RDFacl: A Secure Access Control Model Based on RDF Triple

Jaehoon KIM*a), Member and Seog PARK*†b), Nonmember

SUMMARY An expectation for more intelligent Web is recently being reflected through the new research field called Semantic Web. In this paper, related with Semantic Web security, we introduce an RDF triple based access control model having explicit authorization propagation by inheritance and implicit authorization propagation by inference. Especially, we explain an authorization conflict problem between the explicit and the implicit authorization propagation, which is an important concept in access control for Semantic Web. We also propose a novel conflict detection algorithm using graph labeling techniques in order to efficiently find authorization conflicts. Some experimental results show that the proposed detection algorithm has much better performance than the existing detection algorithm when data size and number of specified authorizations become larger.

key words: database security, access control, authorization conflict, RDF/OWL data, Semantic Web

1. Introduction

Recently, we can find some efforts to secure Semantic Web (SW). Qin and Atluri [1] introduced a class level access control considering authorization propagation by various ontology inferences, and Reddivari et al. [2] introduced an Resource Description Framework (RDF) triple based access control model considering various operations in an RDF store. Jain and Farkas [3] introduced an RDF triple based access control considering an authorization conflict problem in RDF inference.

In this paper, we also introduce an access control model based on the RDF triple, which considers SW inference. However, compared with the existing studies, our model is based on explicit authorization propagation over the ontology hierarchy of upper and lower classes or properties. The explicit propagation of authorizations is that when an authorization is specified for an upper class or property, the same access authorization is also applied to all lower classes or properties over the ontology hierarchy by inheritance. This explicit authorization propagation over the ontology hierarchy, which is a graph, is necessary for more convenient authorization specification of a security administrator as in the eXtensible Markup Language (XML) access control [4]–[7] exploiting the authorization propagation over a tree. Through the explicit authorization propagation, a variety of authorization specifications can be done at one time without the need to be done separately and a security administrator can manage a much lower number of authorizations. Our contribution is, first, to introduce an RDF triple based access control model (named RDFacl) supporting the explicit authorization propagation by inheritance as well as the implicit authorization propagation by inference. Next, using these two contrary propagations, we introduce an authorization conflict problem in SW access control. As for the ontology hierarchy, in this paper, we consider the basic subsumption relationships in RDF, subClassOf and subPropertyOf, and further consider the more complex subsumption relationships in OWL like unionOf, intersectionOf, and oneOf.

With regard to detecting the authorization conflict, Jain and Farkas [3] have suggested a somewhat inefficient algorithm where all RDF triples are inspected. Therefore, in this paper, we also propose an efficient conflict detection method which inspects only the formerly specified authorizations rather than all RDF instances. To efficiently detect an authorization conflict under subsumption inferences, our method inspects only the former authorizations having subsumption relationships with a new authorization.

The remainder of the paper is organized as follows. In Sect. 2, we review recent studies related to SW access control. Next, Sect. 3 briefly explains about the RDF triple and the subsumption relationship, and through Sects. 4, 5, and 6 we introduce the RDFacl. Section 7 introduces the suggested authorization conflict detection method, and Sect. 8 presents some experimental results. Section 9 finally concludes this paper.

2. Related Work

XACML [4], Damiani et al. [5] and E. Bertino et al. [6], [7] introduced the fine-grained access control models for XML documents. According to specified access authorizations, element tags and attributes in an XML tree structure are made to be selectively invisible to users. Even though RDF/OWL documents are described in XML, the existing XML access control models are not desirable as they cannot check the security violation by ontology inference. That is, when a set of access authorizations are explicitly specified for RDF and OWL tags, they do not define which authorizations should be applied to inferred information.

Jain and Farkas’ study [3] is the closest work to ours. However, in our study, we exploit the explicit authorization propagation and suggest a more efficient authorization conflict detection method.

Qin and Atluri [1] considers the implicit authorization
Fig. 1  A sample RDF graph.

propagation and authorization conflict problem for various semantic relations in an ontology. However, their access control policy is not based on the RDF triple structure. Hence, their methods are not incorporated with RDF Semantics [8] and OWL Semantics [9]. In addition, the security object in their model is a class, but in our case it is a property which is represented as the RDF triple. Hence we can support the more fine-grained access control.

Kaushik et al. [10] introduces an access control model for the fine-grained information disclosure of an RDF web document. The main point of their study is to introduce a formal framework to provide disclosure control over parts of an ontology. In addition, they introduce applying several methods of information hiding to RDF data, e.g., removing a specific subtree in an ontology tree or renaming a disallowed class or property according to an authorization. However they do not consider the disclosure problem for highly sensitive data by a prohibited inference. In fact, this problem is closely connected with the authorization conflict problem in this paper, because such an information disclosure arises when two authorizations having conflict relationship are both allowed.

In the paper [11], we already introduced the explicit and the implicit authorization propagation, the authorization conflict problem between two propagations, and the conflict detection algorithm. In this paper, we revise some contents in the previous article, add some experimental results which were not shown in the previous article, and particularly we expand such concepts into some subsumption relationships of OWL.

3. RDF and OWL

An RDF document consists of RDF statements, which use ontology concepts defined in an RDF Schema (RDFS). An RDF statement is represented as a triple of \([s, p, o]\), and an RDF document can be represented as a graph consisting of RDF triples. An RDFS statement can also be represented as a triple.

**Definition 1** (RDF graph and triple): An RDF graph is a set of RDF triples. An RDF triple is represented as \([s, p, o]\), where \(s \in \text{SUBJECT}\), \(p \in \text{PREDICATE}\), and \(o \in \text{OBJECT}\).

- The set \(\text{SUBJECT}\) includes URI (Uniform Resource Identifier) nodes defining classes or properties in RDFS and instances in RDF, and blank nodes.
- The set \(\text{PREDICATE}\) includes URI nodes referencing properties in RDFS.
- The set \(\text{OBJECT}\) includes the URI nodes of the other classes and instances related by \(p\), blank nodes, and literals.

For example, in Fig. 1, the RDF triple \([\text{Weapon}, \text{manufacturedBy}, \text{WeaponCompany}]\) has the class URI constant \(\text{Weapon}\) as \(s\), the property URI constant \(\text{manufacturedBy}\) as \(p\), and the class URI constant \(\text{WeaponCompany}\) as \(o\). \([\text{Titan}, \text{NWQuantity}, 127]\) has the instance URI constant \(\text{Titan}\) as \(s\) and the literal \(127\) as \(o\). \([\text{Titan}, \text{locatedIn}, \_]\) has a blank node as \(o\). The RDF part of Fig. 1 represents the sample RDF document of Fig. 2.

OWL is also based on the RDF triple and adds separate vocabularies for defining various relationships between ontology concepts to the primitive vocabularies of RDF. For example, the RDF graph of Fig. 3 (b) connects the following RDF triples for the OWL document of Fig. 3 (a). Refer Sect. 4.1 in the OWL Semantics [9] for the details of mapping OWL to RDF graph.

\[
\begin{align*}
[&\text{NamedPizza}, \text{rdf:type}, \text{owl:Class}] \\
[&\text{NamedPizza}, \text{owl:unionOf}, \_1] \\
[&\_1, \text{rdf:type}, \text{rdf:List}] \\
[&\_1, \text{rdf:first}, \text{ItalianPizza}] \\
[&\_1, \text{rdf:rest}, \_2] \\
[&\_2, \text{rdf:first}, \text{AmericanPizza}] \\
\end{align*}
\]
Fig. 2  A sample RDF web document.

(a) OWL web document

(b) RDF graph

Fig. 3  An RDF graph for an OWL web document.

[2: rdfl:rest, rdfl:nil]

Definition 2 (subsumption relationship): In RDF, if a class \( c_i \) is the subclass of another class \( c_j \) (\( c_i \subseteq c_j \), subClassOf), \( c_i \) and its instances inherit the properties of \( c_j \), and \( c_i \) can be interpreted as \( c_j \) by inference. If a property \( p_i \) is the subproperty of another property \( p_j \) (\( p_i \subseteq p_j \), subPropertyOf), \( p_i \) can be interpreted as \( p_j \) by inference. Similarly, in OWL, if a class \( c_j \) is the union of other classes \( c_i \) and \( c_k \) (\( c_j = (c_i \cup c_k \), unionOf)), \( c_i \) and \( c_k \) can be interpreted as \( c_j \) by inference.

4. Access Authorization

4.1 Security Object

In our authorization specification, security objects are RDF triples. A security administrator can conveniently bind up the target RDF triples into the following security object pattern.

Definition 3 (security object pattern): A security object pattern is also represented as an RDF triple \([s, p, o]\), where \( s \) and \( p \) can be substituted by variables \( $x \) and \( $y \), respectively, and \( o \) is always the variable \( $z \) (In this study, we do not consider much more fine-grained access control according to \( o \) values). Also, a blank node for \( s \) and \( o \) is not allowed.

For example, in the RDF graph of Fig. 1, for the security object pattern \([$x, NWQuantity, $z]\), the matching RDF triples are \([NuclearWeapon, NWQuantity, literal] \) and \([Titan, NWQuantity, literal] \). In particular, the pattern \([$x, $y, $z]\) matches all edges in the graph.

4.2 Specifying Access Authorization

Access authorizations are formally defined as follows.

Definition 4 (access authorization): An access authorization is a five tuple of the form: \(<subj, obj, act, sign, type>\).

- \( subj \) is the subject to whom the authorization is granted.
- \( obj \) is the security object pattern.
- \( act \) refers to an action performed against the security object. Since in this study we consider applying our access control model to the fine-grained information disclosure of RDF/OWL documents over Web, only read operation is considered.
- \( sign \) is \((+)\) if access is allowed, and \((-)\) if access is forbidden.
- \( type \) is \( R \) (= Recursive) if an authorization should be propagated to lower classes or properties by the subsumption relationship, and \( L \) (= Local) if an authorization should not be propagated. We will explain the details of authorization propagation according to \( type \) in Sect. 5.

4.3 Hidden Portions of an RDF/OWL Document according to an Authorization

We consider applying our access control model to information disclosure of RDF/OWL documents published over Web. For example, according to the authorization \(<Dave, [X, manufacturedBy, Z], read, -, R>\), the tag \(<ex:manufacturedBy rdfl:resource = “ex:CentralCo”> in
the sample RDF document of Fig. 2 must be invisible to the subject Dave. In this subsection, we define which portions of an RDF/OWL document must be hidden according to a specified access authorization. The hidden portions are decided by the value type of $o \in \{\text{class or instance URI constant, blank node, literal}\}$ in Definition 1. Table 1 summarizes this.

**Example 1:** According to $\langle\text{Dave}, \lbrack Sx, \text{manufacturedBy}, S\rbrack, \text{read, } -, \text{R}$>, the $p = \text{"ex:manufacturedBy"}$ and the $o = \text{"rdf:resource = "ex:CentralCo"} “ in Fig. 2 are hidden from the instance $s = \text{"Titan"}$. However, the actual instance $\text{CentralCo}$ referenced by the instance URI constant “ex:CentralCo” is not hidden. According to $\langle\text{Dave}, \lbrack Sx, \text{locatedIn}, S\rbrack, \text{read, } -, \text{R}$>, the $p = \text{"ex:NWQuantity"}$ and the literal 127 are hidden.

**Example 2:** According to $\langle\text{Dave}, \lbrack \text{NuclearWeapon}, \lbrack Sx, S\rbrack, \text{read, } -, \text{L}$>, Dave cannot show all properties of the class NuclearWeapon. If all properties of a class or an instance should be invisible, the whole class or instance should be invisible, e.g., in Fig. 2, $\langle\text{ex:NuclearWeapon rdf:ID = “Titan”} \rangle \ldots \langle\text{ex:NuclearWeapon}\rangle$ is hidden.

### 5. Explicit Authorization Propagation Policy

When the type of an authorization is R, the authorization affects lower classes or properties by inheritance. In this section, we first explain the authorization propagation related with the basic subsumption relationships in RDF Semantics [8], subClassOf and subPropertyOf, and then related with some primary subsumption relationships in OWL Semantics [9], unionOf, intersectionOf, and oneOf.

#### 5.1 rdfs:subClassOf Relationship

As explained in Definition 2, if $c_j \subset c_i$, $c_j$ inherits the properties of $c_i$. Therefore, we define that when an authorization $ca_j$ is specified for a property $p_k$ of $c_i$, $ca_j$ also affects the property $p_k$ of $c_i$. We denote this subClassOf propagation as $ca_j + \rightarrow ca_i +$ or $ca_j - \rightarrow ca_i -$. It is natural that the instances of $c_j$ and $c_i$ follow the authorizations $ca_j$ and $ca_i$, respectively.

**Example 3:** When $ca_j = \langle\text{Dave}, \lbrack \text{SpecialWeapon}, \text{SWQuantity}, S\rbrack, \text{read, } -, \text{R}$> is specified, $ca_j$ derives $\langle\text{Dave}, \lbrack \text{NuclearWeapon}, \text{SWQuantity}, S\rbrack, \text{read, } -, \text{R}$> and $\langle\text{Dave}, \lbrack \text{Missile}, \text{SWQuantity}, S\rbrack, \text{read, } -, \text{R}$> by the propagation policy $ca_j - \rightarrow ca_r -$. When $ca_i = \langle\text{Dave}, \lbrack \text{SpecialWeapon}, \lbrack Sx, S\rbrack, \text{read, } -, \text{R}$> is specified, due to $p(ca_j) = Sy$, $ca_j$ is also applied to lower classes inheriting all properties of SpecialWeapon. That is, $ca_j$ derives the following authorizations: $\langle\text{Dave}, \lbrack \text{NuclearWeapon}, \text{SWQuantity}, S\rbrack, \text{read, } -, \text{R}$>, $\langle\text{Dave}, \lbrack \text{Missile}, \text{SWQuantity}, S\rbrack, \text{read, } -, \text{R}$>, and $\langle\text{Dave}, \lbrack \text{NuclearWeapon, locatedIn}, S\rbrack, \text{read, } -, \text{R}$>, and $\langle\text{Dave}, \lbrack \text{Missile, locatedIn}, S\rbrack, \text{read, } -, \text{R}$>.

**Definition 5 (* pattern):** This pattern is represented as $\lbrack S, \ast, \ast \rbrack$ as in the above example. This is a special security object pattern reserved for conveniently matching all properties of $s$’s lower classes as well as $s$.

#### 5.2 rdfs:subPropertyOf Relationship

We define that if $p_i \subset p_j$, an authorization $pa_i$ for $p_j$ also affects the property $p_i$. We denote this subPropertyOf propagation as $pa_i + \rightarrow pa_j +$ or $pa_i - \rightarrow pa_j -$. Also, instances having the property $p_j$ and $p_i$ follow the authorizations $pa_j$ and $pa_i$, respectively.

**Example 5:** When $pa_j = \langle\text{Dave}, \lbrack \text{ConventionalWeapon}, \text{CQuantity}, S\rbrack, \text{read, } -, \text{R}$> is specified, $pa_j$ also derives $\langle\text{Dave}, \lbrack \text{Rifle}, \text{RQuantity}, S\rbrack, \text{read, } -, \text{R}$>. In the case of $pa_j = \langle\text{Dave}, \lbrack \text{SpecialWeapon}, \lbrack Sy, S\rbrack, \text{read, } -, \text{R}$>, $pa_j$ also affects all subproperties of all properties of SpecialWeapon. For the security object * pattern $\langle\text{Dave}, \lbrack \text{SpecialWeapon}, \ast, \ast \rbrack, \text{read, } -, \text{R}$>, $pa_j$ also affects all subproperties of all properties of all subclasses of SpecialWeapon.

#### 5.3 owl:unionOf Relationship

The above authorization propagation for the basic subsumption relationships can be easily expanded into the more complex subsumption relationships in OWL. First, let us consider the unionOf relationship. As in Definition 2, this relationship states that a subsuming class contains all instances of the subsumed classes. Therefore, when $c_j = (c_i \text{ or } c_k)$, we can consider the explicit propagation $ca_j \rightarrow ca_i$ as in the subClassOf relationship.
Example 6: Figure 3 (b) shows the RDF graph for the OWL document of Fig. 3 (a) (Refer Sect. 4.1 in the OWL Semantics [9] for mapping OWL to RDF graph). When $ca_j = <Dave, [NamedPizza, *, *], read, −, R>$ is specified, due to $p(ca_j) = *$, $ca_j$ is also applied to the subclassed classes. That is, $ca_j$ derives the following authorizations: $<Dave, [ItalianPizza, $y$, $z$], read, −, R>$ and $<Dave, [AmericanPizza, $y$, $z$], read, −, R>$. This propagation can be similarly applied to the $owl:oneOf$ relationship which specifies the members of a class are exactly the set of enumerated individuals.

5.4 $owl:intersectionOf$ Relationship

The $intersectionOf$ relationship states that a subclassed class is exactly the intersection of the subclassing classes. Therefore, when $c_j = (c_i ∩ c_k)$, we can consider the explicit propagation $ca_i → ca_j$. Note that since the instances of $c_j$ are shared by the $c_i$ and $c_k$, $c_j$ becomes the subclassing class and $c_i$ becomes the subclassed class contrary to the $unionOf$ relationship.

Example 7: In Fig. 3, when $ca_i = <Dave, [SpicyPizza, *, *], read, −, R>$ is specified, $ca_{i→}$ is also applied to the subclassed class $SpicyAmericanPizza$. That is, $ca_{i→}$ derives the authorization $<Dave, [SpicyAmericanPizza, $y$, $z$], read, −, R>$.

6. Implicit Authorization Propagation Policy and Authorization Conflict Problem

In this section, we explain the implicit authorization propagation by ontology inference, and analyze the authorization conflict problem between the explicit propagation and the implicit propagation.

6.1 $rdfs:subClassOf$ Inference

The authorization conflicts in SW access control can be classified into two types. One is the explicit authorization conflict and another is the implicit authorization conflict. The explicit authorization conflict addresses that there are several authorizations explicitly having different $sign$ values for the same security object. On the contrary, the implicit authorization conflict addresses that although there are no explicit authorization conflict, a conflict can occur due to ontology inference. Since the explicit authorization conflict is a trivial problem, we concentrate on the explicit authorization conflict in this paper.

- $c_i ⊆ c_j$, $ca_{i→} → ca_{j→}$, $ca_{j→}'$ $⇒$ $ca_{j→}'$ (conflict): This is the unique condition for the implicit authorization conflict in the $subClassOf$ inference. Let us consider the RDF graph of Fig. 4 where $c_i ⊆ c_m ⊆ c_j$, and the authorization propagation of the first row in the table. When the authorization $ca_{j→}$ is first specified for $pt_j = [c_j, p_k, $z$]$, the explicit authorization propagation arises: $ca_{j→} → ca_{m→} → ca_{j→}$. However, when the authorization $ca_{j→}'$ is afterwards specified for $pt_{m} = [c_j, p_k, $z$]$, there is the implicit authorization propagation $ca_{i→}'$ $⇒$ $ca_{m→}'$ $⇒$ $ca_{j→}'$. This is because the security object $pt_j$ can be inferred from $pt_{m}$ by the $subClassOf$ inference. Since $sign(ca_{j→}') ≠ sign(ca_{j→})$, an authorization conflict occurs. Similarly, when $c_i ⊆ c_j$, $ca_{j→} → ca_{j→}$, and a new authorization $ca_{j→}$ with $type = L$ is specified afterwards, there is also a conflict. Remind of that $type = L$ indicates there is no explicit authorization propagation by the Definition 4. Therefore, just the $ca_{j→}$ is overwritten by $ca_{j→}$.

Example 8: $ca_j = <Dave, [Weapon, manufacturedBy, $z$], read, −, R>$ drives $ca_i = <Dave, [NuclearWeapon, manufacturedBy, $z$], read, −, R>$ by the explicit authorization propagation. Then suppose that the authorization $ca_{i→}' = <Dave, [NuclearWeapon, manufacturedBy, $z$], read, +, R>$ is re-specified. Since $[Weapon, manufacturedBy, $z$] can be inferred from $[NuclearWeapon, manufacturedBy, $z$]$, $ca_{i→}'$ must be also applied to $[Weapon, manufacturedBy, $z$]$. That is, $ca_{j→}'$ drives $ca_{j→}' = <Dave, [Weapon, manufacturedBy, $z$], read, +, L>$ by the implicit authorization propagation. Since $sign(ca_{j→}') ≠ sign(ca_{j→})$, this is conflict. Here, note that the $type (= ‘R’ or ‘L’) is just related with the explicit authorization propagation by the Definition 4. The $type$ has nothing to do with the implicit authorization propagation. Therefore, we simply represent that all of the implicitly propagated authorizations have just the type ‘L’ regardless of the ‘L’ or ‘R’ of the specified authorization.

- $c_i ⊆ c_j$, $ca_{i→} → ca_{j→}$, $ca_{j→}'$ $⇒$ $ca_{j→}'$ (conflict-free): In the same manner, as for the second row of the table in Fig. 4, let us consider $ca_{i→} → ca_{m→} → ca_{j→}$ and a new authorization $ca_{j→}'$ specified afterwards for $pt_{m} = [c_j, p_k, $z$]$. In this case, since $ca_{j→}'$ disallows accessing $pt_{m}$, any related inference cannot occur. Hence, this case is conflict-free.

- $c_i ⊆ c_j$, $ca_{j→} → ca_{j→}$, $ca_{j→}'$ $⇒$ $ca_{j→}'$ (conflict-free): Let us consider $ca_{j→} → ca_{j→} → ca_{j→}$ and a new authorization $ca_{j→}'$ specified afterwards as in the third row of the table.

| Conflict | Explicit propagation | Implicit propagation |
|----------|----------------------|---------------------|
| $ca_{i→} → ca_{j→}$ | $ca_{i→}'$ $⇒$ $ca_{j→}'$ | $ca_{i→}'$ $⇒$ $ca_{j→}'$ |
| $ca_{i→} → ca_{j→}$ | $ca_{i→}'$ $⇒$ $ca_{j→}'$ | $ca_{i→}'$ $⇒$ $ca_{j→}'$ |
| $ca_{i→} → ca_{j→}$ | $ca_{i→}'$ $⇒$ $ca_{j→}'$ | $ca_{i→}'$ $⇒$ $ca_{j→}'$ |
| $ca_{i→} → ca_{j→}$ | $ca_{i→}'$ $⇒$ $ca_{j→}'$ | $ca_{i→}'$ $⇒$ $ca_{j→}'$ |

Fig. 4 Authorization conflict in $subClassOf$ inference.
As in the previous case, since $ca_{i−}′$ disallows accessing $pt_i$, there can be no conflict.

- $c_i \subseteq c_j$, $ca_{j−} \rightarrow ca_{i−}$, $ca_{i−}′ \Rightarrow ca_{j−}′$ (conflict-free): Let us consider $ca_{j−} \rightarrow ca_{m−} \rightarrow ca_{i−}$ and a new authorization $ca_{i−}′$ specified afterwards as in the fourth row of the table. In this case, $ca_{i−}′ \Rightarrow ca_{j−}′$ and $sign(ca_{i−}′) \equiv sign(ca_{j−}′)$, there is no conflict.

6.2 rdfs:subPropertyOf Inference

When $p_i \subseteq p_j$, $pa_{j−} \rightarrow pa_{i−}$, and $pa_{i−} \Rightarrow pa_{j−}′$ as in the subClassOf inference, there is a conflict. Figure 5 depicts this situation. For example, $pa_{i−} = <Dave, [Weapon, Quantity, $y$, $z$], read, −, $R$> drives $pa_{j−} = <Dave, [Weapon, Quantity, $y$, $z$], read, −, $R$> as $NWQuantity$ is the subproperty of $Quantity$. If the authorization $pa_{i−}′ = <Dave, [Weapon, NWQuantity, $y$, $z$], read, +, $R$> is specified afterwards, this is a conflict. This is because $[Weapon, Quantity, $z$] can be inferred from $[Weapon, NWQuantity, $z$] by the subPropertyOf inference and $sign(pa_{i−}′) \neq sign(pa_{j−}).$

6.3 owl:unionOf and owl:intersectionOf Inferences

First, when $c_j = (c_i \cup c_k)$, $ca_{j−} \rightarrow ca_{i−}$, and $ca_{i−} \Rightarrow ca_{j−}′$, the unionOf inference also has a conflict. For example, suppose that $ca_{i−}′ = <Dave, [ItalianPizza, $y$, $z$], read, −, $L$> is specified afterwards for the Example 6. Since the instance of ItalianPizza is also the instance of NamedPizza, $ca_{i−}′ \Rightarrow ca_{j−}′$ and $sign(ca_{i−}′) \neq sign(ca_{j−}′)$.

We can also consider the same conflict related with the oneOf inference.

As for the intersectionOf inference, when $c_j = (c_i \cap c_k)$, $ca_{j−} \rightarrow ca_{j−}$, and $ca_{j−} \Rightarrow ca_{i−}′$, there also is a conflict. For example, suppose $ca_{j−}′ = <Dave, [S picyAmericanPizza, $y$, $z$], read, +, $L$> is specified afterwards for the Example 7, $ca_{i−}′ \Rightarrow ca_{i−}$ and $sign(ca_{i−}′) \neq sign(ca_{i−})'$.

7. Efficiently Detecting Authorization Conflict Using Graph Labeling

Jain and Farkas [3] introduced a little inefficient algorithm for detecting the authorization conflict. In their method, whenever a new authorization is specified, the corresponding security labels are first assigned to all RDF triples. Then RDF inference is performed for all RDF triples, and for each inferred triple, it is checked if there is a security violation. That is, their method simply checks all RDF instances. This is inefficient when the number of instances become larger. Therefore, we suggest an efficient conflict detection method using graph labeling techniques [12], [13]. The basic idea is, based on the observation in the previous section, to check only the ancestor authorizations with $sign(−)$ when a new authorization with $sign(+)$ is specified whereas to check only the descendant authorizations with $sign(+)$ when a new authorization with $sign(−)$ and type $L$ is specified. Here, the ancestor/descendant authorization means an authorization of which the security object has subsumption relationship with the security object of the new authorization. For example, $a_i = <Dave, [S pecialWeapon, $y$, $z$], read, −, $L>$ is the ancestor authorization of $a_j = <Dave, [NuclearWeapon, $y$, $z$], read, +, $L$> because $s(a_i) = S pecialWeapon$ is an ancestor node of $s(a_j) = NuclearWeapon$ in the RDF graph of Fig. 1. In order to efficiently identify the ancestor/descendant relationship, we use the graph labeling techniques [12], [13]. In this paper, we skip the details of the graph labeling techniques. We only discuss how we can detect efficiently an authorization conflict using the information of the ancestor/descendant relationship.

The suggested detection algorithm of Fig. 8 selectively tests the cases of authorization conflict according to the conflict decision table (CDT) of Figs. 6 (a) and (b). The CDT summarizes the possibility of conflict according to the type of $s$ and $p$ values in Definition 1, 3, and 5: $s \in$ (class URI constant, instance URI constant) and $p \in$ (property URI constant, $*$). In the case of $s = x$, we can get the URI constant of the highest upper class having the property $p$. For example, since all classes have the property WCode but the highest upper class is Weapon, $s(x, WCode, $z$) = Weapon$. We now prove the CDT through the following examples. Note that since we have shown the subsumption relationships in OWL are similar to the subClassOf in Sects. 5 and 6, it is proven around the subClassOf relationship.

Example 9: Let us consider that an authorization $R5 = <Dave, [NuclearWeapon, $y$, $z$], read, +, $L$> is additionally specified against the authorizations of Fig. 7. In this case, $R1$ is the only authorization which can have conflict with $R5$. Because $S pecialWeapon$ is the ancestor class for NuclearWeapon, $R1$ has ($−$) sign, and $R5$ has (+) sign. Also, since $s(R5) \in$ class URI, $s(R1) \in$ class URI, $p(R5) = y$, and $p(R1) = y$, this is absolutely conflict according to the rule 1 in Fig. 6 (a). This is because "p(R5) = y" means that $R5$ is also specified for all properties inherited from $S pecialWeapon$. This example also illustrates the rules 3, 13, and 15.

Example 10: Again, let us consider a new authorization $R5 = <Dave, [ConventionalWeapon, CWQuantity, $z$],
ConventionalWeapon. This example also illustrates the rule in other rule in Fig. 6 (a): sign\( R \). In this example, the final decision is conflict because the property \( \text{locatedIn} \) is inherited from \( \text{SpecialWeapon} \). However, if the property is not inherited, that is conflict-free. This example also illustrates the rule 27. Similarly, in the case of the rule 29, it is required to check whether or not the two property URIs are the same. If equal, that is conflict, otherwise conflict-free.

Example 12: First, let us consider a new authorization \( R5 = \langle \text{Dave}, [\text{Titan}, \text{Sy}, \text{Sz}], \text{read}, +, \text{L}\rangle \) and the former authorization \( R1 \). Since \( \text{sign}(R1) = -, \text{sign}(R5) = +, s(R1) \in \text{class URI}, s(R5) \in \text{instance URI}, p(R1) = \text{Sy}, \) and \( p(R5) \in \text{property URI} \), this is absolutely conflict according to the rule 7 in Fig. 6 (a). This is because the instance \( \text{Titan} \) certainly inherits all properties from \( \text{SpecialWeapon} \). Next, let us consider \( R5 \) and \( R4 \). In this case, since \( \text{Titan} \neq \text{Tomahawk} \), that is, two instances are each other different objects, this is absolutely conflict-free. Furthermore, since an instance which is represented by URI must be unique in its web ontology, an arbitrary authorization pairs both having an instance URI for \( s \) is absolutely conflict-free. Theorem 1 represents this characteristic of the instance URI which makes the authorization conflict detection more simplified.

Theorem 1: (1) A descendant authorization, which has the \( s \) value of the instance URI type and \(+\) sign, can have conflict with only ancestor authorizations which has the \( s \) value of the class URI type and \( -\) sign. (2) An ancestor authorization with the \( s \) value of the instance URI type is absolutely conflict-free with any descendant authorization.

Proof: (1) Since the URI value of an instance is unique in its web ontology, a descendant authorization \( a_i \) with \( s(a_i) \in \text{instance URI} \) and \( \text{sign}(a_i) = + \) can have conflict with an ancestor authorization \( a_j \) with \( s(a_j) \in \text{class URI} \) and \( \text{sign}(a_j) = - \). (2) Next, since an instance can be interpreted into only the instance of its upper class by the subsumption inference, an ancestor authorization \( a_j \) with \( s(a_j) \in \text{instance URI} \) cannot have conflict with any descendant authorization \( a_i \).

By the case (1) in Theorem 1, the rules 8, 10, 12, 20, 22, 24, 32, 34, 36, 44, 46, 48, 56, 58, 60, 68, 70, and 72 in the CDT are conflict-free. And by the case (2) in Theorem 1, the rules 2, 4, 6, 14, 16, 18, 26, 28, 30, 38, 40, 42, 50, 52, 54, 62, 64, and 66 are conflict-free.

Example 13: In the \( \text{subPropertyOf} \) CDT of Fig. 6 (b), all cases are conflict except the cases corresponding to Theorem 1. The \( \text{subPropertyOf} \) CDT does not have the cases of conflict-free for a non-inherited property as in Example 11. This is because the \( \text{subPropertyOf} \) relationship directly defines the subsumption relationship among properties rather than classes. For example, let us consider a new authorization \( R5 = \langle \text{Dave}, [\text{NuclearWeapon}, \text{NWQuantity}, \text{Sz}], \text{read}, +, \text{L}\rangle \). By the rule 25 of the \( \text{subClassOf} \) CDT, this requires conflict verification against \( R1 \), but by the rule 61 of the \( \text{subPropertyOf} \) CDT, this is absolutely conflict. Because although \( s(R1) = \text{Sy} \), it means the property \( \text{NWQuantity} \) in the \( \text{subPropertyOf} \) relationship (See the RDF graph of Fig. 1). A new authorizations \( R6 = \langle \text{Dave}, [\text{Sx}, \text{WCode}, \text{Sz}], \text{read}, -, \text{L}\rangle \) does not need to be checked for the \( \text{subPropertyOf} \) CDT. It should be checked only for...
Algorithm Method1
Input: An authorization $a_i$ to be specified
Output: Conflict authorization set $ConflictSet$

1 /* check the ancestor authorizations with (-) sign in the subClassOf, unionOf, intersectionOf, or oneOf relationships*/
2 if $sign(a_i) = '+'$ then
3    $RS :=$ retrieve the ancestor authorizations with (-) sign in the subClassOf, unionOf, intersectionOf, or oneOf relationships;
4    if $RS$ is not empty then
5       foreach $rs_i \in RS$ do
6          if ($\{(p(s_i) \in \text{property URI}) \land p(r_i) = S_y \land p(a_i) \notin \text{properties}(s_i)\} \lor \{(p(a_i), p(r_i) \in \text{property URI} \land p(a_i) \neq p(r_i))\}$)
7             /* this checks the case of Example 11 */
8             $Conflict-free; continue;$
9       else
10          if $s(r_i)$ in instance URI then $Conflict-free; continue;$ /* this checks the case of Theorem 1 */
11          else
12             $Conflict;
13             $ConflictSet := ConflictSet \cup rs_i$;
14       end
15    end
16 end /* check the ancestor authorizations with (-) sign in the subClassOf, unionOf, intersectionOf, or oneOf relationships*/
17
18 if $sign(a_i) = '-'$ and $type(a_i) = 'L'$ then
19    $RS :=$ retrieve the descendant authorizations with (+) sign in the subClassOf, unionOf, intersectionOf, or oneOf relationships;
20 if $RS$ is not empty then
21    foreach $rs_i \in RS$ do
22       if ($\{(p(s_i) \in \text{property URI}) \land p(r_i) = S_y \land p(a_i) \notin \text{properties}(s_i)\} \lor \{(p(a_i), p(r_i) \in \text{property URI} \land p(a_i) \neq p(r_i))\}$)
23          /* this checks the case of Example 11 */
24          $Conflict-free; continue;$
25       else
26          if $s(r_i)$ in instance URI then $Conflict-free; continue;$ /* this checks the case of Theorem 1 */
27          else
28             $Conflict;
29             $ConflictSet := ConflictSet \cup rs_i; authorizationID$
30       end
31    end
32 end /* check the ancestor authorizations with (-) sign in the subPropertyOf relationship*/
33
34 if $sign(a_i) = '+'$ then
35    $RS :=$ retrieve the ancestor authorizations with (-) sign in the subPropertyOf relationship;
36 if $RS$ is not empty then
37    foreach $rs_i \in RS$ do
38       if $s(r_i)$ in instance URI then $Conflict-free; continue;$ /* this checks the case of Theorem 1 */
39       else
40          $Conflict;
41          $ConflictSet := ConflictSet \cup rs_i; authorizationID$
42    end
43 end /* check the ancestor authorizations with (-) sign in the subPropertyOf relationship*/
44
45 if $sign(a_i) = '-'$ and $type(a_i) = 'L'$ then
46    $RS :=$ retrieve the descendant authorizations with (+) sign in the subPropertyOf relationship;
47 if $RS$ is not empty then
48    foreach $rs_i \in RS$ do
49       if $s(r_i)$ in instance URI then $Conflict-free; continue;$ /* this checks the case of Theorem 1 */
50       else
51          $Conflict;
52          $ConflictSet := ConflictSet \cup rs_i; authorizationID$
53    end
54 end /* check the descendant authorizations with (+) sign in the subPropertyOf relationship*/
55
56 return $ConflictSet$;

Fig. 8 Our suggested authorization conflict detection algorithm.

the subClassOf CDT. Because $p(R6) = WCode$ is not included in the subPropertyOf relationship.

8. Experiments

8.1 Experimental Setup

In this section, we compare our detection method with Jain and Farkas’ method [3]. Since we could not obtain the optimized implementation of Jain and Farkas’ method, we simulated it as follows. First, a DAG as in Fig. 9 is generated according to the experimental parameter #C, #P, and #S in Table 2. The circle represents a class and the rectangle represents a property. The parameter #S is the average number of subClassOf relationships for each class. For example, in Fig. 9, since c3 is the subclass of c1 and c4, its #S is two. In this experiment, we simplified the inferences related with unionOf and intersectionOf. They can be simply regarded as a set of subClassOfs having the OR and AND arch as in the DAG. For example, c2 = (c6 ∪ c7) and c8 = (c3 ∩ c4 ∩ c5). In the data structure of properties, the variable before_sign stores the sign value assigned by a formerly specified authorization and the variable after_sign stores the sign value by the currently specified authorization. In the data structure of classes, each class has storage spaces for all inherited properties as well as its own properties. For
example, all lower classes of c1 have the property p1.

Next, whenever a new authorization is inputted, the following conflict check is performed for Jain and Farkas’ method. First, with a breadth-first traversal of the DAG, the sign values of the formerly specified authorizations are assigned to all variables of before_sign by the explicit authorization propagation. This step makes all RDF triples have their own most specific security sign value. Then the sign value of the new authorization is assigned to some corresponding variables of after_sign also with a breadth-first traversal. Again, with a breadth-first traversal, it is checked if there is any property with before_sign = ‘−’ and after_sign = ‘+’. If such a property exists, it is a conflict.

Our suggested algorithm also uses the randomly generated DAG and authorizations. As a graph labeling technique, we used the prime number labeling scheme suggested by Wu et al. [13]. All experiments were performed on a Windows XP computer with 1 GB of memory and 3.20 GHz Pentium(R) IV CPU; all codes were written in Java.

8.2 Experimental Results

We experimented according to the parameters in Table 2. Having higher values of #C and #S means that the DAG becomes larger, that is, the size of RDF/OWL schema and data becomes larger and more complex. Also, having a higher value of #A means that there are much more authorizations to be checked against the authorization conflict. First, the graph of Fig. 10 shows the conflict detection time according to #A when #C = 200 and #S = 20. In the experiment, a new authorization was randomly generated for a test DAG, and its authorization conflict check was performed against the formerly specified authorizations. If there is no conflict, the new authorization is added to the former authorization set. This procedure was performed as much as the parameter #A. The graph shows that our RDFacl has much lower detection time than Jain and Farkas’ method according to #A. We can also see that the difference becomes significant as the number of authorizations increases.

Next, the graph of Fig. 11 shows the conflict detection time according to #C when #A = 500 and #S = 2. It also shows that our RDFacl has significantly lower detection time and lower increasing rate. Through some additional experiments which vary the parameter values, we also confirmed that there is a significant difference between two methods. Although our experimental results are approximate due to the simulation of Jain and Farkas’s method, the significant difference apparently shows that our suggested method has more improved detection capability than the existing method.

Although in this paper we omitted showing how our method stores access authorizations with graph labeling, regarding the efficiency of memory usage, our suggested method spends storage space approximately twice as much as the simulated Jain and Farkas’s method. We believe that the additional storage cost is not significant considering the benefit of the improved detection time. In our implementa-

---

### Table 2

| Parameter | Range  | Description                  |
|-----------|--------|------------------------------|
| #C        | 1 to 1,000 | Number of classes in a DAG   |
| #P        | 1 to 5 | Average number of properties for each class in a DAG |
| #S        | 1 to 20 | Average number of subClassOf for each class in a DAG |
| #A        | 1 to 500 | Number of specified authorizations |

---

![Fig. 9 Test DAG generation.](image-url)

![Fig. 10 Detection time comparison according to #A when #C = 200 and #S = 20.](image-url)

![Fig. 11 Detection time comparison according to #C when #A = 500 and #S = 2.](image-url)
tion, in order to maintain the information related with graph labeling, an auxiliary authorization table is separately required besides the main authorization table in the simulated Jain and Farkas’s method. Table 3 shows the additional storage cost per one access authorization in our method.

9. Conclusions and Future Work

The RDF authorization conflict problem is an important problem in RDF access control because RDF data are related with ontology inference unlike XML data. In this paper, we have explained the RDF authorization conflict problem based on two concepts of the explicit and the implicit authorization propagation.

The key ontology inference in RDF is related with the subsumption relationships, subClassOf and subPropertyOf. Therefore, in this paper, we have focused on analyzing the authorization conflict problem in the subsumption inference. We also have shown that the analysis results can be naturally applied to the primary subsumption relationships in OWL, unionOf, intersectionOf, and oneOf.

Currently, our future work is to expand the suggested RDFacl into more complex ontological relationships in OWL [14], e.g., Restriction, equivalentClass, sameAs, equivalentProperty, and complementOf, and also analyze the authorization conflict problem in related inferences.

Acknowledgments

We would like to thank the anonymous referees for their valuable comments on earlier draft of this paper.

This study is supported in part by the Second Stage of BK21. In addition, this work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MOST) (No. R01-2006-000-10609-0).

References

[1] L. Qin and V. Atluri, “Concept-level access control for the semantic Web,” Proc. ACM Workshop on XML Security 2003, pp.94–103, Oct. 2003.
[2] P. Reddivari, T. Finin, and A. Joshi, “Policy-based access control for an RDF store,” Proc. Policy Management for the Web Workshop, pp.78–83, May 2005.
[3] A. Jain and C. Farkas, “Secure resource description framework: An access control model,” Proc. 11th ACM Symposium on Access Control Models and Technologies, pp.121–129, June 2006.
[4] T. Moses, “OASIS extensible access control markup language (XACML) Version 2.0,” OASIS Standard, Feb. 2005.
[5] E. Damiani, S.D.C. Vimercati, S. Paraboschi, and P. Samarati, “A fine-grained access control system for XML documents,” ACM Trans. Information and System Security, vol.5, no.2, pp.169–202, 2002.
[6] E. Bertino, S. Castano, E. Ferrari, and M. Mesi, “Specifying and enforcing access control policies for XML document sources,” World Wide Web Journal, vol.3, no.3, pp.139–151, 2000.
[7] E. Bertino and E. Ferrari, “Secure and selective dissemination of XML documents,” ACM Trans. Information and System Security, vol.5, no.3, pp.290–331, 2002.
[8] RDF Semantics, W3C Recommendation, http://www.w3.org/TR/rdf-mt/
[9] OWL Web Ontology Language Semantics and Abstract Syntax, W3C Recommendation, http://www.w3.org/TR/owl-semantics/
[10] S. Kaushik, D. Wijesekera, and P. Anmammad, “Policy-based dissemination of partial Web-ontologies,” Proc. 2005 Workshop on Secure Web Services, pp.43–52, Nov. 2005.
[11] J. Kim and S. Park, “An RDF triple based access authorization specification considering subsumption relationships,” Proc. First International Workshop on Secure Semantic Web, pp.462–469, April 2008.
[12] V. Christophides, G. Karvounarakis, D. Plexousakis, M. Scholl, and S. Tourounis, “Optimizing taxonomic semantic Web queries using labeling schemes,” Journal of Web Semantics, vol.11, no.1, pp.207–228, Nov. 2003.
[13] G. Wu, K. Zhang, C. Liu, and J. Li, “Adapting prime number labeling scheme for directed acyclic graphs,” Proc. Database Systems for Advanced Applications (DASFAA), pp.787–796, April 2006.
[14] OWL Web Ontology Language Overview, W3C Recommendation, http://www.w3.org/TR/owl-features/

Jaehoon Kim received the B.S. and M.S. degrees in computer science from Konkuk University in 1997 and 1999, respectively and the Ph.D. degree in computer science from Sogang University in 2005. He worked as a senior researcher at Telecommunication R&D center in Samsung Electronics Company from 2005 to 2006. From 2006, he is a research professor of computer science at Sogang University. His major research areas are Web database, database security, data privacy, Semantic Web, and data mining.

Seog Park received the B.S. degree in computer science from Seoul National University, Korea, in 1978, the M.S. and the Ph.D. degrees in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 1980 and 1983, respectively. He is a professor of computer science at Sogang University, Seoul, Korea. Since 1983, he has been working in the Department of Computer Science of the College of Engineering, Sogang University. His major research areas are database security, realtime systems, data warehouse, digital library and web database. Dr. Park is a member of the IEEE Computer Society, ACM, and the Korea Information Science Society. Also, he has been a member of Database Systems for Advanced Applications (DASFAA) steering committee since 1999.