITECTURE

Figure 2 shows the RXACL architecture for enforcing access control for XML documents. The corresponding algorithm is given in Figure 3. The architecture contains three main components: 1. Query engine, 2. Access Control, and 3. User history. The XML query engine is responsible for generating responses to user's requests. This component uses an existing XML query engine, development of such an engine is outside of the scope of this paper. The access control component evaluates the authorization of the user's requests based on the security policy and data previously released to the user. The history component keeps track of data released to each user.

When a data request is submitted to an XML query engine, the result and the query are sent to be checked for security violations. If the security policy is not violated then the result is combined with data in the user's history file. The security check module then checks for violations in the newly extended trees. If there are no violations detected, the query answer is returned to the user and the history file is updated. If a security violation is detected at any phase, the query is rejected.

An efficient way to check security violations is to work at schema-level, without considering the actual data. However, schema-level detection, when considering associations, may result in incorrect denial of data requests. Data-level access control will allow higher data availability than schema-level but increases computational cost. Here, we focus on data-level detection.

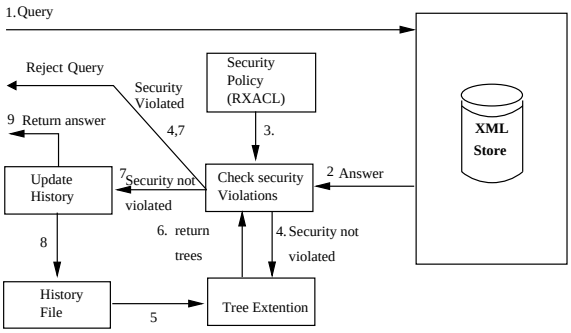


Figure 2: RXACL architecture for enforcing XML access control

3. DEFINITIONS

This section describes definitions and specification of RXACL. We start with a formal description of node-labeled trees that we use to represent XML instances, DTDs, and security objects. To define XML trees, path, and key constraints, we use formalism similar to the one described in Buneman et al. [7, 8, 9].

DEFINITION 3.1. (Labeled tree)
A *labeled tree*, or a tree, is defined recursively as follows:

1. The empty set $\{\}$ is a tree, called the empty tree.
2. A single *node* $\{n\}$ is a tree.
3. If $t_1, t_2, \ldots, t_k$ are trees, then $\{n \rightarrow \{t_1, t_2, \ldots, t_k\}\}$ is a tree. In this case we say that $\{n \rightarrow \{t_1, t_2, \ldots, t_k\}\}$ represents a tree with root node n that has outgoing edges to subtrees $t_1, t_2, \ldots, t_k$.

The nodes of the trees are labeled. Labels may be actual facts, node variables (corresponding to any node value), or path variables (corresponding to any path). Constants correspond to element, attribute and text nodes.

Algorithm 1:	RXACL Security Architecture
INPUT	1. User's query Q 2. XML document 3. User's id u 4. Security policy POL 5. User's history-file $Hist$ (Forest of XML trees received by the user) 6. $\mathcal{K}$, set of candidate keys that hold on the original document
OUTPUT	Answer to Q and update of the user's history-file or refusal of Q
METHOD	1. Generate answer to the user query, call it $Answ$ 2. Use Algorithm 2 to verify that $Answ$ does not contain any subtrees not permitted to the user 3. **if** disallowed subtree is detected (i.e., Algorithm 2 returns True) **then** reject Q and **quit** **else** (a) Use key constraints to extend ($\cup_E$) $Hist$ with $Answ$ (partially merge new answer and the user's history file) (b) For each modified tree in $Answ \cup_E Hist$ check for existence of disallowed subtree using Algorithm 2 (c) **if** disallowed subtree is detected (i.e., Algorithm 2 returns True) **then** reject Q and **quit** 4. Answer Q and update $Hist$ as $Hist = Answ \cup_E Hist$

Figure 3: RXACL Security Architecture

A labeled tree is called a *ground tree* if all of its nodes are constants.

XML instances and DTDs are ground trees. Node and path variables are important for defining security objects and expressing general queries over an XML database.

DEFINITION 3.2. (Path-expression)
A *path-expression* $P = \{e_1 \rightarrow \{e_2 \rightarrow \{\ldots \rightarrow \{e_k\}\}\}\}$ is a path, where e_i is a node with label that is either a constant, a node variable, or a path variable.
An *absolute path-expression* with respect to a tree T, denoted as P^a, is a path-expression $\{e_1 \rightarrow \{e_2 \rightarrow \{\ldots \rightarrow \{e_k\}\}\}\}$, where e_1 corresponds to the root node of T.
A *relative path-expression* originating from a node n_i, denoted as P^r, is a path-expression $\{e_1 \rightarrow \{e_2 \rightarrow \{\ldots \rightarrow \{e_k\}\}\}\}$, where e_1 corresponds to n_i.
A *ground path-expression* (or ground path) is a path expression where all e_i's are constants.

We assume that a path traverses only downward in an XML tree. Path-expressions are used to identify nodes and subtrees within a tree. For this, we need the notion of path containment within a tree.

DEFINITION 3.3. (Path Containment)
Let $P = \{e_1 \rightarrow \{e_2 \rightarrow \{\ldots \rightarrow \{e_k\}\}\}\}$ be a path-expression and T a ground tree. We say that P is contained in T iff there exists a mapping ν from the symbols of P to the constants and ground paths of T, such that $\nu(P) = \{\nu(e_1) \rightarrow \{\nu(e_2) \rightarrow \{\ldots \rightarrow \{\nu(e_k)\}\}\}\}$ is a ground path in T. We require that ν maps
1. Every constant $e_i \in P$ to the same constant,
2. Every node variable $e_i \in P$ to a single constant in T, and
3. Every path variable $e_i \in P$ to a ground path $\{e'_{j_1} \rightarrow \{\ldots \rightarrow \{e'_{j_l}\}\}\}$ in T.

Path-expressions are used to express general queries and protection objects. For security considerations, it is often required to decide whether a path-expression is contained at least once in the database or in the query answer. For this, we say that a ground-tree satisfies a path-expression if there exists at least one ν from the path-expression to the ground tree.

DEFINITION 3.4. (Satisfaction of Path-Expression)
Let T denote a ground tree, and P denote a path-expression. We say that T *satisfies* a path expression P iff P is contained in T. The set of mappings $\nu_1, \nu_2, \ldots, \nu_m$ that maps P to ground paths $P'_1, P'_2, \ldots, P'_m$ in T are called satisfying mappings. A ground tree T satisfies a set $\mathcal{P} = \{P_1, \ldots, P_k\}$ of path expressions, iff T satisfies all path-expressions P_i $(i = 1, \ldots, k)$ in $\mathcal{P}$.

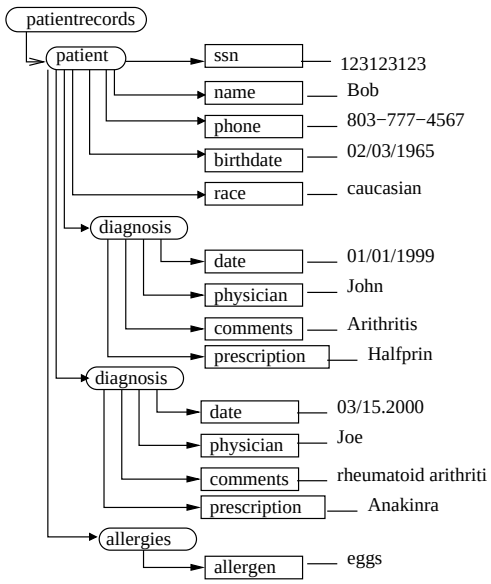
Answers for separate queries on XML documents form a forest of trees. We assume, that these trees can be *extended* (partially merged) only if there are key constraints that allow such an extension. The tree extension operation is defined later in Section 4. Intuitively, an XML key means that there are no two paths in the XML documents that coincide on the same values and positions. For the combination of history information and result set, keys are especially important, because if two distinct paths agree on all values over a key, then their subtrees can be extended. XML constraints and keys have been studied recently by the research group of Buneman and Fan in [20, 18, 17, 9, 8, 19].

DEFINITION 3.5. (XML Key)
Let T denote an XML tree. We say that $K = (P^a, \{p^r_1, \ldots, p^r_l\})$ is a key of T iff

1. T satisfies the absolute path P^a, and

```
<?xml version="1.0">
<!DOCTYPE patientrecords SYSTEM "patient.dtd">
<patient>
 <ssn>123123123</ssn>
 <name>Bob</name>
 <phone>803−777−4567</phone>
 <birthdate>02/03/1965</birthdate>
 <race>caucasian</race>
 <diagnosis>
  <date>01/01/1999</date>
  <physician>John</physician>
  <comment>Arithritis</comment>
  <prescription>Halfprin</prescription>
 </diagnosis>
 <diagnosis>
  <date>03/15/2000</date>
  <physician>Joe</physician>
  <comment>rheumaoid arithritis</comment>
  <prescription>Anakinra</prescription>
 </diagnosis>
 <allergies>
  <allergen>eggs</allergen>
 </allergies
</patient>
```

(a)



(b)

Figure 4: (a) Example XML document (b) Tree representation of XML document

2. There are no two distinct nodes e and e', no mappings ν and ν' where $\nu(P^a) = \{\ldots \rightarrow e\}$ and $\nu'(P^a) = \{\ldots \rightarrow e'\}$, such that for all $i \in 1, \ldots, l$ $\{e \rightarrow \{\nu(p_i^r)\}\}$ satisfies $\{e' \rightarrow \{\nu'(p_i^r)\}\}$ $(i = 1, \ldots, l)$ and vice versa.

3.1 RXACL Specifications

We recommend a security model that uses RDF statements to express security access control rules. An *RDF statement* is a statement describing a resource that can be identified by Universal Resource Identifier (URI). An RDF statement comprises of three parts, *subject* (resource), *predicate* (property), and *object* (value). The object of a statement is the value of the predicate. The object may be a literal or another resource.

There are current efforts [26, 27, 28] to use XML itself to express access control requirements on XML documents. While XML provides syntactic constructs to represent data, RDF provides additional capability for describing data semantics [2]. This means that the properties of the RDF resources can be clearly expressed and relationships defined. However, using XML only, it is not possible to infer relations between different elements and attributes without help from external mechanisms. Also, RDF provides syntax independent representation of data semantics.

In our model, access control policy is specified by a set of RDF statements. The use of RDF allows development of semantic aware security policies. For example, a security policy with semantics may be used to detect inconsistent policies as described by Koch et al. [25]. Also, RDF seems promising to support syntax-independent policy representation, as well as express privacy and trust requirements [24]. To the author's best knowledge, our work is the first, to attempt to develop a formal access control language using RDF. Such a language seems promising to develop security policies at a higher abstraction level.

In this paper we show how RXACL can be used to represent the Discretionary Access Control model [1] with flexible security granularity. We plan to extend our framework to incorporate role-based and multilevel access control modules.

We define access control rules as a triple $(s, o, \pm a)$, where, s is the subject, o is a security object, and a is signed access type. The rule means that subject s is given access $\pm a$ to object o.

We use a closed security policy, where all permitted accesses must be defined. We assume that non-existence of a positive authorization on an object implies that the user is disallowed to access the object, resulting in a negative authorization. Explicit denial of access can also be stated in the form $(s, o, -a)$. The domain of all objects comprises of all XML nodes and declared associations; i.e., association objects explicitly defined for the subjects.

Before defining the RDF representation of the security objects, we need to discuss the granularity of these objects. Intuitively, we want to ensure that a user is able to access all authorized information, but nothing more. In the Introduction, we presented an example, showing that assigning security labels to each XML node may limit data availability. We propose a model, where protection objects correspond to simple XML nodes or the association among XML nodes. More specifically, we define the security objects as follows:

DEFINITION 3.6. (Protection Object)
Protection objects are node-labeled trees.
A *simple security object* o is a tree, where all distinct subtrees $t_1, t_2, \ldots, t_k$ of o have the same access permission as o. That is, for every subtree $t_i \in o$, $(s, t_i, a) = (s, t_i|t_j, a)$, $i, j = 1, \ldots, k$ *and* $t_i \neq t_j$, where $t_i|t_j$ represents the case when t_i

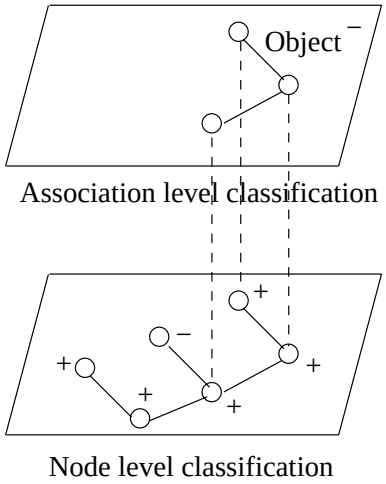


Figure 5: Two-layered access control model.

and t_j are together in the same tree. Simple security objects are equivalent to node-level security classification.
An *association security object* o is a tree, where at least one subject s does not have the permission to access o, while he/she is allowed to access all proper subtrees of o. That is, for a subject s and every proper subtree $t_i \in o$ $(i = 1, \ldots, k)$, access permission $(s, t_i, +a)$ can be derived, while s is denied access a to o, that is $(s, o, -a)$.

Association objects cannot be expressed at node-level, and represent a new layer (association-level) for defining access control. Note that the nodes contained in an explicitly defined association have two classifications assigned to them. First, they are classified as simple security object, resulting in a node-level assignment. Second, they are classified as part of an association, resulting in an association-level assignment. See Figure 5 for an illustration.

Consider again our initial example. We want to ensure that a patient's name, phone number and birth date are public, while his/her diagnosis is confidential when released together with the patient's name. These requirements can be represented by giving permission to access the nodes <patient>, <name>, <phone>, <birthdate>, and < diagnosis >. Paths originating from <patient> node to the above nodes are also public when released individually. Subtrees, formed by paths ({ patient $\rightarrow$ {*phone*}}, {*patient* $\rightarrow$ {*birthdate*}}, [{*patient* $\rightarrow$ {*name*}} OR { patient $\rightarrow$ {diagnosis}}]) are not confidential and correspond to simple security objects. Of course, we do not need to declare access permissions for all possible subtrees. It is sufficient and desired to generate access rules for each exclusive maximal subtree. All subtrees of a simple security object have same permission as the object itself (c.f. Definition 3.6). Declaration of access permissions for these subtrees will be redundant.

To represent an association object, consider the paths {*patient* $\rightarrow$ {*name*}} and {*patient* $\rightarrow$ {*diagnosis*}}. Access to these paths is permitted when they are accessed separately but the subtree {*patient* $\rightarrow$ {*name*, *diagnosis*}} is confidential, therefore it is an association object. Moreover, our definition describes a minimal association object.

Simple security objects have a RDF property *location*. The value of the location is a path expression identifying a node or subtree of the XML tree. Figure 6 shows an example of simple security object represented in RXACL.

An association security object is represented by an absolute path expression and a set of relative path-expressions. The absolute path expression identifies the association root. Each relative path-expression originating from the association root identifies a simple security object. These simple security objects together form the association object. Figure 7 shows an example of association represented using RXACL. Separately, the data contained in the name and diagnosis tags are not sensitive. However, if they are released together or they can be combined, the sensitive association is disclosed.

DEFINITION 3.7. (Association object containment)
Let $\mathcal{O}_1 = (P_1^a, \{p_1^r, \ldots, p_m^r\})$ and $\mathcal{O}_2 = (P_2^a, \{p_1'^r, \ldots, p_n'^r\})$ be association objects. We say that $\mathcal{O}_1$ is contained in $\mathcal{O}_2$ iff $P_1^a = P_2^a$ and for all paths p_i^r $(i = 1, \ldots, m)$ there is a path $p_j'^r$ $(j = 1, \ldots, n)$ such that p_i^r is contained in $p_j'^r$.

In the RXACL model, an access control rule is expressed as a RDF resource with three properties: accesstype, user and object. The RDF Graph for an access control rule is shown in Figure 8. The rule is identified by a URI rxacl:RULE R0, and has rdf:type rxacl:RULE. It states that user Alice is not allowed to read security object described by property rxacl:object with URI rxacl:ASSOCIATION-A0.

Each access request of a subject is evaluated against the predefined access control rules. The following section describes in detail the functionality of the access control module.

4. ACCESS CONTROL MODULE

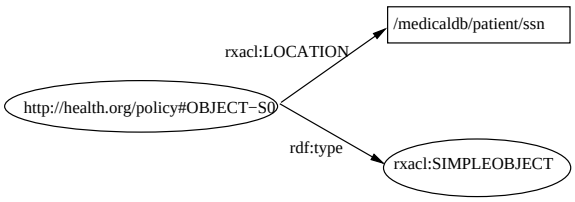
In this section we present algorithms using RXACL for controlling access to XML documents. We assume that answers to user's queries form a forest of trees. Trees can be partially merged only if there exists supporting meta-data that ensures that only those paths are united that correspond to unique paths in the original document. In this work we use XML keys as the basis of combining trees. Ofcourse, other meta-data, like integrity constraints and statistical correlations, may also allow tree unions. However these problems are outside the scope of this document. Partially merging XML trees T_1 and T_2, over a key corresponds to "extending" both T_1 and T_2 over the key paths. We define the *tree-extension* of two XML trees as follows:

DEFINITION 4.1. (Tree-extension)
Let T_1, T_2 be two labeled trees that may contain variables. The *tree-extension* at the presence of keys $K = \{k_1, \ldots, k_n\}$, denoted by $\cup_E$, is defined recursively as follows:

1. $\{\} \cup_E T = T \cup_E \{\} \stackrel{def}{=} T$
2. Let $k_i = (P^a, p_1^r, \ldots, p_l^r)$ $(i = 1, \ldots, n)$ be an XML key in K. Then if there exists satisfying mappings ν_1 and ν_2 from the absolute and relative paths of k_i to T_1 and T_2, such that $\nu_1(P^a) = \nu_2(P^a)$ and $\nu_1(p_j^r) = \nu_2(p_j^r)$, $j = 1, \ldots, l$ then let t_1 and t_2 denote the subtrees rooted at the end nodes of $\nu_1(P^a)$ and $\nu_2(P^a)$, respectively. We extend t_1 and t_2 as follows: Let $(p_1^2, \ldots, p_m^2)$ be the paths of t_2 not satisfied by t_1. Then $t_1 = t_1 \cup_E (p_1^2, \ldots, p_m^2)$. Subtree t_2 is extended similarly.

```
<?xml version='1.0'?>
<rdf:RDF
 xmlns:rdf='http://www.w3.org/1999/02/22−rdf−syntax−ns#'
 xmlns:rxacl='http://www.cse.sc.edu/research/isl/rxacl#'
 >
  <rdf:Description rdf:about='http://health.org/policy#OBJECT−S0'>
    <rxacl:LOCATION>/medicaldb/patient/ssn</rxacl:LOCATION>
    <rdf:type rdf:resource='rxacl:SIMPLEOBJECT'/>
  </rdf:Description>
</rdf:RDF>
```

(a)

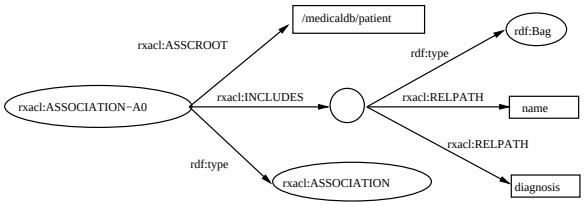


(b)

Figure 6: (a) An example of Simple security object (b) RDF Graph representation of simple object

```
<?xml version='1.0'?>
<rdf:RDF
 xmlns:rdf='http://www.w3.org/1999/02/22−rdf−syntax−ns#'
 xmlns:rxacl='http://www.cse.sc.edu/research/isl/rxacl#'
 >
 <rdf:Description rdf:about='rxacl:ASSOCIATION−A0'>
   <rxacl:ASSCROOT>/medicaldb/patient</rxacl:ASSCROOT>
   <rdf:type rdf:resource='rxacl:ASSOCIATION'/>
   <rxacl:INCLUDES rdf:nodeID='A0'/>
 </rdf:Description>
 <rdf:Description rdf:nodeID='A0'>
   <rdf:type rdf:resource='rdf:Bag'/>
   <rxacl:RELPATH>diagnosis</rxacl:RELPATH>
   <rxacl:RELPATH>name</rxacl:RELPATH>
 </rdf:Description>
</rdf:RDF>
```

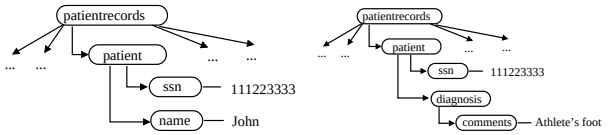
(a)



(b)

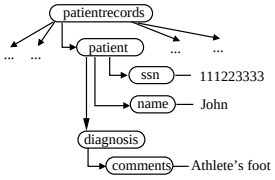
Figure 7: (a) An example of association object (b) RDF Graph representation of association object

We use tree extension to partially merge a new query answer with the user's history file. An example of a tree-extension is shown in Figure 9. In this example, the patient's SSN is the key constraint that holds over the path to the patient node. Since trees T_1 and T_2 coincide on the same SSN values we can conclude that they contain information about the same patient. This information is used to extend the two trees.

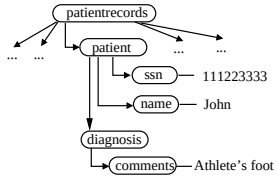


(a)

(b)



(c)



(d)

Figure 9: (a) T_1 (b) T_2 (c) T_1 after extension (d) T_2 after extension

4.1 Access Control Policy

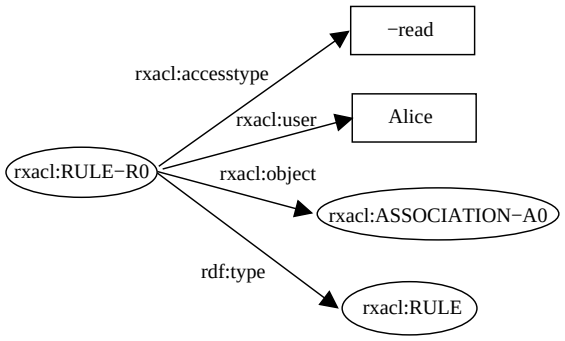
Since we use closed security policy, we assume that all allowed accesses are explicitly defined in security policy. This model is straightforward to check disclosure of simple security objects. Since any two or more XML nodes may form an association object there are exponentially large number of possible association objects. Therefore, we need to limit the domain of all association objects. We assume that the domain of association objects is formed by all declared associations in the security policy. If a subject s is not explicitly permitted access a to a declared association object o, we assume that the subject is disallowed to access o, that is $(s, o, -a)$ is generated. This limited domain of association objects allows our model to operate with complexity dependent on the number of declared associations and not by all possible associations. For a given subject s, we extend the security policy as shown in Figure 10.

The security check module uses the algorithm shown in Figure 12 to detect violations of security policy. The input information for this algorithm includes security policy, an XML tree (query answer), subject s, and user's history information. If the security policy is violated the algorithm returns true otherwise false.

To demonstrate working of Algorithm 2, we will use a security policy that contains the single rule shown in Figure 8. That is the name and the diagnosis of a patient cannot be read by Alice. The input XML tree representing the data requested by user Alice is shown in Figure 11. In the following example we will demonstrate detection of direct security violation.

```
<?xml version='1.0'?>
<rdf:RDF
 xmlns:rdf='http://www.w3.org/1999/02/22−rdf−syntax−ns#'
 xmlns:rxacl='http://www.cse.sc.edu/isl/rxacl#'
 >
  <rdf:Description rdf:about='rxacl:RULE−R0'>
   <rxacl:USER>Alice</rxacl:USER>
   <rxacl:ACCESSTYPE>−read</rxacl:ACCESSTYPE>
   <rxacl:OBJECT rdf:resource='rxacl:ASSOCIATION−A0'/>
   <rdf:type rdf:resource='rxacl:RULE'/>
  </rdf:Description>
</rdf:RDF>
```

(a)



(b)

Figure 8: (a) An example of RXACL rule definition (b) RDF Graph representation of RXACL rule

Procedure 1:	Objects disallowed to subject s
INPUT	1. Security policy POL 2. Subject s
OUTPUT	Set of all disallowed objects $\mathcal{O}$
METHOD	1. Compute $\mathcal{O}_1$, the set of all simple objects denied to subject s: (a) Assume all nodes in DTD tree T are denied access. (b) For all permitted simple objects, mark nodes of simple objects in T. (c) For set of all permitted associations, mark nodes of the associations in T. (d) $\mathcal{O}_1$ is the set of all unmarked nodes in T. 2. Compute $\mathcal{O}_2$, the set of all associations denied to subject s: (a) Let $\mathcal{O}_3$ be set of all associations defined in the security policy. (b) Let $\mathcal{O}_4$ be set of all associations that are explicitly allowed to subject s. (c) Let $\mathcal{O}_5$ be set of all associations that are explicitly denied to subject s. (d) Set of associations for which there are no rules defined for subject s is $\mathcal{O}_6 = \mathcal{O}_3 - \mathcal{O}_4 - \mathcal{O}_5$. (e) Compute all denied associations by including the undefined associations that are not *contained* (c.f. Definition 3.7) in an explicitly permitted association for s: i. Let $\mathcal{O}_2 = \mathcal{O}_5$ ii. For all associations o in $\mathcal{O}_6$ if o is not *contained* in an association in $\mathcal{O}_4$ then add o to $\mathcal{O}_2$, i.e., $\mathcal{O}_2 = \mathcal{O}_2 \cup \{o\}$ 3. All disallowed objects of s are $\mathcal{O} = \mathcal{O}_1 \cup \mathcal{O}_2$

Figure 10: Objects disallowed to subject s

In Step 1, the rule set is evaluated to obtain the list $\mathcal{L}$ of all objects for which Alice does not have clearance. $\mathcal{L}$ consists of the Association shown in Figure 7. Now containment the of association root in the input XML tree t is checked. Path expression /patientrecords/patient is satisfied by tree t. Set T consisting of all *patient* elements is obtained on satisfaction of this path expression. We begin security check for association object assuming that it is contained in tree and security is violated. Now, existence of elements satisfying all relative path expressions is checked. Relative path expressions *name* and *diagnosis* are satisfied by *patient* subtree in t. These elements map to Bob's medical information. After the evaluations, we find that all the relative paths in *association-A0* are contained in T. Security violation is reported. If any of the relative paths in an association security object is not contained in tree T, we know that our assumption about security violation was incorrect and we can start security check for next object. If no security violations are detected by this algorithm, the user's request is permitted, and the history file is updated.

THEOREM 4.1. (Detection of security violations) A security violation is detected by Algorithm 2, iff the XML tree t contains security object o that is not permitted for the user, i.e., either the original security policy does not contain

Algorithm 2:	XML Security Check
INPUT	1. XML tree t 2. Security policy POL 3. Subject s 4. History H
OUTPUT	TRUE of FALSE (If security policy is violated, return True, otherwise return False)

METHOD	**begin**
	1. From POL, retrieve all objects o such that s does not have a clearance for.
	2. Let $\mathcal{L}$ denote the list of these objects.
	3. For each object o in $\mathcal{L}$
	begin (Check existence of object in t)
	if the object o is *simple security object* then
	(a) Check if it is contained in tree t by checking if its path expression is contained in t
	(b) **if** path is found, then return TRUE (security is violated)
	else (the object is an *association security object*)
	if association key expression is contained in tree t then
	(a) Let $\{t_1, \ldots, t_l\}$ be set of subtrees satisfying association key expression
	(b) For each subtree in $\{t_1, \ldots, t_l\}$
	i. For each relative paths $p_1, \ldots, p_k$ of association object
	if p_i is not contained in the tree then
	Check next object (continue step 3)
	endif
	ii. return:TRUE (security is violated)
	else
	Check next object (continue step 3)
	endif
	end
	4. return:FALSE (security is not violated)
	end

Figure 12: XML Tree Security Check

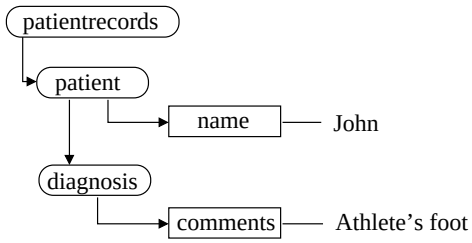


Figure 11: Input XML: Working example for algorithm 2

a rule or a set of rules that yield $(s, o, +a)$, or $(s, o, -a)$ is derivable from the security policy.

Note, we assume the simple conflict resolution strategy that negation takes precedence. That is, if there are two conflicting permissions on an object o at the same abstraction level (i.e., node-level, association level), such that both $(s, o, +a)$ and $(s, o, -a)$ can be derived, we interpret it as s is denied the access a to object o.

Proof Sketch: The proof of theorem 4.1 follows naturally from Procedure 1 to compute the set of disallowed objects for a subject, the definitions of simple and association objects, and Algorithm 2. We assume that the original policy is consistent at each level (node and association levels). First, we show that our procedure to compute the set of disallowed objects will generate rule $(s, o, -a)$, for an object in $\mathcal{O}_1$ or $\mathcal{O}_2$ only if there are no rules in the security policy that would yield $(s, o, +a)$. If the security policy contains a rule that would yield positive authorization for a simple security object, nodes of the simple security object are marked in step 1(b) of the Procedure 1. If the protection object is an association, the nodes the association object in t are marked as permitted in step 1(c) of the Procedure 1.

Next, we show how the XML security check algorithm detects security violations. For this, we use proof by contradiction. Assume, that there is a disallowed object o in the XML tree t that is not detected by Algorithm 2. If the disallowed object is a simple security object, then T must contain path-expression of the simple security object. If o is an association object, then t must satisfy the absolute path expression of the association root and must contain mappings of all relative path expressions in the association to the ground paths of t. Algorithm 2 must have detected these paths and consequently o in step 3(a) or 3(b), this is a contradiction. □

5. RELATED WORK

Several XML access control models have been developed recently [4, 5, 6, 14, 16, 21, 26, 27]. These access models adapt traditional access control lists and provide extensions to XML syntax. Existing models allow node-level security granularity by assigning access restrictions to the nodes and links of XML documents. However, none of these models focus on providing access control for data associations.

The XML Access control model developed by Bertino et al. [5, 6, 3] provides varying protection granularity levels and considers the case when XML documents do not conform to a predefined document type definition (DTD). The access control model proposed can be used for DTD-based and document-based policies. The access control policy is represented using a 5-tuple of subject, object specification, privilege, propagation option and signed access decision. The object specification is a path expression identifying one or more nodes in the XML document and provides granularity at schema level, document level and element level.

The access control model proposed by Damiani et al. [14, 13, 12, 15] defines and enforces access restrictions directly on the XML document structure and content. The authors define a set of security elements used to provide instance level as well as schema level authorizations with the granularity reaching the lower level XML components. The system returns to the requester a partial view of the documents that conforms to data protection requirements. Access control policy stored at each server contains a set of access authorizations (subject, object, action, sign, type). Objects in the XML documents are represented by XPath expressions for the element or attribute.

Gabillon et al. [21] present a similar access control system derived from the same access control list framework. The server stores a set of authorizations in the form (object, subject, access, priority). The priority is used in the conflict resolution procedures.

The above XML access control models specify security constraints as binary decisions for granting or denying access to nodes identified by their object path expression. These models do not provide any mechanism for expressing and enforcing security constraints on associations among elements in the document.

Kudo et al. [26, 27] proposed an access control model that provides provisional authorization [22], security transcoding, and integrated audit trails. The security transcoding is a transformational operation that retrieves secure data from the repository and modifies data in the repository. Their work does not discuss enforcement of security constraints on associations.

Context-based access control has been studied in relational databases by Keefe et al. [23]. They introduced three techniques for enforcing context-based security constraints by performing query modifications. The first technique uses a relational model to represent metadata. This technique has the drawback of difficulty in expressing complex security classifications. The second technique uses logic-based rules for expressing computations and methods of execution. The number of rules required to represent a context-based constraint is exponential in the number of objects classified by the constraint. This technique was inefficient because the number of rules to be processed was very large. The third technique is based on using graphs to index security constraints and make access more efficient. Complexity of access control algorithms that use graphs representing context-based access constraints with groups of n members have complexity of $\mathcal{O}(n^2)$. These query modification techniques are not applicable in XML databases due to the semi-structured nature of data. Use of these techniques will result in over-classification.

6. CONCLUSIONS

In this paper we presented an RDF-based access control framework. RDF provides powerful and intuitive representation of meta-data, including security objects and policies. Our goals were to provide flexible security granularity and expressive access control model based on current web technologies.

This paper is our initial attempt to define RXACL, an RDF-based Access Control Language, for expressing and enforcing access control policies for XML documents. In this work we presented methods and algorithms to extend upon the existing XML access control models by allowing to define not only XML nodes and links as security objects, but their associations. We believe, that such flexibility is necessary to ensure increased information availability while providing security. Currently our framework supports discretionary access control under closed policy. The general form of access rules follow the $(s, o, \pm a)$ form, where s is a subject, o is a protection object, and $\pm a$ is a signed access mode. User requests are evaluated against the access control policy and the user's history file.

We are also planning to extend our framework to incorporate Role-Based and Mandatory Access Control models, as well as to extend the access control paradigm to inference protection, using the semantic constructs of RDF and ontologies. We will also incorporate schema-level analysis and protection against collaborating users.

7. ACKNOWLEDGMENTS

This material was partially supported by the National Science Foundation under Grants No. DUE-0112874 and IIS-0237782.

8. REFERENCES

[1] J. Barkley. Security in open systems. NIST Special Publication 800-7, 1994.

[2] T. Berners-Lee. Why RDF model is different from the XML model. Design Issues note, retrieved on September 1, 2003 from http://www.w3.org/DesignIssues/RDF-XML.html, October 1998.

[3] E. Bertino, M. Braun, S. Castano, E. Ferrari, and M. Mesiti. Author-X: A Java-based system for XML data protection. In *Proc. of 14th IFIP WG11.3 Working Conference on Database Security*, The Netherlands, August 2000.

[4] E. Bertino, S. Castano, and E. Ferrari. Securing XML documents with Author-X. *IEEE Internet Computing*, 3, May/June 2001.

[5] E. Bertino, S. Castano, E. Ferrari, and M. Mesiti. Controlled access and dissemination of XML documents. In *Proc. of 2nd ACM Workshop on Web Information and Data Management*, pages 22–27, Kansas City, 1999.

[6] E. Bertino, S. Castano, E. Ferrari, and M.Mesiti. Specifying and enforcing access control policies for XML document sources. In *World Wide Web Journal*, volume 3. Kluwer Academic Publishers, 2000.

[7] P. Buneman, S. B. Davidson, W. Fan, C. S. Hara, and W. C. Tan. Keys for XML. In *Proc. of the tenth International Conference on World Wide Web*, pages 201–210, 2001.

[8] P. Buneman, W. Fan, J. Simééon, and S. Weinstein. Constraints for semistructured data and XML. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 30(1):47–54, 2001.

[9] P. Buneman, W. Fan, and S. Weinstein. Path constraints on semistructured and structured data. In *Proc. ACM PODS*, pages 129–138, 1998.

[10] W. W. W. Consortium. Resource description framework (RDF) model and syntax specification. W3C Recommendation, retrieved on February 22, 1999 from http://www.w3c.org/TR/1999/REC-rdf-syntax-19990222, February 1999.

[11] W. W. W. Consortium. Extensible Markup Language Language 1.0 specification. W3C Recommendation, retrieved on October 6,2000 from http://www.w3.org/TR/2000/REC-xml-20001006, October 2000.

[12] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Design and implementation of an access control processor for XML documents. In *Proc. of 9th International World Wide Web Conference*, The Netherlands, 2000.

[13] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Regulating access to semistructured information on the web. In *Proc. of 16th IFIP TC11 Annual Working Conference on Information Security: Information Security for Global Information Infrastructures*, Beijing, China, August 2000.

[14] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. XML access control systems: A component-based approach. In *Proc. of 14th IFIP WG11.3 Working Conference on Database Security*, The Netherlands, August 2000.

[15] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Securing XML documents. In *Proc. of Conference on Extending Database Technology*, Prague, March 2002.

[16] F. Dridi and G. Neumann. Towards access control for logical document structure. In *Proc. of the Ninth International Workshop of Database and Expert Systems Applications*, pages 322–327, Vienna, Austria, August 1998.

[17] W. Fan and L. Libkin. Finite satisfiability of keys and foreign keys for XML data. Technical Report TUCIS-TR-2000-002, Department of Computer and Information Sciences, Temple University, 2000.

[18] W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. In *Proc. of 20th ACM Symposium on Principles of Database Systems*, 2001.

[19] W. Fan, P. Schwenzer, and K. Wu. Keys with upward wildcards for XML. *Lecture Notes in Computer Science*, 2113:657–667, 2001.

[20] W. Fan and J. Simeon. Integrity constraints for XML. In *Proc. of 19th ACM Symposium on Principles of Database Systems*, pages 23–34, 2000.

[21] A. Gabillon and E. Bruno. Regulating access to XML documents. In *Proc. of 15th IFIP WG11.3 Working Conference on Database Security*, 2001.

[22] S. Jajodia, M. Kudo, and V. S. Subrahmanian. Provisional authorizations. In *Proc. of 1st Workshop on Security and Privacy in E-Commerce*, 2000.

[23] T. F. Keefe, M. B. Thuraisingham, and W. T. Tsai. Secure query-processing strategies. In *IEEE Computer*, volume 22, March 1989.

[24] G. Klyne. Scenarios for using RDF in support of trust and access control. Memo, SWAD-Europe Project, Retrieved from http://ninebynine.org/SWAD-E/Trust-scenarios.html, December 2002.

[25] T. Koch, C. Krell, and B. Krḿer. Policy definition language for automated management of distributed systems. In *Proc. of The Second International Workshop on Systems Management*, Toronto, Canada, June 1996. IEEE Computer Society.

[26] M. Kudo and S. Hada. XML document security based on provisional authorizations. In *Proc. of the 7th ACM conference on Computer and Communications Security*, Athens, Greece, November 2000.

[27] M. Kudo and S. Hada. Access control model with provisional actions. In *IEICE Trans. Fundamentals*, volume E84-A, 2001.

[28] OASIS. eXtensible Access Control Markup Language (XACML) Version 1.1. Retrieved on July 24, 2003 from http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-1.1.pdf, July 2003.