Role Based Access Right Specification for Secure Information Sharing*

Ganesh Godavari

Department of Computer Science, University of Colorado, 1420 Austin Bluffs Parkway

Colorado Springs, Colorado 80917 USA

gkgodava@cs.uccs.edu

Abstract

In this report, we explore the issues involved with the access right specification for large scale information sharing system. We propose a model for specification of the access rights in a large scale information sharing system. The proposed model helps in easy role right specification for the four components of the RBAC model.

Keywords: Access Right Specification, Secure Information Sharing, Attribute Certificate.

1 Introduction

As the WWW is quickly becoming a place for sharing of information, piracy and misuse of information are becoming a real threat. Security and Authorization have become necessary. This situation not only provides excellent business opportunities but also posts research challenges. One of the most challenging problems is specification and manageability of authorization policies across the organization. Role-Based Access Control (RBAC) simplifies the access control administration and provides better manageability in enterprise environments by allowing permissions to be managed in terms of user job roles [7]. RBAC maps user job roles to application permissions so that the access control administration can be accomplished in terms of the job role of users. This means that administrators will have to set up and assign clients with roles, such as employee, manager and administrator, without having to change the access permission on each client.

Role based authorization policy specification is a critical task in managing a large scale information sharing system. This authorization policy specification must allow easy and flexible access control specification, as access control decisions are driven by an authorization policy. This authorization policy must be stored in a secure and untamperable format.

Public Key Certificate (PKC) strongly binds a public key to its subject (country, location, organization unit etc.) helping to identify the holder of the certificate, and cannot be used for holding the authorization policy. IETF has

*This research work was supported in part by a NISSC AFOSR Grant award under agreement number F49620-03-1-0207.

proposed Attribute certificates (AC) as a solution for the authorization services [1, 10, 5]. AC is designed to convey a potentially short-lived attribute about a given subject to facilitate flexible and scalable privilege management. The attribute certificate points to a public-key certificate that can be used to authenticate the identity of the attribute certificate holder.

The organization of this paper is as follows. Section 2 provides an overview of the related research. Section 3 describes the design and implementation of our authorization policy specification format. In Section 4 we present a case study for usage of authorization policy specification format. Conclusion of our privilege specification is presented in Section 5.

2 Related Research Technologies

2.1 Role Based Access Control

Role-based access control [4, 9, 8, 11, 2] has gained attention as a proven alternative to traditional discretionary and mandatory access control mechanisms. RBAC helps specify organization's security policies reflecting its organizational structure. RBAC reference model and functional specification are organized into four components: Core RBAC, Hierarchical RBAC, Static Separation of Duty Relations (SSD), Dynamic Separation of Duty Relations (DSD).

In the core RBAC, a user can be assigned one or more roles, and a role can be assigned to one or more users. Roles are based on the user's job responsibilities in the organization. This provides for flexibility and finer granularity during the assignment of access permissions to roles and users to roles. In the hierarchal RBAC model, the role hierarchy partially determines which roles and permissions are available to users via inheritance. For example, a senior role can inherit permissions from junior roles. A user establishes a session during which he activates some subset of roles of which he is a member. SSD relations help in preventing conflict of interests, that arise when user gains permission associated with conflicting roles. DSD relations help to place constraints on roles that can be activated in the session of a user.

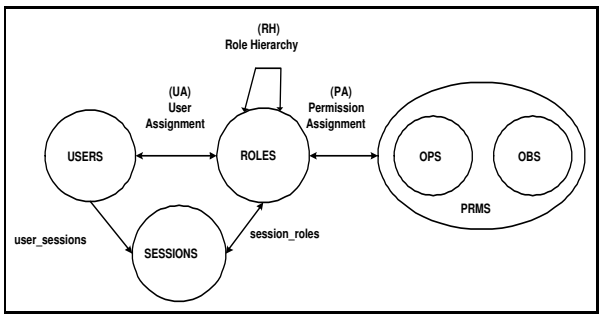

Figure 1: model[7]

The Core RBAC in Figure 1 consists of 1) a set of users (USERS) where a user is an intelligent autonomous agent,

2) a set of roles (ROLES) where a role is a job function, 3) a set of objects (OBS) where an object is an entity that contains or receives information, 4) a set of operations (OPS) where an operation is an executable image of a program, and 5) a set of permissions (PRMS) where a permission is an approval to perform an operation on objects. The cardinalities of the relationships are indicated by the absence (denoting one) or presence of arrows (denoting many) on the corresponding associations. For example, the association of user to session is one-to-many. All other associations shown in the figure are many-to-many. The association labeled Role Hierarchy defines the inheritance relationship among roles.

Further information about RBAC is available at [7].

2.2 eXtensible Access Control Markup Language

eXtensible Access Control Markup Language (XACML) is an XML based language that describes access control policy language and a request/response language. XACML is proposed by OASIS consortium and is currently in preliminary stages. XACML defines two entities: Policy Enforcement Point (PEP) and Policy Decision Point (PDP). Request for a resource is submitted to the PEP, which creates an XACML request based on attributes like subject, resource etc., and sends it to PDP. Policy Decision Point analyzes the request and retrieves policies that are applicable to this request, and determines whether access should be granted based on the XACML rules for evaluating policies. An XACML response is sent by the PDP to the PEP, which can then permit or deny access to the requester.

Figure 2 shows permission policy specification for the role of an employee. The role employee can create a purchase order. Although XACML provides flexibility and expressiveness in specifying policies, it is very complex and cumbersome in specifying the policies [6].

Further information about XACML is available at [3].

3 SIS-Authorization Policy Specification

Organizational roles help in simplifying the management of privileges. These privileges can be expressed in a policy document that clearly specifies what privileges are needed by each role in the organization in order to carry out the duties required by that role. This policy document is called Authorization Policy, and is specified in terms of rights or privileges. Clearly the mapping of roles to rights is one to many. Rights can be specified using simple *if* <*condition*>*do* <*action*> format. This kind of privilege specification is not only simple but is also close to the human thought process.

Simplicity in the privilege specification was one of the major goals of our secure information sharing project. We identified four features that are required for role-based access privilege specification:

1. Resource Name: name of the resource being requested. Each resource can have different privilege requirement

```
<PolicySet PolicySetId=‘‘Permission PolicySet for employee role’’>
  <Policy PolicyId=‘‘Permissions specifically for the employee role’’
       RuleCombiningAlgorithm=‘‘permit-overrides’’>
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <AnyAction/>
      </Actions>
    </Target>
    <Rule RuleId=‘‘Permission to create a purchase order’’
        Effect=‘‘Permit’’>
      <Target>
        <Subjects>
          <AnySubject/>
        </Subjects>
        <Resources>
          <Resource>
            <ResourceMatch MatchId=‘‘string-match’’>
              <AttributeValue>purchase order</AttributeValue>
              <ResourceAttributeDesignator
                    AttributeId=‘‘resource-id’’/>
            </ResourceMatch>
          </Resource>
        </Resources>
        <Actions>
          <Action>
            <ActionMatch MatchId=‘‘string-match’’>
              <AttributeValue>create</AttributeValue>
              <ActionAttributeDesignator AttributeId=‘‘action-id’’/>
            </ActionMatch>
          </Action>
        </Actions>
      </Target>
    </Rule>
  </Policy>
  <PolicySetIdReference>Permission PolicySet for employee role</PolicySetIdReference>
</PolicySet>
```

Figure 2: Example of a Permission <PolicySet >for employee role []

and access to these resources can be restricted based on the user role privileges.

2. Parameter(s) to a Resource: resource behavior changes according to the type and number of parameters passed to the resource. For example, results of a query to a database table depends not only on the number of parameters being requested but also on the type of parameters being requested.

3. External privilege validation: in a cooperative environment where information is shared between various organizations, privilege verification often involves checking for privileges with other organizations in order to process the resource request. This also helps to interact with external policy decision points like xacml.

4. Role Hierarchy specification: this should allow for clear and precise specification of privilege(s) to avoid any conflict of interest privilege specification in and between the role(s).

Figure 3 shows the context free grammar of the privilege specification format in our SIS model. Privileges are expressed in a bunch of simple 'if' statements, where each statement allows any depth of parentheses. This gives the users maximum convenience in specifying privilege rules. In each rule the precedence of operators can be changed by using parentheses. The operator '#' is used for regular expression matching.

```
sisprivilegeset <role name> <privilegeset name>
{
<privilege>   := if ( <expression> ) do <action>
<expression>:= <term> | <term> && <expression> | ( <expression> ) | ! ( <expression> )
<term>    := <factor> | <factor> || <term> | ( <term> )
<factor>  := <variable operator value>
<operator>     := > | >= | < | <= | == | != | #
<action>  := grantAccess | rejectAccess | acquirePrivileges <privilegeset Name> |
contact <authroization server>
}
```

Figure 3: sis privilege specification format

acquirePrivilages is used to tell the authorization Policy enforcer to check the feature set against the privilege set specified by it privilegeset name. This kind of privilege specification is especially useful in situations where all or a subset of privileges are delegated by one user to another. It is also useful in specifying the mapping of role hierarchy to privilege's in an organization.

External Privilege validation is achieved through *contact* action of the privilege specification. contact action in the privilege specification makes the authorization policy enforcer to query external authorization policy enforcer specified by the *authorization server*. *grantAccess* and *rejectAccess* are used to grant and reject privileges.

The privilege rules can be categorized into a list identified by the 'privelageset name'. Categorizing privileges into a set allows for clear specification of the privileges based on the type of resource being accessed. In addition to delegated privilege specification, it also helps in evaluation of only a subset of privileges rather than the all the privileges associated with a role. If none of the privilege rules in the *privilegeset* match then resource request is rejected.

4 Case Study

In order to illustrate the use of our privilege policy specification, the following scenario was considered: A web server is setup by each department in the university. Class Material and information about students like student-id, SSN, emergency contact number is available on the web and is maintained by the department in which student is majoring in. The student information is available through a web program called *displayStudentProfile.cgi* and the parameter is *updateStudentContactInformation* for updating their emergency contact information.

The following constraints are applied:

- Student can read all the tutorials on the web irrespective of the department he/she is enrolled in; also his/her

personal information can be retrieved and modified.

- Department Chair of departmentA must be able to retrieve all students information belonging to the departmentA.
- Department Assistant of departmentA needs to get student information to inform the student belonging to departmentB that he/she is allowed to take the course outside the students major.

The first constraint involves specification of the privileges required by the role student. The second constraint shows how additional privilege can be associated to a role in addition to the privileges required to carry out the duties associated with the role. The third constraint specifies how our role based privilege specification can be used in a distributed environment where cooperation of various organizations is mandatory to have a seamless interaction between them.

Figure 4, 5, and 6 shows the privileges associated with the roles of student, department chair and department assistant.

```
sisprivilegeset student filematch {
    if ( ( url # ``*tutorial*'' ) && ( requestAction # ``GET'' ) ) do grantAccess
}

sisprivilegeset student cgimatch {
    if ( ( url # ``displayStudentProfile.cgi'' ) &&
  ( parameter # ``updateStudentContactInfo*'' ) ) do grantAccess
}
```

Figure 4: student privilege policyset specification file

```
sisprivilegeset departmentchair filematch {
    if ( ( url # ``*tutorial*'' ) ) do grantAccess
    if ( ( url # ``*'') ) do  acquirePrivileges departmentchairprivileges
}



sisprivilegeset departmentchair cgimatch {
    if ( ( url # ``displayStudentProfile.cgi'' )  && ! ( parameter # ``updateStudentContactInfo*'' )
 &&  ( organizationalunit # ``departmentA'' ) ) do grantAccess
    if ( ( url # ``*'') ) do  acquirePrivileges departmentassistantprivileges
}

sisprivilegeset departmentchair departmentchairprivileges {
# web privileges required to carry out department chair duties
:
.

}
```

Figure 5: Department chair privilege policyset specification file

```
sisprivilegeset departmentassistant filematch {
    if ( ( url # ‘‘order’’ ) ) ) do grantAccess
    if ( ( url # ‘‘inventory*’’ ) ) ) do grantAccess
    if ( ( url # ‘‘*’’) ) do  acquirePrivileges departmentassistantprivileges
}

sisprivilegeset departmentassistant cgimatch {
# contact other departments for information in case the student does not belong to departmentA
    if ( ! ( organizationalunit # ‘‘departmentA’’ ) ) do contact departmentB departmentC
# display information about all students belonging to departmentA
    if ( ( url # ‘‘displayStudentProfile.cgi’’ )  && ! ( parameter # ‘‘updateStudentContactInfo*’’ )
 &&  ( organizationalunit # ‘‘departmentA’’ ) ) do grantAccess
# acquire privileges that are needed to carry out department assistant duties
#that are general to all departments
    if ( ( url # ‘‘*’’) ) do  acquirePrivileges departmentassistantprivileges
}

sisprivilegeset departmentassistant departmentassistantprivileges {
# web privileges required to carry out department assistant duties
:
.

}
```

Figure 6: Department Assistant's privilege policyset specification file

5 Conclusion

we have proposed a new privilege policy specification which is very powerful and flexible. This kind of privilege policy specification based on the user organization role can be used to create a secure information sharing environment. We have provided a simple case study showing how our privileges specification format can be used to grant additional privileges to a role in addition to the privileges required to carry out the duties of the role.

References

[1] Chadwick, D. W., and Otenko, A. The permis x.509 role based privilege management infrastructure. *Future Gener. Comput. Syst. 19*, 2 (2003), 277–289.

[2] D. Ferraiolo, J. C., and Kuhn, D. R. Role-based access control: Features and motivations. *Computer Security Applications Conference* (1995), 241.248.

[3] eXtensible Access Control Markup Language (XACML). *authorization policies specification in XML.* http://www.oasis-open.org/specs/index.php, 2004.

[4] Ferraiolo, D., and Kuhn., D. R. Role-based access control. *15th NIST-NCSC National Computer Security Conference* (1992), 554–563.

[5] Javier Lopez, Antonio Mana, J. J. O. J. M. T., and Yague, M. I. Integrating pmi services in corba applications. *Comput. Stand. Interfaces 25*, 4 (2003), 391–409.

[6] Lorch, M., Proctor, S., Lepro, R., Kafura, D., and Shah, S. First experiences using xacml for access control in distributed systems. In *XMLSEC '03: Proceedings of the 2003 ACM workshop on XML security* (2003), ACM Press, pp. 25–37.

[7] NIST. Role-based access control, 2004.

[8] R. Sandhu, E. J. Coyne, H. L. F., and Youman., C. E. Role-based access control models. *IEEE Computer 29* (1996), 38–47.

[9] Sterne., D. F. A tcb subset for integrity and role-based access control. *15th NIST-NCSC National Computer Security Conference NIST/NSA* (1992).

[10] Thompson, M. R., Essiari, A., and Mudumbai, S. Certificate-based authorization policy in a pki environment. *ACM Trans. Inf. Syst. Secur. 6*, 4 (2003), 566–588.

[11] von Solms, S. H., and van der Merwe, I. The management of computer security profiles using a role-oriented approach. *Comput. Secur. 13*, 9 (1994), 673–680.