A Simple Approach to Conflict Resolution for Access Control in XML

Tuangporn Tantichalermpan¹ and Yongyuth Permpoontanalarp²
Logic and Security Laboratory
Department of Computer Engineering
King Mongkut’s University of Technology Thonburi
91 Suksawasd 48, Bangmod, Bangkok 10140
Email: ¹tuang_3@yahoo.com, ²yongyuth@cpe.eng.kmutt.ac.th

Abstract

Extensible Markup Language (XML) is a language that offers a flexible way to create common interchange formats for information sharing. In order to allow only authorized users to access information in XML, an access control system is required. Such an access control system requires the specifying of authorizations for each piece of information to be shared. Since there may be more than one authorization specified for the same information, conflicting authorizations may occur. We argue that existing approaches to conflict resolution do not offer two important properties which are intuitive conflict resolution policy and simple implementation within the same framework. In this paper, we propose a simple approach which offers the two important properties within a single framework for resolving conflicting authorizations in XML. Many kinds of conflict resolution policy can be employed in our approach without the need for the new implementation.

Keyword: Access Control / Computer Security

1. Introduction

Due to the Internet and Intranet, information sharing is becoming widespread. The security of shared information is crucial. Such security issue concerns with the accessibility of such information by authorized users. In order to allow only authorized users to access such information, an access control system [7, 10] is required. Access control policies govern the access of users to information on the basis of the user’s identity and a collection of rules and such rules are called authorization rules or just authorizations.

Extensible Markup Language (XML) [5, 6] offers a flexible way to create common interchange formats for information sharing on the Internet and Intranet.

Access control in Operating systems provides the controlling of accesses to XML documents. Such kind of access control offers a document-level access control. In particular, a whole document is either granted to or prohibited by a user. Recently, access control systems for XML e.g. [1], [2], and [3] are proposed to deal with a content-level access control. In particular, some part of a document can be either granted to or prohibited from a user. In this paper, we focus on the content-level access control.

An XML document is represented as a tree where a node in a tree represents a content element in the document. Authorization, namely permission or prohibition, is required for every node in such a tree. However, it is not practical to require an administrator to specify authorizations for every node because there may be a large number of nodes in a document. Thus, authorizations can be specified for some nodes in a document. By using some propagation mechanism of such specified authorizations along the tree, authorizations can be obtained for every node.

Since there may be more than one authorization applied for the same piece of information, conflicting authorizations may occur. For example, during the propagation of authorizations along a tree, a conflict between specified authorizations and propagated authorizations may occur. When such a conflict occurs, a method to resolve such conflict is required.

In [2,3], intuitive conflict resolution policies are proposed. In particular, in [2] the conflict resolution policy is based on the notion of Most-Specific-Objects and Denial-Takes-Precedence. More specifically, if there is a conflict between specified authorizations and propagated authorizations, the specified authorization will take precedence over the propagated authorizations. This is because a specified authorization is considered to be more specific than a propagated authorization. However, in case of the conflict between two specified authorizations, negative authorization (prohibition) will take precedence over positive authorization (permission). Also, [3] employs similar kind of conflict resolution policy. Even though [2,3] provide intuitive conflict resolution, they require complicated implementation mechanism.

In [1], another kind of conflict resolution policy is proposed. The resolution policy is based on priority defined by an administrator. Priority is represented
by the set of integers and it is used to prioritize authorization rules. In particular, an authorization rule with higher value of priority will take precedence over an authorization rule with lower value of priority. Thus, the use of priority offers a simple way to calculate the precedence of conflicting authorization rules. However, the conflict resolution in [1] is neither intuitive nor practical. If there are many conflicting rules, it may be hard to know which rule will take precedence.

Nonetheless, the approach in [1] offers an advantage over the other two existing approaches [2, 3] in that by using the concept of priority the former provides a simple mechanism to implement the conflict resolution system, whereas the latter two require complex implementation mechanisms.

In this paper, we propose a simple approach for resolving conflicting authorizations in XML. Our approach offers not only intuitive conflict resolution policy but also a simple mechanism to implement such conflict resolution. Our approach is motivated by the use of priority in [1] and the intuitive conflict resolution policy in [2, 3]. In particular, our approach employs priority as an underlying mechanism for conflict resolution and it allows intuitive conflict resolution policies to be specified. Furthermore, we propose a mapping from intuitive conflict resolution policies to priority-based conflict resolution.

Given authorization rules and intuitive conflict resolution policies, our mapping would generate priorities for those authorization rules which avoid conflicts according to the given policies. Many kinds of conflict resolution policies can be employed in our approach without the need of new implementation of the mechanism. In addition, we prove that our mapping is correct with respect to the intuitive conflict resolution policy.

2. Related Work

A document in XML requires the presence of Document Type Definition (DTD) which defines types and structure of information in documents. Authorizations can be specified for both documents and their DTD. Thus, XML access control system takes into account authorizations specified for both documents and their DTD by combining them together. However, the authorizations specified for documents are considered to be more specific than those specified for their DTD. Thus, if there is a conflict between authorizations in document and its DTD, some conflict resolution policy is required, for example, more specific authorizations take precedence over less specific ones.

There are three main systems [1, 2, 3] that proposed the access control for XML. The three systems share similar concept in that they provide methods for authorization propagation and conflict resolution. However, the methods employed in those systems are different.

2.1. Gabillon’s approach [1]

In [1], an authorization rule is in the following form: (subj, signed-action, obj, pri) where subj, signed-action, obj and pri stand for a subject (or a user), signed action, an object and priority, respectively. Signed action stand for either the permission of the prohibition of a privilege (or authorization) given to a subject on an object, for example grant-read which permits read access and deny-read which denies read access. Priority is just the set of integers.

The conflict resolution policy is based on priority and the order of authorization rules in an authorization file. Both priority and order are defined by an administrator. The conflicting rule with higher priority prevails over the conflicting rule with lower priority. However, if those conflict rules have the same priority, then authorization rule with the lower order in authorization file (sheet) prevails over the conflicting authorization rule with the higher order.

In [1], authorizations are propagated from a node they are specified to all descendents of the node. For example, assume that the object and subject are represented in tree structure as in figure 1 and figure 2, respectively. Suppose that authorization rules are as follows.

Rule #1: (sub1, deny-read, ob1, 1)
Rule #2: (sub2, grant-read, ob5 pri= 1)
Rule #3: (sub3, grant-read, ob2 pri= 3)

For object ob5 and subject sub2, rule #1 is in conflict with rule #2. Because rule #1 is propagated to all descendents of subject sub1 and object ob1, and rule #2 is propagated to all descendents of subject sub2. For object ob2 and subject sub3, rule #1 is in conflict with rule #3 due to the propagation. Rule #2 takes precedence over rule #1 due to the order of authorization rules whereas rule #3 takes precedence over rule #1 due to the priority.

Figure 1. The object hierarchy
The following is the subject hierarchy

![Subject Hierarchy Diagram](image)

Figure 2. The subject hierarchy

Even though the conflict resolution in [1] offers a simple mechanism to resolve conflicts by using priority and the order of conflicting rules, the approach is neither intuitive nor practical. In particular, if there are many authorization rules caused by large structure of XML documents and large number of subjects, then it is difficult for administrators to design and assign priorities to those rules which guarantee the correctness of the conflict resolution. This is because it is hard for administrators to see the effects of the conflict resolution of a large set of authorization rules by simply looking into those rules.

Moreover, it is not practical to define number-based priorities which can be used for a large number of rules since such number-based priorities are not suitable for the situation where authorization rules may be subsequently inserted or deleted. For example, Rule #4: (sub3, grant-read, ob2) is required to insert into the authorization file. The administrator has to know what the priority and the order of this new rule should be. Therefore the administrator needs to check all of existing authorization rules that may be in conflict with this new rule and then specify the priority number and order of new rule in authorization file.

2.2. Bertino’s approach [2]

In [2], an authorization rule is in the form: \( (\text{subj}, \text{obj}, \text{action}, \text{propagation option}, \text{sign}) \) where action, propagation option and sign stand for privilege, the method of propagation and either the permission or the prohibition of action. The propagation options include cascade, first_lev, and no_prop. The cascade means the propagation from a node to all its descendants. The first_lev means the propagation from a node to its children. The no_prop means no propagation.

The conflict resolution policy of this approach is the Most-Specific-Subject-Takes-Precedence policy and Denial-Takes-Precedence policy. The Most-Specific-Object-Takes-Precedence policy uses the specificity of objects in the object hierarchy to resolve conflicting authorization rules. If objects in those conflicting rules have the same specificity, then the Denial-Takes-Precedence policy will be employed which allows the prohibition to take precedence over the permission.

The following are the authorization rules.

Rule #1: (sub2, ob1, read, first_lev, grant)
Rule #2: (sub2, ob4, read, no_prop, deny)
Rule #3: (sub2, ob4, read, no_prop, grant)

According to the object hierarchy illustrated in figure 1, Rule #1 and Rule2 are in conflict due to the propagation. Since object ob4 is more specific than object ob2 in the object hierarchy, Rule2 prevails over Rule1. Rule #2 is in conflict with rule #3. Rule #2 takes precedence over rule #3 since both rules are defined for the same object (thus they have the same level of object specificity) and rule #2 is defined for denial sign.

Even though [2] offers intuitive approach to the conflict resolution of authorization rules, it requires a complex mechanism to implement such conflict resolution.

2.3. Samarati’s approach [3]

In [3], authorization rule is in the form: \( (\text{Subject}, \text{Object}, \text{Action}, \text{Sign}, \text{Type}) \). Type encodes two kinds of information: propagation and conflict resolution during the propagation. There are two choices of propagation: no propagation and cascade propagation. There are three levels of priorities, namely Hard, Neutral and Soft which are used for resolving a conflict during the propagation. There are eight kinds of types and they can be ordered totally. For example, RDH type means the cascade propagation with hard priority which takes precedence over RDN type which stands for the cascade propagation with neutral priority. If RDH type is propagated to a node with a conflicting authorization of RDN type, RDH will take precedence over RDN on the node.

After the propagation, there may be conflicting authorizations defined for a node in a document. Those conflicts will be resolved using the policy: Most-Specific-Subject-Takes-Precedence and Denial-Takes-Precedence.

For example, refer to the figure 1 and figure 2.
Rule #1: (sub1, ob1, read, -, RS)
Rule #2: (sub2, ob5, read, +, LS)
Rule #3: (sub2, ob5, read, -, LS)

where RS and LS stand for the cascade propagation with soft priority and no propagation with soft priority, respectively.
Rule #1 is in conflict with rule #2 due to the authorization propagation from object ob1 to object ob5. Since LS has more priority than RS, rule #2 takes precedence over rule #1.

Rule #2 is in conflict with rule #3. Type cannot be used to resolve this case. But the policy Most-Specific-Subject-Takes-Precedence and Denial-Takes-Precedence can be used for resolving this conflict. Therefore, rule #3 prevails over rule #2 because of the Denial-Takes-Precedence policy.

Similar to [2], Samarati’s approach offers an intuitive conflict resolution policy but it requires complicated implementation for the conflict resolution.

3. Our Simple Conflict Resolution Method

In our approach, we use priority as the central mechanism for conflict resolution. In particular, appropriate priorities are represented by the set of integers and are generated automatically for authorizations, given such authorizations and intuitive conflict resolution policies as inputs. By combining the use of priority and intuitive conflict resolution policy in this way, our approach extends [1] with intuitive conflict resolution policy. The output of our method is a set of authorizations with priority and those authorizations are in the same format as those in [1]. Thus, the authorizations obtained from our method can be used as an input to [1]. The overview of our architecture is shown in figure 3.

An input authorization rule in our approach is in the form (subject, object, signed-action) and an output authorization rule obtained after the calculation of priorities is in the form (subject, object, signed-action, priority).

Figure 3. The overview of our approach to conflict resolution with Gabillon’s system

Our system calculates the priority number for each rule that the administrator has defined by using our Priority Definition.

We propose a definition used to calculate priority for an authorization. The definition takes some input parameters. By specifying values for those parameters, one or more combinations of intuitive conflict resolution policies can be captured in our definition. Therefore, several kinds of policies can be used in our system.

**Priority definition**

The priority in our approach can be defined as follows:

\[ P = (W1*LS) + (W2*LO) + (W3*S) \]

where: 
- \( P \) = Priority number,
- \( LS \) = Level number of subject node in the subject hierarchy tree (from the root to a specific subject node),
- \( LO \) = Level number of object node in the object hierarchy tree (from the root to a specific object node),
- \( S \) = Sign value which is either 0 or 1, and
- \( W1, W2, W3 \) = weights which are integers \( \geq 0 \).

Suppose that \( Ts \) stands for the total level numbers of the subject hierarchy tree, and \( To \) stands for the total level numbers of the whole object hierarchy tree, \((W1*Ts)\) means the maximum value that the subject can contribute for resolving a conflict. Moreover, \((W2*To)\) means the maximum value that the object can contribute for resolving a conflict, and \(W3\) means the maximum value that the sign can contribute for resolving a conflict.

This definition is general in that it is not defined for any particular kind of conflict resolution policy. However, it can be used for some particular kind of conflict resolution policies by specifying values for weight parameters that are \( W1, W2 \) and \( W3 \). Below, we show how to capture each kind of conflict resolution policies in our definition in general ways. In particular, instead of showing specific values for those parameters, we identify conditions on those parameters that correspond to each kind of conflict resolution policies.

In this paper, we consider four intuitive conflict resolution policies which are Denial-Takes-Precedence, Permission-Takes-Precedence, Most-Specific-Subject-Takes-Precedence, and Most-Specific-Object-Takes-Precedence. In case of the conflict between two authorizations, Denial-Takes-Precedence means that negative authorization (prohibition) will take precedence over positive authorization (permission) whereas Permission-Takes-Precedence means that positive authorization will take precedence over negative authorization.

Most-Specific-Subject-Takes-Precedence means that if there is a conflict between specified authorizations and propagated authorizations for a subject, the specified authorization for the subject will take precedence over the propagated
authorizations for the subject. *Most-Specific-Object-Takes-Precedence* has the same meaning as *Most-Specific-Subject-Takes-Precedence* but it is applied to objects.

Our priority definition consists of three parts: subject part \((W_1*LS)\), object part \((W_2*LO)\), and sign part \((W_3*S)\). All of 4 conflict resolution policies can be captured in our priority definition as follows.

1. \((W_1*LS)\) is used to capture *Most-Specific-Subject-Takes-Precedence* policy, and \(W_1 > 0\).
2. \((W_2*LO)\) is used to capture *Most-Specific-Object-Takes-Precedence* policy, and \(W_2 > 0\).
3. \((W_3*S)\) is used to capture *Denial-Takes-Precedence* policy, and \(W_2 > 0\) and for deny action, \(S = 1\) and for grant action, \(S = 0\).
4. \((W_3*S)\) is used to capture *Permission-Takes-Precedence* policy, and \(W_2 > 0\) and for deny action, \(S = 0\) and for grant action, \(S = 1\).

If only one policy is used, weights for the two remaining parts must be set to 0. For example, if only *Most-Specific-Subject-Takes-Precedence* policy is used, then \(W_2\) and \(W_3\) are equal to 0.

If two or more policies are used, we have to consider the order relationship of those policies, and specify weights so that they correspond to the order relationship. For example, in *Subject-Takes-Precedence* policy and *Denial-Takes-Precedence* policy, *Subject-Takes-Precedence* policy is the first policy that is used for resolving and then *Denial-Takes-Precedence* policy will be used in the case that first policy cannot resolve the conflict. The combined policies can be mapped into our priority definition as:

\[
P = (W_1*LS) + (W_2*LO) + (W_3*S)
\]

where \(W_1 > W_2 > 0\)

Since only *Subject-Takes-Precedence* policy and *Denial-Takes-Precedence* policy are considered, then \(W_2\) is equal to 0. Moreover, since *Subject-Takes-Precedence* policy takes effect before *Denial-Takes-Precedence* policy, \(W_1\) is more than \(W_3\). Note that if \(W_1 > W_3\), then the part \((W_1*LS)\) contributes more than the part \((W_3*S)\) for the calculation of priority.

The following are example of capturing the intuitive conflict resolution policies into our priority definition.

**Example 1** If we want to capture the *Most-Specific-Object-Takes-Precedence* policy and *Denial-Takes-Precedence* policy into our priority definition, the priority definition will be:

\[
P = (W_1*LS) + (W_2*LO) + (W_3*S)
\]

where \(W_2 > W_3 > 0\), \(W_1*LS = 0\), \(S_{deny} = 1\) and \(S_{grant} = 0\).

The priority definition can be redefined as:

\[
P = (W_2*LO) + (W_3*S)
\]

This combination is referred to the conflict resolution policies as used in Bertino’s approach [2]. With reference to figure 1 and 2, the following are the authorization rules.

**Rule #1**: (sub2, ob3, grant-read)

**Rule #2**: (sub2, ob4, deny-read)

**Rule #3**: (sub2, ob4, grant-read)

Assume \(W_2 = 2\), \(W_3 = 1\). The authorization with priority after generated by our priority definition will be:

**Rule #1**: (sub2, ob2, grant-read, pri=4)

**Rule #2**: (sub2, ob4, deny-read, pri=7)

**Rule #3**: (sub2, ob4, grant-read, pri=6)

According to the object hierarchy illustrated in figure 1, rule #1 and rule2 are in conflict due to the propagation. Since rule #2 has more priority than rule #1, rule2 prevails over rule1. Rule #2 is in conflict with rule #3. Rule #2 takes precedence over rule #3 since rule #2 has more priority than rule #3. These results are the same as Bertino’s approach [2].

**Example 2** If we want to capture the *Most-Specific-Object-Takes-Precedence* policy, the *Most-Specific-Subject-Takes-Precedence* policy and the *Denial-Takes-Precedence* policy into our priority definition, the priority definition will be:

\[
P = (W_1*LS) + (W_2*LO) + (W_3*S)
\]

where \(W_2 > W_1*TS, W_1 > W_3 > 0\), \(S_{deny} = 1\) and \(S_{grant} = 0\).

The authorization rules are as follow.

**Rule #1**: (sub1, ob1, deny-read)

**Rule #2**: (sub2, ob5, grant-read)

**Rule #3**: (sub3, ob2, grant-read)

**Rule #4**: (sub3, ob3, grant-read)

**Rule #5**: (sub12, ob10, deny-read)

**Rule #6**: (sub15, ob13, deny-read)

From the subject hierarchy we know that \(TS = 6\).

Assume that \(W_3 = 1\), \(W_1 = 2\), \(W_2 = 13\).

After generating the priority number, the authorization rules will be shown as following.

**Rule #1**: (sub1, ob1, deny-read, pri = 16)

**Rule #2**: (sub2, ob5, grant-read, pri = 43)

**Rule #3**: (sub3, ob2, grant-read, pri = 30)

**Rule #4**: (sub3, ob3, grant-read, pri = 30)

**Rule #5**: (sub12, ob10, deny-read, pri = 61)

**Rule #6**: (sub15, ob13, deny-read, pri = 76)

From 6 rules above, it can be concluded as follows.

Rule #1 is in conflict with rule #2, rule #3 and rule #4. rule #1 is prevailed by these 3 rules.

Rule #3 is in conflict with rule #5. Rule #5 takes precedence rule #3 since rule #5 has more priority than rule #3.

Rule #2 is in conflict with rule #6. Rule #6 takes precedence rule #2 since rule #6 has more priority than rule #2.
There are the same priority number, that is rule #3 and rule #4, but there is no need for consideration because these two rules are not in conflicting.

Note that: If we want to add another new authorization rule into the existing authorization file, we can add immediately without rearranging or changing the existing authorization rules, although it is in conflict with such existing. For example, rule #7: \((\text{sub17, ob13, grant-read})\) is needed to add into authorization file of example 2. The priority number for rule #7 is 77 that is generated by our Priority Definition. Rule #7 conflict with rule #6, rule #7 takes precedence over rule #6 since rule #7 has more priority than rule #6.

4. Analysis
To analyze the correctness of our specification, we prove the following theorems. Due to space limit, we omit the detail of the proof here but such detail can be found in [11].

**Theorem 1**
Our priority definition can resolve conflicts between any two conflicting authorization rules correctly according to one of the four conflict resolution policies.

**Theorem 2**
Our priority definition can resolve conflicts between any two conflicting authorization rules correctly according to any combination of the four conflict resolution policies.

We prove theorems 1 and 2 by case analysis on each of the four conflict resolution policies and each combination of the four conflict resolution policies, respectively.

**Theorem 3**
Our priority definition can resolve conflicts correctly for any number of the conflicting authorization rules according to any combination of the four conflict resolution policies.

We prove this theorem by using induction on the number of conflicting rules. The base case which is for any two conflicting rules is due to theorem 2. The inductive step holds since any conflict occurred is essentially between any two rules and the result of conflict resolution is a set of priority numbers which is totally ordered.

5. Conclusion
In this paper, we argue that existing approaches to conflict resolution in XML do not offer two important properties which are intuitive conflict resolution policy and simple implementation within the same framework. Then, we propose a simple approach for resolving conflicting authorizations and show that our approach offers both intuitive conflict resolution policy and a simple implementation mechanism within a single framework. Many kinds of conflict resolution policy can be employed in our approach without the need for the new implementation of the mechanism.

**Acknowledgement**
The research reported in this paper is supported by National Research Council of Thailand.

6. References