XML 文書の変更における機密情報漏洩の検出手法

チャットウィチェンチャイ　ソムチャイ‡　　岩井原　瑞穂†

‡県立長崎シーボルト大学国際情報学部 〒851-2195 長崎県西彼杵郡長与町まなび野 1-1-1
†京都大学大学院情報学研究科社会情報学専攻 〒606-8501 京都市左京区吉田本町
E-mail: ‡somchaic＠sun.ac.jp, †iwaihara＠db.soc.i.kyoto-u.ac.jp

Abstract: To provide fine-grained access control to data in an XML document, XML access control policy is defined based on the contents and structure of the document. In this paper, we discuss confidential data disclosure problem caused by unsecured-update that modifies contents or structures of the document referred by the access control policy. In order to solve this problem, we propose an algorithm that decides whether a given update request of a user against an XML document is an unsecured-update under the user's access control policy.

Key Words: XML documents, Access Control Policy, XPath, Tree Embedding.

1. Introduction

XML [12] is rapidly gaining popularity as a mechanism for sharing and delivering information among businesses, organizations, and users on the Internet. The need of protecting confidential data in XML documents is becoming more and more important. A number of XML access control models are proposed in the literature [1, 3, 5]. XACML [8] is an OASIS standard for access control of XML documents. To provide fine-grained access control to data in XML document, these models use path expressions of XPath [13] for locating sensitive nodes in XML documents. The identification of a sensitive node is no longer restricted to the value of the node itself but depends on the context, the form of the path (from the root node to that node) and the children/descendants of that node. Hence definition of access control policy is strongly related to the node values and the structural relationship between nodes of XML documents. In the statistic analysis approach [7], XPath queries to the XML database can be checked whether having intersection with access control policies. The result of statistic analysis of a query is either grant, deny, or indeterminate. In the grant case, the XML database is accessed to answer the query. In the deny case, query evaluation is terminated without accessing the XML database. In the indeterminate case, the XML database is accessed to retrieve necessary data to determine accessibility. Updating XML data is still a research issue [11, 2, 6]. In [11], a set of basic update operations for both ordered and unordered XML data is proposed. The authors describe extensions to the proposed standard XML query language, XQuery, to incorporate the update operations. In [2], the authors have proposed an infrastructure for managing secure update operations on XML data. Each subject in the collaborative group only receives the symmetric key(s) for the portion(s) he/she is enabled to see and/or modify. Additionally, attached to the encrypted document, a subject receives some control information, with the purpose of making him/her able to locally verify the correctness of the updates performed so far on the document, without the need of interacting with the document server. In [6], the authors define new action types to systematically manage complex information of access right and to process various update queries in an efficient manner.

As we said before, definition of access control policy is strongly related to content and the structural relationship between nodes of XML documents. Confidential data disclosure problem may arise by the update that modifies node values or the structural relationship between nodes referred by the access control policy.

Motivating Scenarios: Consider the sample XML document (see *company.xml*) of Figure 1 stored in a XML database server. Suppose that Jane is in charge of a personnel officer of ABC Co., Ltd. Jane is allowed to maintain staff information except salaries of managers of London branch. Therefore, the security manager defines access control policy which consists of the following authorization rules for Jane.

R1: <Jane, company.xml, /company, rw, +>

R2: <Jane, company.xml, //branch[name="London"]
//staff[rank="Manager"]/salary, rw, –>

Authorization rule *R1* states that Jane is allowed to read and write data of the subtree rooted by company node of *company.xml*. *R2* states that Jane is not allowed to read and write data of subtree rooted by salary node of the managers of London branch. Based on *R1* and *R2*, salary data of Sara doesn't appear in the view (see Figure 2) over *company.xml* for Jane.

Confidential Disclosure Problem: Now, we show how Jane reads Sara's salary which is not allowed by the access control policy. Jane issues to the server an update request modifying rank value of Sara from "Manager" to "Clerk". By this way, Jane can read salary of Sara by requesting the server to send her the view over the updated *company.xml*. This security problem arises because there exists no authorization rule denying Jane to modify the data referred by the predicates of path expression of *R2* which denote the conditions of addressing the confidential data in *company.xml*.

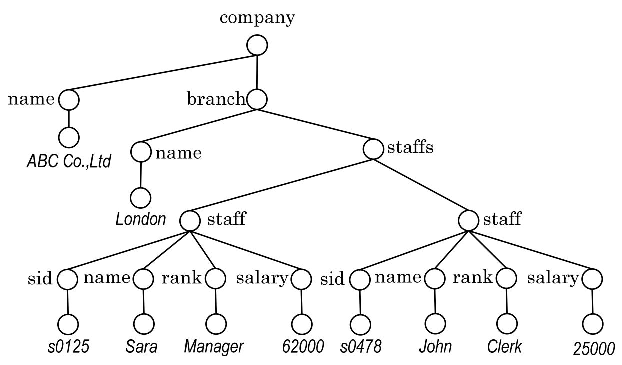

Fig.1. An example of a sample XML document

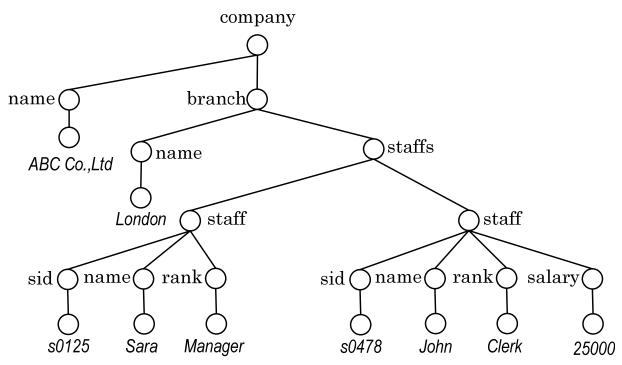

Fig.2. A View on the XML document of Fig.1 for Jane

To the best of our knowledge, there is no previous work discussing this security problem. The objective of this paper is to propose an algorithm that decides whether a given update request against an XML document is permitted under the requestor's access control policy and will not cause the confidential data disclosure. If the algorithm decides that the update request of the requestor has no privilege to execute the update request or the update causes the confidential data disclosure, the algorithm will reject the update request. Otherwise, the algorithm passes the update request to XML database system.

The rest of the paper is organized as follows. In Section 2, we give formal definitions of XML tree, tree patterns, tree embedding, authorization rules and update requests. Section 3 presents a formal definition of the problem. In Section 4, we present an algorithm that computes security labels that impose update constraints for some document nodes for given XML tree under given access control policy of a user. Section 5 presents an algorithm that decides whether given update request is not unsecured-update request and is permitted under the user's access control policy. Finally, the last section concludes this paper.

2. Basic Concepts and Definitions

2.1 Trees and Tree Patterns

We view an XML document as an unranked (in the sense that the number of children nodes of a particular node can be unbounded), ordered tree. Each node in the tree corresponds to an element, attribute or value. The edges in the tree represent immediate element-subelement or element-value relationships. Attribute nodes and text values can be handled similarly to element nodes.

Definition 2.1 An XML document is a tree $t = <V_t,E_t,r_t>$ over an infinite alphabet Σ called XML tree, where

- V_t is the node set and E_t is the edge set;
- $r_t \in V_t$ is the root of t; and
- each node v in V_t has a label (denoted as $label_t(v)$) from Σ. ▌

We assume that each text node is labeled with its textual value. Given an XML tree $t = <V_t, E_t, r_t>$, we say that $t' = <V_{t'}, E_{t'}, r_{t'}>$ is a subtree of t if $V_{t'} \subseteq Vt$ and $E_{t'} = (V_{t'} \times V_{t'}) \cap E_t$.

In this paper, we discuss a fragment of XPath[13] queries (called a *Simple XPath*). This fragment consists of label tests, child axes(/), descendant axes(//), and branches([ ]). Note that XPath expressions with upward axis (e.g., parent and ancestor axis) can be transformed into equivalent upward-axis-free ones [9], and are thus excluded from our discussions. Simple path can be generated by the following grammar ('ε' is the empty path, 'l' is a label for element or attribute name, and 'c' is a string constant):

$$p ::= \varepsilon \mid l \mid /p \mid //p \mid p_1/p_2 \mid p_1//p_2 \mid p[q]$$
$$q ::= p \mid p\ \theta\ c$$
$$\theta ::= < \mid \le \mid = \mid \ge \mid >$$

The above simple XPath expressions can be represented by the following tree patterns.

Definition 2.2 (Tree Patterns): A tree pattern p is a tree $<V_p, E_p, r_p, o_p, c_p>$ over Σ, where V_p is the node set and E_p is the edge set, and:

- each node v in V_p has a label from Σ, denoted as $label_p(v)$;
- r_p, $o_p \in V_p$ are the root and output node of p respectively; and
- c_p is a labelling function assigning a symbol from {'<', '≤', '=', '≥', '>'} to a text node. ▌

We present a child edge with a single line and present a descendant edge with a double line. For example, an XPath query *company/branch[name="London"]//staff [name="Sara"]/rank* is represented as a tree pattern shown in Figure 3(b), where the dark node is the output node. The size of a tree pattern, written as $|p|$, is defined as the number of its nodes. Without loss of generality, we refer to tree patterns as patterns in the rest of this paper.

We now define an embedding (also called pattern match) from a pattern to an XML tree as follows:

Definition 2.3 (Tree Embedding): Given an XML tree $t = <V_t, E_t, r_t>$ and a pattern $p = <V_p, E_p, r_p, o_p>$, an embedding from p to t is a function emb: $V_p \rightarrow V_t$, with following properties for every $x, y \in V_p$:

- Label-preserving: $\forall x \in V_p$, $label_p(x)=label_t(emb(x))$;
- Structure-preserving: $\forall e = (x, y) \in E_p$, if $label_p(e)$ = '/', $emb(x)$ is a child of $emb(y)$ in t ; otherwise, $emb(x)$ is a descendent of $emb(y)$ in t; and
- Value-matching: $\forall x \in V_p$ where $emb(x) \in V_t$ is a text node, the Boolean expression: $label_t(emb(x))\ c_p(x)\ label_p(x)$ is true. ▌

The embedding emb maps the output node o_p of p to a node $emb(o_p)$ in t. We say that the subtree $sub(t, p, emb)$ rooted by $emb(o_p)$ of t is the result of embedding. Note that $sub(t, p, emb)$ can also be seen as an XML tree. As an example, dashed lines between Figure 3(a) and (b) shows an embedding and its result is shown in Figure 3(c). Actually, there could be more than one embedding from p to t. We define the result of p over t, denoted as $p(t)$, as the union of results of all embeddings, i.e., $\cup_{emb \in EMB}\{sub(t, p, emb)\}$ where EMB is the set including all embeddings from p to t. Furthermore, we define an empty pattern denoted by ε as the result of evaluating ε over any XML tree is empty.

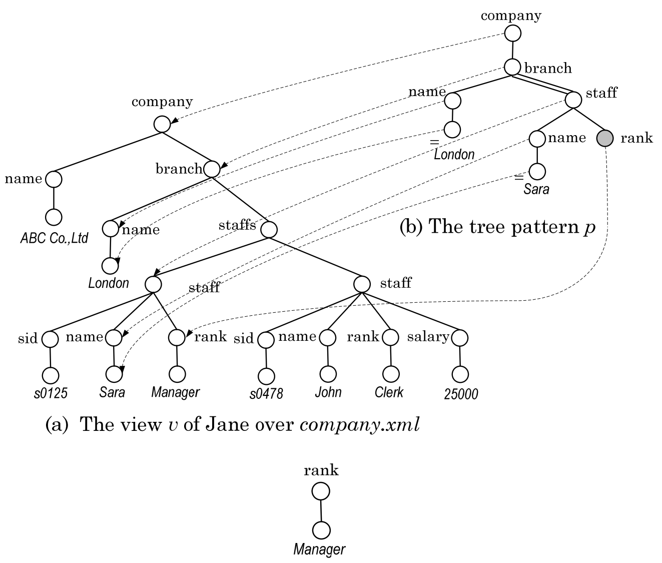

Fig.3. Embedding of the tree pattern p on the view v.

2.2 Authorization Rules

We use the term *access control policy*, or simply *policy*, for a set of *authorization rules*. Each authorization rule has the following format:

<*subject, doc-id, path, priv, sign*>, where

- *subject* is a user name, a user group, or a role[10];
- *doc-id* denotes an XML document identifier;
- *path* denotes a path expression of XPath identifying nodes within the XML document;
- *priv* is either read denoted by r or read/write denoted by rw; and
- *sign* ∈ {'+', '−'}, where '+' denotes grant and '−' denotes denial.

Authorization can be positive (granting access) or negative (denying access) to document nodes of an XML document. The *read* privilege allows a subject to view a document node. The *write* privilege allows a subject to append/remove a document node, and modify content of a document node. Authorization specified on a node is propagated to its all descendant nodes. The possibility of specifying authorization with different sign introduces potential conflicts among authorization rules. Here, the conflict resolution of the model is based on the following policies: *Descendant-take-precedence:* An authorization rule specified at a given level in the document hierarchy prevails over the authorization rules specified at higher levels; and *Denial-take-precedence:* In case conflicts are not solved by *descendant-take-precedence* policy, the authorization rule with negative sign takes precedence. We apply *denial-by-default* policy that denies any access request for a document node whose authorization cannot be derived from the authorization rules defined by the security manager.

2.3 Update Requests

We give a definition of an update request as follows.

<*subject*, *op*, *doc-id*, *path*, *content*>, where

- *subject* is a user name, a user group, or a role;
- *op* is *remove*, *append*, or *change* operation; and
- *doc-id* is an XML document identifier;
- *path* denotes a path expression of XPath identifying the context nodes within the XML tree; and
- *content* denotes either (*i*) name of an element / attribute, or (*ii*) textual value of the node to be written.

Table 1 explains details of the *operation* argument of an update request and necessary privileges of a subject for executing the operation. In this paper, for simplicity we assume that the documents before and after update hold the same *doc-id*. We also assume that a subject is allowed to append a node if the subject has *read/write* privilege on the node so that the subject can confirm the write result.

3. Problem Formalization

Let t be an XML tree before update, and t' be the XML tree after update. To address the confidential data disclosure problem in t', we need to identify information used to define how a node of t is mapped to that of t'. We call this information a *tree mapping*, which is defined as follow.

Let N_{del} be the set of deleted nodes of t, and N_{add} be the set of nodes that are newly added to t'. We call $N - N_{del}$ the set of *source nodes*. We also call $N' - N_{add}$ as the set of *target nodes*.

Definition 3.1 (Tree Mapping): Let t be an XML tree before update and t' be the XML tree after executing update u. Let N_t be the set of source nodes of t, and $N_{t'}$ be the set of target nodes of t'. $tmap_u$: $N_t \rightarrow N_{t'}$ is a total mapping from N_t to $N_{t'}$ by u. ▌

operation	*content*	Necessary privilege
The *append* operation appends a new node as a child of the context node.	Element name, attribute name, or textual value of the new node.	The *read/write* privilege on the new node. The *read* privilege on the parent node of the selected context node.
The *remove* operation allows the subtree rooted by the selected context node to be removed.		The *read/*write privileges on the selected context node and its all descendant nodes.
The change operation allows the content of the selected context node to be changed.	The new textual value.	Combinations of necessary privilege for *remove* and *append* operations.

Table 1: Necessary privileges for executing an update request

We define an *unsecured-update request* that results in confidential data disclosure as follows.

Definition 3.2 (Unsecured-Update Request): Let N_t be the set of source nodes of XML tree t before update, and $N_{t'}$ be the set of target nodes of XML tree t' after update, and $tmap_u$: $N_t \rightarrow N_{t'}$ is a total mapping from N_t to $N_{t'}$ by update request u. Let P_s be an access control policy of subject s on XML tree t, $deny_{s,r,t} \in N_t$ be the node set of t that is not allowed to read by s under P_s, and $permit_{s,r,t'} \in N_{t'}$ be the node set of t' that is allowed to read by s under P_s. u is an *unsecured-update request* under P_s if there exist $v \in deny_{s,r,t}$ and $v' \in permit_{s,r,t'}$ such that $v' = tmap_u(v)$ after executing u. ▌

For example, <*Jane, change, company.xml, //staff [name="Sara"]/rank, "Clerk"*> is an unsecured-update request under access control policy $P = \{R1, R2\}$ because salary of Sara which is confidential information becomes readable by Jane after executing this update request.

We use the following notations for defining properties of remove, append and change requests which are not unsecured update requests.

Definition 3.5 (Relevant Node Set): Let t be an XML tree, and $Path = \{p_1, p_2, .., p_m\}$ be a set of path expressions of authorization rules of access control policy P. *Relevant node set* of t under P, denoted by $RelNode(t, Path)$, is the set of nodes such that:
$RelNode(t, Path) = \{v \mid v \in \eta(t, p_i, emb), \forall emb \in EMB_i, \forall p_i \in Path\}$, where $1 \leq i \leq m$, $\eta(t, p_i, emb)$ is the set of nodes of t mapped from p_i by emb, and EMB_i is the set of embeddings from p_i to t. ▌

Lemma 1: (Secured Remove Request): Let t be an XML tree, P be access control policy of s on t. Let $Path'$ be the set of path expressions of authorization rules with negative sign, and $RelNode(t, Path')$ be the relevant node set of t under $Path'$. $<s$, "remove", t, p, $>$ is not the unsecured update request under P if the following conditions hold:

1. $\forall v \in \tau(t, p)\ write_P(v)$ = '+'; and
2. $p(t) \cap RelNode(t, Path') = \varnothing$,

where $\tau(t, p)$ is the node set of the subtrees rooted by $p(t)$. Here we call $<s$, "remove", t, p, $>$ holding the above conditions, *secured remove request for t under P*.

▌

Definition 3.6 (Relevant Paths): Let $t = <V_t, E_t, r_t>$ be an XML tree, and $<s$, "append", t, p, $content>$ be an append request. Let $Path = \{p_1, p_2, .., p_m\}$ be a set of path expressions of authorization rules of access control policy P. *Relevant paths* of p for appending *content* under P, denoted by $RelPath(p, content, P)$, is the subset of $Path$ of authorization rules of P where each q of the subset holds the following properties:

(1) q has a leaf node v where the Boolean expression: $label_t(v)\ c_q(v)\ content$ is true; and
(2) the leaf node v has the parent node whose label is the same as that of output node of p. ▌

We denote $SubRelPath(p, content, P)$ the set of path expressions computed from path expressions of $RelPath(p, content, P)$ by deleting its node v (see condition 1 and 2 of definition 3.4).

Lemma 2 (Secured Append Request): Let $t = <V_t, E_t, r_t>$ be an XML tree, Let P' and P'' be the set of authorization rules with negative sign and positive sign, respectively, where $P' \cup P'' = P$. $<s$, "append", t, p, $content>$ is not the unsecured append request under P if

```
Algorithm LabelTree (t, P_s)
Input: 1. XML tree $t = <V_t, E_t, r_t>$, and
       2. Access control policy $P_s = \{R_1, R_2, .., R_m\}$ of subject $s$ on $t$.
Output: XML tree $t$ with security labels.
Method:

Step1:   Initialize read and write labels of each $v \in V_t$ with $\varepsilon$.
Step2:   For each $R_i = <s, doc\text{-}id_i, path_i, priv_i, sign_i> \in P_s$, where $1 \leq i \leq m$ do {
Step3:     Let $p$ be the tree pattern of $path_i$, and $p(t)$ be the set of nodes of $t$ addressed by $p$.
           Compute $p(t)$.
Step4:     For each $u_k \in p(t)$ do {
Step5:       If ($priv_i$ = 'r' or $priv_i$ = 'rw') then {
Step6:         If $rlbl(u_k) = \varepsilon$ then $rlbl(u_k) = <r, sign_i, 1>$
Step7:         else If sign of $rlbl(u_k)$ is '+' and $sign_i$ = '–' then
                    $rlbl(u_k) = <r,–,1>$   /* denial-take-precedence*/
Step8:       }
Step9:       If ($priv_i$ = 'w' or $priv_i$ = 'rw') then {
Step10:        If $wlbl(u_k) = \varepsilon$ then $wlbl(u_k) = <w, sign_i, 1>$
Step11:        else If sign of $wlbl(u_k)$ is '+' and $sign_i$ = '–' then
                  $wlbl(u_k) = <w,–,1>$   /* denial-take-precedence*/
Step12:      }
Step13:      If $sign_i$ = '–' then {
Step14:        Let $emb_k$ be the embedding from $p = <V_p, E_p, r_p, o_p>$ to $t$, where $u_k = emb_k(o_p)$.
Step15:        For each $v \in V_p$ and $v \neq o_p$ do
Step16:          $wlbl(emb_k(v)) = <w,–,0>$ /* update constraint for preventing data disclosure to $u_k$ */
             }
           }
         }
Step17:  return t.
```

Fig.4. The *LabelTree*

the following conditions hold:

(1) $\forall v \in p(t)$ $read_P(v)$ = '+' ;

(2) One of the following conditions hold:

  2.1 $\forall v \in p(t)$ $write_P(v)$ = '+' ; or

  2.2 $\exists q'' \in SubRelPath(p, content, P'')$ s.t. $q''(t) \supseteq p(t)$; or

(3) $\neg\exists q' \in SubRelPath(p, content, P')$ s.t. $q'(t) \cap p(t) \neq \varnothing$; and

(4) $p(t) \cap RelNode(t, SubRelPath(p, content, P)) = \varnothing$.

Here we call <*s*, "append", *t*, *p*, *content*> holding the above conditions, *secured append request for t under* P_s. ▌

Lemma 3: (Secured Change Request): Let *t* be an XML tree, and $P_s = P'_s \cup P''_s$ be access control policy of subject *s*, and $RelNode(t, P'_s)$ be the relevant node set of *t* under P'_s. <*s*, "change", *t*, *p*, *content*> is not the unsecured update request under P_s if the following conditions hold:

(1) $\forall v \in p(t)$ $write_P(v)$ = '+';

(2) $\exists q'' \in SubRelPath(p, content, P'')$ s.t. $q''(t) \supseteq p(t)$;

(3) $\neg\exists q' \in SubRelPath(p, content, P')$ s.t. $q'(t) \cap p(t) \neq \varnothing$; and

(4) $p(t) \cap RelNode(t, SubRelPath(p, content, P)) = \varnothing$.

Here we call <*s*, "change", *t*, *p*, *content*> holding the above conditions, *secured change request for t under* P_s. ▌

4. Security Labelling Algorithm

We first specify the authorization types of document nodes by security labels that are defined as follows.

Definition 4.1 (Security Labels): A security label for a node *n* of XML tree $t = <V_t, E_t, r_t>$ is represented by

<*priv, sign, flag*>, where

- *priv* is either read denoted by *r* or write denoted by *w*;
- *sign* ∈ {'+', '–'}, where '+' denotes grant and '–' denotes denial; and
- *flag* ∈ {*1*, *0*} denotes type of authorization propagation, where value 1 denotes *cascade*, and value 0 denotes *no-cascade*.

We call a security label with read privilege *read label*. We define *rlbl*: $V_t \rightarrow \{<r,+,0>, <r,+,1>, <r,-,0>, <r,-,1>, \varepsilon\}$ is a total mapping from nodes of V_t to the read labels, where ε denotes undefined authorization. Note that *rlbl*(*n*) denotes the read label on node *n*. We call a security label with read privilege *read label*. We define *wlbl*: $V_t \rightarrow \{<w,+,0>, <w,+,1>, <w,-,0>, <w,-,1>, \varepsilon\}$ is a total mapping from nodes of V_t to the write labels. *wlbl*(*n*) denotes the write label on node *n*. ▌

Given an XML tree *t* and access control policy P_s of user *s*, we propose the *LabelTree* algorithm (see Figure 4) that computes read and write labels for only the document nodes that satisfied by path expressions of authorization rules of P_s. For each authorization rule, step4 thru step16 of *LabelTree* assign security labels to the nodes addressed by the path expression $path_i$ of the rule. If sign of the rule is negative, *LabelTree* also assigns security labels of negative sign and no-cascade to the nodes of *t* corresponding to nodes of $path_i$ in order to prevent unsecured-update.

Figure 5 shows the XML tree of Fig.1 after labelling read and write security labels by *LabelTree* under access control policy *P* = {*R1*, *R2*}. Notice *LabelTree* generates write labels with negative sign for the nodes that are mapped from tree pattern *p* depicted in Fig. 2(b) by tree embedding. As Jane is not allowed to modify names/values of these nodes and structural relationship among these nodes, the confidential data disclosure will not occur.

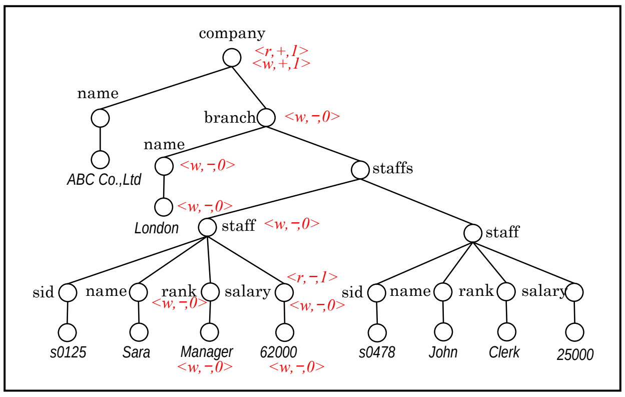

Fig.5. The XML tree after labelling read and write security labels by the *LabelTree* algorithm

Theorem 1: Given an XML tree $t = <V_t, E_t, r_t>$, an access control policy $P_s = \{R_1, R_2, .., R_n\}$ for subject *s*, where $R_i = <s, doc\text{-}id_i, p_i, priv_i, sign_i>$, *LabelTree* computes security labels for document nodes of *t* in $O(|t|^4 \bullet |P_s|)$ where $|t|$ is the size of *t* and $|P_s|$ is number of authorization rules of P_s.

Proof: Gottlob et al. [4] have proposed the polynomial-time algorithms for XPath processing by

```
Algorithm UpdReqCheck (t, P, updreq, Permit)
Input:
   1. XML tree $t = <V_t, E_t, r_t>$ with read and write labels,
   2. Access policy $P = P' \cup P''$ of subject $s$ on $t$, and
   3. Update request $updreq$ of $s$ on $t$, where $updreq = <s, op, doc\text{-}id, p, content>$.
Output:
Permit whose value denotes whether the update request should be permitted.
Method:
Step1:  Let $Path'$ and $Path''$ be the sets of path expressions of $P'$ and $P''$, respectively.
        $Permit$ = FALSE.
Step2:  Compute $p(t)$ where $p(t)$ be the set of nodes of $t$ that is located by $p$.
Step3:  For each $v_i \in p(t)$ do {
Step4:     If $op$ = 'remove' or $op$ = 'change' then {
Step5:          If ($s$ has no write privilege on $v_i$ and all descendant nodes $v_i$) or
                   ($p(t) \cap RelNode(t, Path') \neq \varnothing$) then return Permit
Step6:     }
Step7:     If $op$ = 'append' or $op$ = 'change' then {
Step8:          If $read_p(v_i)$ = '–' then return Permit
Step9:          If $\neg\exists q'' \in SubRelPath(p, content, P'')$ s.t. $q''(t) \supseteq p(t)$ or
                    $\exists q' \in SubRelPath(p, content, P')$ s.t. $q'(t) \cap p(t) \neq \varnothing$ then return Permit
Step10:         If $p(t) \cap RelNode(t, SubRelPath(p, content, P)) \neq \varnothing$ then return Permit
           }
Step11: }
Step12: $Permit$ = TRUE
Step13: return Permit
```

Fig. 5: The *UpdReqCheck* algorithm

using a form of dynamic programming. Based on this, computation of the node set satisfied by the Extended Wadler Fragment [4], which covers our simple XPath expression, is processed in time $O(|t|^2 \bullet |p|^2)$, where $|t|$ denotes the size of the XML document tree and $|p|$ is the size of the query. Therefore, the complexity of step3 is $O(|t|^2 \bullet |p|^2)$. Since $|p|$ and $|p(t)|$ can be as big as $|t|$, step4 thru step16 are executed $|t| \bullet |P_s|$ times in the worst case. Then, the complexity of *LabelTree* is bounded to $O(|t|^4 \bullet |P_s|)$ where $|P_s|$ is the number of authorization rules of access control policy P_s. ▌

5. Update Request Checking Algorithm

Based on security labels of document nodes computed by the *LabelTree* algorithm from access control policy of a user, we propose the *UpdReqCheck* algorithm (see Figure 6) that decides whether given update request *updreq* on XML tree *t* is not an unsecured-update request under access control P_s of subject *s* on *t*.

Theorem 2: Given an XML tree $t = <V_t, E_t, r_t>$ with security labels, access control policy P_s of *s* on XML tree *t*, and an update request *updreq <s, op, doc-id, p, content>*, *UpdReqCheck* decides whether *updreq* is not an unsecured-update request under P_s in $O(|t|^2 \bullet |p|^2 \bullet |P_s| \bullet |p(t)|)$, where $|t|$ is the size of *t*, $|p|$ is the size of query of *updreq*, $|P_s|$ is number of authorization rules of P_s, and $|p(t)|$ is number of nodes addressed by *p*.

Proof: Computation of $p(t)$ at step2 can be done in $O(|t|^2 \bullet |p|^2)$. Step3 is processed $|p(t)|$ times. Time complexity of processing step4 and step13 is bounded to that of checking *insert-before* operation request. Time complexity of step9-step11 is $O(|t|^2 \bullet |p|^2 \bullet |P_s|)$. As total time complexity of step1-step15 is $O(|t|^2 \bullet |p|^2 + |t|^2 \bullet |p|^2 \bullet |P_s| \bullet |p(t)|)$, time complexity of *UpdReqCheck* is bounded to $O(|t|^2 \bullet |p|^2 \bullet |P_s| \bullet |p(t)|)$. ▌

6. Conclusions and Future Work

As authorization rules of existing XML access control model are defined based on node values and the structural relationship between nodes of XML documents, confidential data disclosure problem may arise by the unsecured-update that modifies values or the structural relationship between nodes referred by the authorization rules. In order to solve this problem, this paper has formalized the problem and proposed an algorithm that decides whether a given update request against an XML document is not unsecured-update request and is permitted under the requestor's access control policy.

We are going to investigate the possibility of utilizing DTDs or schemas of XML documents to reduce the complexity of computing security labels for XML tree and the complexity of deciding whether a given update request is the unsecured-update request.

References

[1] E. Bertino, S. Castano, E. Ferrari, M. Mesiti, "Specifying and Enforcing Access Control Policies for XML Document Sources," WWW Journal, vol.3, n.3, 2000.

[2] E. Bertino, G. Mella, G. Correndo, E. Ferrari. "An infrastructure for managing secure update operations on XML data," In Proc. of 8th ACM Symposium on Access Control Models and Technologies (SACMAT03), pp.110-122, 2003.

[3] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati, "A Fine-Grained Access Control System for XML Documents," ACM TISSEC, vol. 5, no. 2, 2002.

[4] G. Gottlob, C. Koch, and R. Pichler, "XPath Query Evaluation: Improving Time and Space Efficiency". In Proc. 19th IEEE International Conference on Data Engineering (ICDE'03), 379-390, 2003.

[5] M. Kudo and S. Hada, "XML Document Security based on Provisional Authorization," Proc.7th ACM Conf. Computer and Communications Security, pp. 87-96, 2000.

[6] C.H. Lim, S. Park, S.H. Son, "Access control of XML documents considering update operations", In Proc. of the 2003 ACM workshop on XML security, pp. 49-59, 2003.

[7] M. Murata, A. Tozawa, M. Kudo, S. Hada, "XML Access Control Using Static Analysis," Proc. ACM Conf. Computer and Communications Security, pp. 73–84, 2003.

[8] OASIS XACML Technical Committee, "eXtensible Access Control Markup Language (XACML) Version 2.0," http://www.oasis-open.org/specs/index.php#xacmlv2.0 (Feb 2005).

[9] D. Olteanu, H. Meuss, T. Furche, F. Bry, "XPath: Looking Forward," In XML-Based Data Management and Multimedia Engineering, EDBT Workshop, LNCS 2490, 109-127, 2002.

[10] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," IEEE Computer, 29(2), pp.38-47, 1996.

[11] I. Tatarinov, Zachary G. Yves, Alon Y. Halevy, Daniel S. Weld. "Updating XML". In ACM SIGMOD 2001 May 21-24, Santa Barbara, California, USA.

[12] W3C (2000). Extensible Markup Language (XML) 1.0 (Second Edition). Available at http://www.w3c.org/TR/ REC-xml (Oct 2000).

[13] W3C (1999). XML Path Language (XPath) Version 1.0. Available at http://www.w3c.org/TR/xpath (Nov 1999).