SCHEMA BASED XML SECURITY: RBAC APPROACH

Xinwen Zhang, Jaehong Park, and Ravi Sandhu
George Mason University
{xzhang6, jpark2, sandhu)}@gmu.edu

Abstract Security of XML instance is a basic problem, especially in enterprise with large number of users and XML objects as well as complex authorizations administration. In this paper, a role-based access control (RBAC) model based on XML Schema is proposed. RBAC has been proven to be efficient to improve security administration with flexible authorization management. XML Schema is a specification to define format and contents of XML instance. Access control based on a schema will be transported to all its instances. As a proposed alternate of XML Document Type Definition (DTD), XML Schema supports complex constraints for XML components, such as elements, attributes, datatypes and groups. Also, XML Schema provides a mechanism to build rich reuse relationships between schemas and elements. These will be applied in reusable permissions in our model, which efficiently simplify the security administration. Based on these features fine-grained access control can be achieved. At the same time, our model also supports instances-level authorization naturally, which provides a uniform mechanism for XML security. A abstract implementation is presented in this paper for our proposed model. "Pure" XML technologies will be applied in the implementation mechanism, which make the system lightweight and can be easily embedded into existing systems.

1. Introduction

Because of its platform-independent characteristics, XML [9] has been increasingly used in many environments to integrate applications and communicate between systems. XML instance is a structured format with meta-data for real data. Because of the ability to express complex reference relationship between data, a XML instance may be generated from various resources with varying security requirements. On the other side, a user can be allowed to access only particular parts of a XML instance. For example in an enterprise, a XML document can consist of information from applications among a few departments and several databases. When an internal or external user tries to access this document, his/her access rights have to be monitored according to security policies in all these departments and databases. The final instance which the user can read or modify is the result of enforcement with overall authorizations.

In this paper, an access control model is proposed to control all accesses to XML instances in such distributed and heterogeneous environment. In an enterprise or organization, there are large number of users and XML objects. Also, there are complex relationships among users, objects, and arbitrary authorizations between users and objects. Each user has identification and attributes. An object can be a XML document, message, dynamically generated XML instance, or any XML elements. Because of the complex data sources with different security policies, authorizations management will be burdensome. Role-based access control model [1, 2] has proven to be efficient in security administration with roles providing a layer of indirection between users and objects. The role-permission assignment is comparatively stable while user-role assignment can be more dynamic. At the same time, RBAC provides strong data type abstraction ability. All the components in the model can be customized and fit into particular applications very easily.

XML Schema [11, 12] is a mechanism to define the content and relationship of elements in an XML instance. A well-validated XML document must follow the format specified by one or several schemas. In our proposed model, permissions a user can invoke are defined on schema or schema element level and will be transported to all XML instances specified by these schema or elements. The permission on a schema component will be transported to all XML instance data which is specified by this component. At the same time, substantial permission reuse can be generated based on the rich relationships between elements, datatypes and attributes in a schema, or between schemas. We will use these relationships to build permission reuse hierarchy. Based on this, fine-grained, flexible, and easy-customized access control model can be achieved. With the unique features of XML Schema, extensible, modular and reusable security policies can be generated in distributed environment.

The remainder of this paper is organized as follows: Section 2 presents the background of XML Schema and the main difference from DTD. Section 3 describes and discusses the model in details. Section 4 briefly presents the high-level implementation of the access control model. Section 5 reviews the previous work on XML document security. The difference between this work to others is presented. Section 6 concludes the paper and outlines the future work to continue on this topic.

2. XML and XML Schema

XML instance has two basic requirements: well-formed and validate-formed. Well-formalization requires XML document to follow some syntax, such as, there is exactly one element that completely contains all other elements, elements may nest but not overlap, etc. Validation requires XML instance to contain specified elements and attributes, following specified datatypes and relationships. Document Type Definition (DTD) and XML Schema are two main validation specification mechanisms.

Document Type Definition(DTD) is the first and earliest language to define the structure and content of XML documents. But it has many limitations which are critical in enterprise and distributed environments. A DTD file is not itself a well-formed and valid XML document. The rules in DTD are not meta-data, but rather some special formats to show the order of elements. The problem with this is that a special process is needed for an XML parser to parse the content in DTD. Another problem is that it is difficult to specify constraints on structure and content of XML instance with DTD. Actually, DTD only specifies the appearance order of element and its subelements and attributes, but cannot define complex relationships and constraints. DTD cannot define datatypes, which make it difficult to be reusable, extensible, and modular. A defined DTD cannot be used by other DTDs, and rules in DTD cannot be reused and extended by other rules within or out of this DTD. All these limitations prevent DTD from being widely applied in distributed and scalable systems. XML Schema is an alternate in modern enterprise environments with some new features. XML Schema is XML document itself, which XML parser can handle just like normal XML instance. Therefore, XPath and XQuery can be applied to specify fine-grained schemas objects. Complex user-defined datatypes can be created in XML Schema. Rich description and relations of schemas and components can be expressed. Hierarchy can be established based on these relationships. This makes schema reusable and extensible. Namespace is supported in XML Schema to solve name conflictions. This helps modular deployment of security administration in our model.

With these reasons, modern XML specifications are all based on schema. At the same time, the improvement of XML Schema results in flexible schema-based access control policy. With DTD's limitations, permission based on DTD is not modular, extensible, and reusable. The access control policy on XML instance documents and DTD have to be implemented separately, since DTD is not XML well-formed and valid-formed. By using schema, we can define and enforce the permissions on schema objects and instance objects with uniform mechanism. Also, in distributed environment, the authorizations from various services and departments can be assembled easily with pure XML technologies, since policy based on schemas is extensile and reusable with schema's characteristics.

3. Extended RBAC Model

Role-based access control is a policy-neutral model studied by many researchers during the past decade. The flexibility and efficiency in security management make it an attractive access control solution in many commercial systems. The main components of RBAC96 model includes users, roles, role-hierarchy, permissions, user-role assignments and permission-role assignment [1].

Figure 1 shows our proposed model extended from original RBAC96 model. The users, roles, role hierarchy, user-role assignment and sessions are the same as that of RBAC96 model. Instead of direct assignment of roles and final permissions, in this model, there are schema-based permissions (*SP*) and explicit role-permission assignments *(EPA)* between roles and schema objects. *SP* are defined by associating some atomic access types with schema components. *EPA* is the assignment between roles and *SP*. By the instance mapping (*IM*) function from schema objects *(SO)* to instance objects (*IO*), *SP* and *EPA* imply the instance-based permission *(IP)* and implicit role-permission assignments (*IPA*). Secure Schema Object Hierarchy (SSOH) is a partial order between schema objects defined by a security administrator, in which the permissions defined on low level objects will be transported to high level objects. Some constraints are specified for the *SP* and *EPA*.

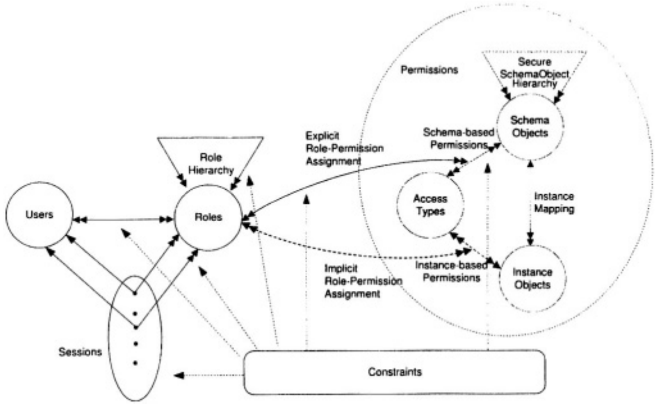

Figure 1. Extended RBAC Model

Extended RBAC Model:

- *U* (Users), *R* (Roles), *P* (Permissions), *S* (Sessions), *AT* (Access Types), *IO* (Instance Objects), *SO* (Schema Objects), *SP* (Schema-based Permissions), *IP* (Instance-based Permisson), *IM* (Instance Mapping), *SSOH* (Secure Schema Object Hierarchy)

- *S, UA* and *RH* are same as RBAC96
- $SP \subseteq AT \times SO$: Schema-based Permissions
- $IP \subseteq AT \times IO$: Instance-based Permissions
- $IM : SO \rightarrow 2^{IO}$: Instance Mapping
- $P = IP \cup SP$: Permissions
- $SSOH \subseteq SO \times SO$: Secure Schema Object Hierarchy
- $EPA \subseteq R \times SP$: Explicit Role-Permission Assignment
- $IPA \subseteq R \times IP$: Implicit Role-Permission Assignment
  $IPA = \{(r : R, ip : IP) \mid \exists(at : AT, so : SO, io : IO) \cdot [(r, (at, so)) \in EPA] \wedge [ip = (r, (at, io))] \wedge [io \in IM(so)]\}$
- $PA = EPA \cup IPA$: Permission-role Assignment
- *schema_permissions* : $R \rightarrow 2^{SP}$, a function mapping a role to a set of explicitly assigned schema-based permissions.
  $schema\_permissions(r) = \{sp : SP \mid (r, sp) \in EPA\}$
- *schema_permissions** : $R \rightarrow 2^{SP}$, a function mapping a role to a set of explicitly assigned schema-based permissions with *SSOH.*
  $schema\_permissions^*(r) = \{sp : SP \mid \exists(so : SO, so' : SO, at : AT) \cdot [(so' \leq so) \wedge (sp = (at, so)) \wedge (r, (at, so')) \in EPA]\}$
- *schema_permissions*** : $R \rightarrow 2^{SP}$, a function mapping a role to a set of explicitly assigned schema-based permissions with *SSOH* and *RH.*
  $schema\_permissions^{**}(r) = \{sp : SP \mid (\exists r' : Roles) \cdot [(r' \leq r) \wedge [sp \in schema\_permission^*(r')]\}$
- *instance_permissions* : $R \rightarrow 2^{IP}$, a function mapping a role to a set of implicitly assigned instance-based permissions.
  $instance\_permissions(r) = \{ip : IP | (r, ip) \in IPA\}$
- *instance_permissions** : $R \rightarrow 2^{IP}$, a function mapping a role to a set of implicitly assigned instance-based permissions with *SSOH.*
  $instance\_permissions^*(r) = \{ip : IP \mid \exists(at : AT, so : SO, io : IO) \cdot [(r, (at, so)) \in$
  $schema\_permission^*(r)] \wedge [ip = (r, (at, io))] \wedge [io \in IM(so)]\}$
- *instance_permissions*** : $R \rightarrow 2^{IP}$, a function mapping a role to a set of implicitly assigned instance-based permissions with *SSOH* and

RH.
$instance\_permissions^{**}(r) = \{ip : IP \mid \exists(at : AT, so : SO, io : IO) \cdot [(r, (at, so)) \in schema\_permission^{**}(r)] \wedge [ip = (r, (at, io))] \wedge [io \in IM(so)]\}$

In the following subsections we will explain details of the main components in this extended RBAC model.

3.1. Objects

Definition 1 *(Schema Objects (SO)) A schema object is a XML Schema or schema component(s), may be patterned by an XPath or XQuery expression.*

Definition 2 *(Instance Objects(IO)) A instance object is a XML instance or instance component(s).*

Definition 3 *(Instance Mapping) Instance Mapping (IM) is a mapping between SO and IO:*
$IM : SO \rightarrow 2^{IO}$, *and* $\nexists so_1, so_2 \in SO, io \in IO \cdot (so_1 \neq so_2) \wedge (io \in IM(so_1) \wedge io \in IM(so_2))$

Since it is a well-formed XML document, a XML Schema can be treated as a tree structure, with nodes as the schema elements, attributes and datatypes. XPath can be applied on schema to capture the tree paths with particular conditions. A path can be absolute starting from the root, or relative from current position. For example, "/" is to return the root element of a XML document, while *"customerInfo/name"* selects the child element of *"name"* in *"customer Info"*. Another example, *"customer Info*[*@gender* = "*Male*"]" selects all *customer Info* nodes where the value of *gender* attributes is equal to "Male". XPath can select nodes which satisfy some conditions, such as, *"customer Info/billing Address*[*state* = "*VA*"]" selects all the *customer Info* nodes with Virginia state of billing address. There some built-in functions and in XPath to strengthen the capability of expressions, which can satisfy most of fine-grained specifications.

IM is a one-to-many mapping relationship from *SO* to *IO*. In our model, we use *IM* to implicitly specify the authorization in instance level. Specifically, the permissions defined on schema object will be transported to all its instance objects.

3.2. Secure Schema Object Hierarchy

Definition 4 *(Secure Schema Object Hierarchy (SSOH)) SSOH is the partial order relationship between schema objects:* $SSOH \subseteq SO \times SO$

Mostly SSOH is based on reuse relationships between schema objects. The reuse relationships can be regarded as partial order with acyclic, transitive and reflexive relations. Several types of reuse mechanisms have been specified in W3C XML Schema:

1 Datatype library. A datatype library consists of basic schema datatypes as building blocks. Schema elements and attributes can be created with these basic datatypes by specifying their "type" name.

2 Datatype derivation. New datatype or element can be derived from existing datatypes defined in the same schema or other schemas. There are two types of derivations: restriction and extension.

3 Schema element reference. Without duplicated its definition, a new element can be created by referring to another element in the same schema or from other schemas.

Some modularity mechanisms have been specified by W3C XML Schema to reuse datatypes and elements defined in a different schema, specifically including "include", "redefine" and "import". The first two only can be applied under the same namespace, while the third one can be used between different schemas.

Note that it is the security administrator's duty to decide which schema objects will be put in the hierarchy. At the same time, other relationships can be used in *SSOH* definition by a security administrator.

3.3. Access Types

In this model, four basic access types have been considered to an XML object: *Read, Create, Update* and *Delete.*

1 *Read:* "Read" XML information is the most frequent access to external users. A user activates some roles and try to read some instance information. The authorization enforcement point checks the *Read* permissions defined on the schema objects with these role.

2 *Create:* "Create" access type is particularly for the document composer or source of message. The composition operation has to be verified by the composer's permission on the targeted XML information before storing or transporting. The final version of the created instance may be validated by a schema.

3 *Update:* "Update" permission is to modify the content of a XML object. Also the final version of the modified instance may be validated by a schema.

4 *Delete:* "Delete" is to remove a XML instance or elements in an instance, including the elements and contents. Besides the security check, XML validation check will be launched as well.

3.4. Permissions

Definition 5 *(Schema-based Permissions (SP)) A schema-based permission (explicit permission) is association of a schema object and its allowed access type:* $SP \subseteq AT \times SO$

Definition 6 *(Instance-based Permissions (IP)) An instance-based permission (implicit permission) is association of an instance object and its allowed access type:* $IP \subseteq AT \times IO$

Definition 7 *(Permission Reuse) In SSOH,* $\forall so_1, so_2 \in SO, at \in AT, (so_1 \leq so_2) \wedge ((at, so_1) \in SP) \Rightarrow (at, so_2) \in SP$

Permission reuse greatly reduces the number and complexity of permissions needed to define. This will simplify the authorization management of the access control system. At the same time, modular and extensible security policy can be achieved by using existing permission based on building block objects. The permissions for new schema objects is extensible with capacity of adding new permissions besides the inherited permissions.

3.5. Constraints

Some constraints are introduced in our model to make it fine-grained and flexible.

1. Recursive

"Recursive" constraint is used when permission based on a schema component will be automatically transported to all its sub-components, such as, a *Read* permission on an element will transported to all its sub-elements and attributes. Since schema objects are nested meta-data structure, recursive transportation can happen in several levels. We use $r(n)$ to express n level recursiveness. $r(0)$ means permission not transported, while $r(\infty)$ means always nested recursiveness. For example, (*Read, /*, $r(\infty)$) expresses the permission of *Read* access to a schema, and to all components in this schema.

2. Recursive Direction

In some cases, permissions on schema component can be transported to its super-component. Sometimes this bottom-up recursive direction is useful. For example, if role *CSR* (Customer Service Representative) has *"Read"* permission on *CreditCard* element, the role will have *"Read"* permissions on all other elements or schemas who include this element, since credit card information is normally more sensitive than it's super elements. We use + and – to present top-down and bottom-up directions respectively, such as $r + (n)$, $r - (n)$.

Other constraints can be defined according to the real service and business logic by system designer and security administrator.

3.6. Explicit Permission on XML Instance

So far in our model we assumed that an XML instance is defined by a schema. The explicit permissions defined on schema objects implicitly specify the permission to instance objects. In real world there are two cases that instance-level authorization is desired: one is that there are some *"arbitrary"* XML documents without schema validation; another is that for a schema, the authorization for an instance is different from other instances. For these cases, the permission should be defined based on the instance level. Since XML Schema itself is a well-formed XML document, the permission specification can be easily applied to instance objects without any change. Both schema and instance objects can be expressed in XPath or XQuery. The only thing with extra consideration is to specify which level object is used in explicit permission. This is one of the benefits of the proposed schema based access control model. Generally the permission on instance level will overcome that in schema level.

4. High-level Implementation Mechanism

XML provides a uniform mechanism to solve problems in heterogeneous environment. Figure 2 shows the server-pull [2] implementation architecture. All the messages transported among the services are identified in XML. XML requests and responses are XML messages, whose format will be defined in schemas. We will use some existing XML standard to transport the security information, such as SAML (Security Assertion Markup Language) [13]. With SAML, the messages for user authentication, user-role assignment request/response, permission-role assignment request/response will be in standard format. And the underlying mechanism of user authentication, role server, and policy server can be abstract. That means, the implementation mechanism can be embedded to other systems very easily. This is an important advantage with" pure" XML implementation. Figure 3 shows an algorithm for the process of access control decisions. The algorithm is described in a way for clear exposition rather than efficiency. Implementation details of the XML mechanism will be our future work. As shown in the algorithm, an access request includes a user information (u), a role activated by the user (r), access type *(at),* and the target XML document (*target.xml*) to be accessed. To make an access control decision, some other related information is needed, such as the schema(s) of *target.xml* (*target.xsd*), the possible expected output schema $(target'.xsd)$, as well as the security information: user-role assignment (*UR.xml*), permission-role assignment (*PR.xml*), role hierarchy (*RH.xml*), and Secure Schema Object Hierarchy (*SSOH.xml*). In some cases, the output of an access control decision is required to satisfy some expected schema. The function $im(target'.xsd)$ is to check if output $target'.xml$ can be validated by $target'.xsd$. Since the authorization process

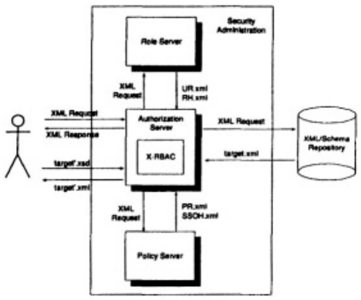

Figure 2. High-level Implementation Architecture

can remove some nodes of the input object, the output may not satisfy some particular schema, which is required by most applications. In this case, the access will be denied.

Function $roles(u)$ returns a set of roles assigned to a user, including direct assignment and inheritance with role hierarchy. Function *parse(target.xml)* returns a set of data with tree structure. For each instance component i, function $sm(i)$ returns its corresponding schema object. The key part of the algorithm is a recursive method $recursive\_access(t,$ *rootnode*), which is a depth-first algorithm. For each subtree t, the method first checks the permission to the root node. If the access to it has been permitted, all subtrees will be checked by the same mechanism. Otherwise, access to the whole element and its sub-elements will be denied.

Function *s_permissions*(*at*, r, *PR.xml, RH.xml, SSOH.xml*) returns a set of schema nodes which is accessible to r or a subrole of r in role hierarchy. *PR.xml* is defined for each schema. There are different implementations of this permission-role assignment. Here, we just provide a simple and abstract structure as shown in Figure 4. The basic format is similar to a schema document. In this example, each schema component is followed by one or more < *permission* > tags to specify the permissions. A < *permission* > tag has two attributes: *role* and *access*. The value of < *role* > is a role name, and the value of *access* is a pre-defined access type. Since RBAC is a close model, any other role which is not specified in the < *permission* > of a schema component cannot access this object. Because of the large number of schema objects and roles, in the real implementation, a visual tool will be developed to do the permission-role assignment.

UR.xml and *RH.xml* are more static in real implementation mechanism. *UR.xml* is very straightforward. The main component is a user information

```
XML-based Algorithm:
Input: Access request: $(u, r, at, target.xml)$
       Schema of target: $target.xsd$
       Schema of expected output: $target'.xsd$
       Security information: $PR.xml, UR.xml, RH.xml, SSOH.xml$
Output: $target'.xml$
Method:
       // Verify user-role assignment:
1)     if $r \notin roles(u, UR.xml, RH.xml)$
2)         ACCESS denied;
3)     endif
       // Parse and generate instance tree $t$ rooted in $rootnode$ of $target.xml$:
4)     $t = parse(target.xml)$
5)     $recursive\_access(t, rootnode)$
6)         $so = sm(rootnode)$
7)         if $so \in s\_permissions(at, r, PR.xml, RH.xml, SSOH.xml)$
8)             $ACCESS$ $rootnode$ is permited;
9)             $add(target'.xml, rootnode)$;
10)            if ($rootnode$ is not leaf)
11)                foreach subtree $st \in t$ rooted in $subnode$
12)                    $Recursive\_access(st, subnode)$;
13)                endfor
14)            endif
15)        else
16)            $ACCESS$ $t$ is denied;
17)    end $recursive\_access$;
       // Validation check with $target'.xml$ according to $target'.xsd$ if needed;
18)    if $target'.xml \in im(target'.xsd)$
19)        Output $target'.xml$;
20)    else Access $target.xml$ denied;
21)    endif
```

Figure 3. Algorithm of XML Access Control

with some sub-nodes of name, department, role name, etc. In the real world, user-role assignment is built on organization structure, maybe be cooperated by some department other than security administrator [3]. So the *UR.xml* may be derived from other systems. This is out of the range of this paper. What we present here are all very high-level and conceptual. In the next step of this work, we will enrich and finalize the details of this algorithm. Schemas for *UR.xml, PR.xml, RH.xml,* and *SSOH.xml* will be defined. Since input and output in the algorithm are all XML documents, open source XML APIs will be applied, such as SAX (Java implemented XML parser).

```
...
<xs:element name="customerInfo">
   <permission role="CSR" access="read"/>
   <permission role="CSR" access="write"/>
   <xs:sequence>
      <xs:element name="ssn" type="xs:string">
         <permission role="CSR" access="read"/>
      </xs:element>
      <xs:element name="name">
         <permission role="CSR" access="read"/>
         <permission role="CSR" access="write"/>
         ...
      </xs:element>
      <xs:element name="creditCardInfo">
         ...
      </xs:element>
   </xs:sequence>
   <xs:attribute name="gender">
      <permission role="CSR" access="read"/>
      <permission role="CSR" access="write"/>
      ...
   </xs:attribute>
</xs:element>
...
```

Figure 4. PR.xml Example

5. Related Work

E.Damiani et al. [6] and E.Bertino et al. [7, 8] did some work in XML document access control, which is close to our work. E.Damiani et al. used organization-level DTD and site-level DTD as objects, built access control model for XML document based on authorization rules of *(subject, object, permission, recursive)*. They only considered *read* operation in the paper. The algorithm to compute the final view of XML document based on the subject's authorization rules is presented with DOM tree labelling and transformation. E.Bertino et al. provided XML document access control policies, model and implementation based on authorization built on DTD and XML document. The authorization propagations from element-to-subelement, element-to-attribute, DTD-to-instance were considered with recursive options. Based on this, a Java-based access control system, named **Author-χ,** is implemented with discretionary access control (DAC) model. Both Damiani and Bertino's models are based on DTD, using the relationship between XML instance document and DTD to transport authorizations. Another point is that they used DAC model. The main difference between our work and these previous work is that we use RBAC model based on XML Schema. The user-role and permission-role greatly will improve the security administration with vast users and XML objects. Also, the schema based permission and permission reuse enable modular and reusable policy deployment. Another point is that our model is not only for static XML documents, but for dynamically generated XML messages, such as SOAP and other XML protocols.

From industry, mainly there are two projects on XML security motivated by Organization for the Advancement of Structured Information Standards (OASIS): Security Assertion Markup Language (SAML) [13] and extensible Access Control Markup Language (XACML) [14]. The motivation of SAML is to define "assertion" for security statements in XML, such as authentication and authorization. Some XML protocols have been specified in SAML to exchange assertion in distributed environment. Our framework is orthogonal to SAML. Actually we will use SAML mechanism in our architecture. As shown in Figure 2, all XML requests and responses will be implemented in SAML messages and protocols. XACML is similar to our work, which applies XML format to specify access control policy with objects identified in XML. The main difference is that XACML focus on policy level, including logical predicates, rules, and policy combining. So in XACML, there is no clear access control model supported. In our model, we focus on model level. A XML-based RBAC model is clearly supported, which is policy neutral.

6. Conclusion and Future Work

This paper presented an extended RBAC model for XML security. Permissions in the model are defined based on XML Schema components and will be transported to all instances. Different from the previous access control model built on DTD, complex object hierarchies can be achieved with rich relationships between schemas components. The permission reuse through these hierarchies greatly improves the security administration by modular, reusable and extensible permissions. Several constraints are presented in the model. The proposed model can be modularly deployed and flexibly administrated in distributed environments. The model can be applied to no-schema based XML instance and instance level authorizations easily. The abstract implementation architecture is presented.

The detail implementation of proposed model is the next step of this work. XML technology is applied in the implementation mechanism. The modular and reusable features of XML will be the benefits to the RBAC model. SAML will be applied in our implementation for all XML messages. In the future we will study the access control in Web Services technologies, where the applications are built from XML protocols. As a service-oriented software environment, Web Services have roles like service provider, registry, requester, deployer, as well as system administrator. The schema based RBAC model can be a solution of Web Services security.

References

[1] R.Sandhu, E.J.Coyne, H.L.Feinstein, Role based access control models, IEEE Computer, 29, (2), pp.38-47, 1996.

[2] Joon S.Park, R.Sandhu, and Gail-Joon Ahn, Role-Based Access Control on the Web, ACM Trans. Information and System Sec., Vol.4, No.1, Feb. 2002.

[3] Sejong Oh, Ravi Sandhu, A Model for Role Administration Using Organization Structure, In Proc. of ACM Symposium on Access Control Models and Technologies, 2002.

[4] E.Bertino, S.Jajodia, and P. Samarati, Supporting Multiple Access Control Policies in Database Systems, IEEE Symposium on Security and Privacy, 1996.

[5] S.Jajodia, P.Samarati, and V.S.Subrahmanian, and E.Bertino, A United Framework for Enforcing Multiple Access Control Policies, ACM SIGMOD International Conference on Management of Data, 1997.

[6] 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 Sec., Vol.5, No.2, May 2002.

[7] E. Beritino, S.Castano, E.Ferrai, and M.Mesiti, Specifying and Enforcing Access Control Policies for XML Document Sources, World Wide Web Journal, Vol.3, No.3, 2000.

[8] E. Beritino, S.Castano, E.Ferrai, and M.Mesiti, Author-x: a Java-Based System for XML Data Protection, 14th IFIP WG 11.3 Working Conference on Database Security, 2000.

[9] World Wide Web Consortium (W3C), Extensible Markup Language (XML), http://www.w3.org/XML, October, 2000

[10] World Wide Web Consortium (W3C), XML Path Language (XPath), http://www.w3.org/TR/xpath20, August, 2002

[11] World Wide Web Consortium (W3C), XML Schema Part 0: Primer, http://www.w3.org/TR/xmlschema-0, May, 2001

[12] World Wide Web Consortium (W3C), XML Schema Part 1: Structures, http://www.w3.org/TR/xmlschema-1, May, 2001

[13] OASIS, Security Services TC, http://www.oasis- open.org/committees/tc_home.php?wg-abbrev=security

[14] OASIS, eXtensible Access Control Markup Language TC, http://www.oasis-open.org/committees/tc_home.php?wg.abbrev=xacml