CHAPTER 4

EVALUATION OF EXISTING ROLE BASED ACCESS CONTROL MECHANISM

4.1 PROBLEM STATEMENT

In this chapter we investigate some of the implementations of Role Based Access control and implement RBAC to compare efficiency of the access control mechanism.

4.2 ROLE BASED ACCESS CONTROL IN GRID SECURITY

The members of the virtual organization are the proprietors of these resources and the members desire to sustain a local control over their resources. This has necessitated the need for access control mechanisms in virtual organizations. RBAC provides mechanism for global naming of users which is accepted and supported at all sites the user accesses. It is necessary for the user’s identity to be asserted in multiple domains while the jobs progress (Marty Humphrey et al 2005). It must be possible for users to access resources in a dynamic fashion devoid of any administrator intervention. These resources need to be synchronized appropriately and must interrelate securely with other services. Therefore, it is essential for the resources to possess global identities and they need to be accessed without any local policy violation (Anil Pereira et al 2006).

In Access Control List (ACL) based authorization, for each resource, a list of authorized users with their privileges on that resource is
specified on the ACL. The user is authenticated by the VO to access the resource based on the associated privileges. The ACL approach has many disadvantages. It is not scalable to large users and organizations in the VO must agree on access policy in advance that is enforced by each local site administrator. ACL is vulnerable to errors due to the dynamic aspect of the grid. The management of ACL based on individual user becomes cumbersome and error prone. To overcome these problems faced in ACL, RBAC authorization model has been used.

4.3 EXTENSIBLE ACCESS CONTROL MARKUP LANGUAGE

eXtensible Access Control Markup Language (XACML) is an XML-based language (Eastlake et al., 2002) for access control that has been standardized within OASIS (Moses 2005). XACML is a standard that defines the syntax and semantics of a language for expressing and evaluating access control policies. XACML standard describes both a policy language and an access control decision request/response language written in XML. It provides an access control model and is a common language to express and to enforce security policies in a variety of environments. It has several advantages such as scalability, flexibility, extensibility, reusability and a policy referencing mechanism.

XACML supports a wide range of data types and functions that can be used in various environments. It defines rules on how authorization decisions from evaluating applicable policies are combined. Therefore, it is used as a replacement for various application-specific and proprietary access control policy specification languages. XACML primarily allows developers to describe access control policies for XML objects and other objects like Web services or files. XACML is used to express three different types of access control information:
Privilege attributes
Privilege management policies
Access control policies

It deals with the basic element called attributes. Attributes specify the characteristics of the subject, resource, action or environment in which the access request is made. XACML is an access control-related Web services standards and also supports Attribute Based Access Control (ABAC) model.

4.3.1 XACML Policies

An XACML policy is composed of the following top level elements.

- PolicySet
- Policy
- Combining Algorithms
- Target
- Rules
- Obligations.

**Policy/PolicySet:** Each XACML policy contains exactly one Policy or PolicySet root XML element. A PolicySet is a container that can hold other Policies or PolicySets and references to policies found in remote locations. A Policy represents a single access control policy, expressed through a set of Rules with Permit or Deny effect. A Policy or PolicySet may contain multiple policies or rules, each of which may evaluate to different access control decisions.

**Combining Algorithms:** XACML has a collection of Combining Algorithms to reconcile the decisions made by these rules. Each algorithm
represents a different way of combining multiple decisions into a single result. There are Policy Combining Algorithms used by PolicySet and Rule Combining Algorithms used by Policy. The combiner can be regarded as a logical conjunction of permits. There are seven combining algorithms to build up increasingly complex policies. It wraps all policies together with a Policy Combining Algorithm that resolves conflicts among policy decisions. Policy Combining Algorithms include

- Permit-overrides
- Deny- Overrides
- First-applicable
- Ordered-permit-overrides
- Ordered-deny-overrides
- Only-one-applicable.

**Target:** A Target is basically a set of simplified conditions for Subject, Resource and Action. These conditions use boolean functions to compare values found in a request with those included in the Target. If all the conditions of a Target are satisfied, then its associated PolicySet, Policy, or Rule applies to the request.

**Rules:** Rules are cores of XACML Policies. A Policy can contain different number of Rules combined with a Combining Algorithm. A rule specifies a logical expression stating the conditions a request must satisfy for the rule to be applicable and a decision in case the conditions are satisfied. Rules have an effect of Permit or Deny that is associated with successful evaluation of the Rule. Figure 4.1 shows the basic blocks of XACML access control system.
The main components of the XACML architecture are:

- **The Policy Decision Point (PDP):** The PDP receives an XACML request, fetches the applicable policy(s) from the policy administration point, retrieves the attribute values from the policy information point, evaluates the request against the applicable access control policies and returns an authorization decision to the PEP.

- **The Policy enforcement point (PEP):** The PEP receives an access request, extracts the attributes in the request, generates an XACML request and sends it to the PDP for evaluation. It also makes sure that all the obligations with an authorization decision are executed. An obligation is an action that should be performed together with the enforcement of an authorization decision and is specified in an access control policy.
• **The Policy administration point (PAP):** The PAP creates an XACML access control policy(s) and stores it in a policy database server and sets a restriction in order to prevent unauthorized access to the access control policies. It conducts a regular check in order to maintain the uniqueness of policy identifiers.

• **The Policy information point (PIP):** The PIP is a component that acts as a directory server that stores the attribute values and makes it available to the PDP. Attributes values are the data that describe the characteristics of a subject, resource, action and environment.

### 4.4 IMPLEMENTATION OF RBAC IN CAMPUS GRID BASED VO

#### 4.4.1 Introduction

In the previous chapter the performance advantage of campus based virtual organization was investigated using Discretionary Access Control (DAC). In this section it is proposed to investigate the performance of the network using RBAC for access control. VOs implemented in college campus environment need to address challenges as which users are part of which research group within the VO. Examples of relationships include

• A professor may be a member of several research groups. In this scenario the resource allocation will be different for each research group / different project.

• A professor may have multiple roles in a VO. He may act as a research group administrator in certain scenarios and as regular VO member in other scenarios.
- A group of professors may alternate the administration of the VO with one professor acting as the administrator at a given time.

From the above scenarios it is seen that security mechanisms including DAC exhibit several limitations due to the complex structure of the VO. Enforcement mechanism for utilization of CPU cycles at the system level may not be aware of the VO groups or roles. RBAC overcomes many of the challenges faced in the VOs security mechanism.

4.4.2 Example Roles and Permissions in a Campus VO

Consider a campus grid with five users and four roles, with the roles being professor, faculty, administrator and student. Users may have multiple roles as indicated below.

- user1 --> Professor
- user2 --> Faculty
- user3 --> Faculty, Administrator
- user4 --> Administrator
- user5 --> Student

Roles will have the following relationships.

- Professor has more authority than a faculty.
- Professor has more authority than an administrator.

The permitted actions for each of the role can be described using rules.

1. Professor can utilize the entire resources allocated for the research group.
2. Faculty can utilize a resource which is a subset of the total resources allocated to the research group and allocated by the administrator.

3. Administrator can define local policies within the college for resource utilization.

4. A Professor and Administrator together can create a Student user.

5. Student can use only the resources that are defined by administrator.

The following should be noted:

- The first three rules grant permissions to a specific role;
- The fourth rule requires two roles for an action to be permitted;
- The fifth rule restricts permissions given in the previous rules.

4.4.2.1 Implementation of RBAC profile

The first three rules can be implemented easily.

- Professor permission policy set will contain a reference to both faculty permission policy set and administrator permission policy set.

- The administrator action would be part of the professor permission policy

- Modification action will be part of the administrator permission policy set.
• The fourth rule indicates that you should be both a professor and an administrator to create a temporary user.
  
  o It is possible to write a new policy set that applies to users who have both professor and administrator role
  
  o It is also possible to include an additional reference that needs to be added to the professor permission policy set so that it inherits the permission of the administrator policy set.

• The fifth rule is more complicated as it restricts permissions that have already been given in previous rules
  
  o To add the fifth rule would require the use of a different methodology that determines if there is at least one policy that permits the action and none of the other policies deny the action.

4.5 METHODOLOGY

4.5.1 Proposed XACML Profile for RBAC

A Role is defined as a specific XACML Attribute. An XACML Attribute has an identifier and a value of a particular data type. Any XACML PDP can evaluate policies using arbitrary Attribute identifiers (XACML Profile for Role Based Access Control, Draft 01, 13 February 2004)

XACML defines about 20 standard data types (string, integer, x500Name, etc.), along with standard functions for comparing and manipulating these data types. Any compliant XACML PDP can evaluate policies using any values of any of the XACML standard data types. XACML can be extended to use new data types and their corresponding functions without affecting the structure of the language.
An XACML Attribute may pertain to a Subject, a Resource, an Action, or to the Environment in which the access request is being made. In this Profile, a Role is defined as a Subject Attribute. XACML supports multiple subjects per access request, indicating various entities that may be involved in making the request. For example, there is usually a human user who initiates the request, at least indirectly. There are usually one or more applications or code bases that generate the actual low-level request on behalf of the User. There is some computing device on which the application or code base is executing, and this device may have an identity such an IP address. In this profile, a Role Attribute may be assigned to any of the types of subjects involved in making an access request.

In this Profile, there are two types of policies each expressed as an XACML <PolicySet>.

- **Role PolicySets**
  
  o These are applicable only to holders of a given Role.
  
  o They reference the **Permission PolicySet** that specifies the permissions to be granted to holders of this Role.

- **Permission PolicySets**

  o These are never used as base policies, but only by reference from **Role PolicySets**.

  o These are applicable to any Subject

  o They specify a set of Permissions to be granted to holders of a given Role or to holders of any Role that is senior to the given Role.
Permission PolicySets need to be stored in the policy repository in such a way that they can never be used as the initial policy for an XACML Policy Decision Point (PDP). This is because a Permission PolicySet by itself needs to be applicable to any user (Subject) in order to allow Hierarchical RBAC. It is the Role PolicySet that controls which Permission PolicySets will be applied to holders of a particular role.

The Permission PolicySets that do not reference other Permission PolicySets could be XACML <Policy> elements rather than <PolicySet>. Requiring them to be <PolicySet> elements allows roles that were initially non-hierarchical to later include other permissions without requiring any change to the definitions of roles that reference them.

The two types of policies allow support for Hierarchical RBAC, where a more senior role can acquire the permissions of a more junior role. A simpler Profile supporting only the Core RBAC requirements is almost trivial to create with XACML. Designing the Profile to support both Core RBAC and for Hierarchical RBAC, however, seems preferable to having two separate profiles.

XACML does not handle the assignment of role attributes to users, so this Profile assumes that the presence in the XACML Request Context of a role attribute for a given user (Subject) is a valid assignment.

A sample policy set for permission and role is shown in Figure 4.2 and Figure 4.3
Figure 4.2 Permission <Policy Set> for the VO administrator

```xml
<PolicySet xmlns="urn:oasis:names:tc:xacml:1.0:policy"
PolicySetId="PPS:VO administrator:role"
PolicyCombiningAlgId="&policy-combine;permit-overrides">
  <Target/>
  <Subjects><AnySubject/></Subjects>
  <Resources><AnyResource/></Resources>
  <Actions><AnyAction/></Actions>
  <Target/>
  <!-- Permissions specifically for the VO administrator role -->
  <Policy PolicyId="Permissions:specifically:for:the:VO administrator:role"
RuleCombiningAlgId="&rule-combine;permit-overrides">
    <target/>
    <Subjects><AnySubject/></Subjects>
    <Resources><AnyResource/></Resources>
    <Actions><AnyAction/></Actions>
    <Target/>
    <!-- Permission to load a job in the campus grid -->
Effect="Permit">
      <Target/>
      <Subjects><AnySubject/></Subjects>
      <Resources><Resource>
        <ResourceMatch MatchId="&function;string-match">
          <AttributeValue
DataType="&xml:string">purchase order</AttributeValue>
        </ResourceMatch>
      </Resources>
      <Actions><Action>
        <ActionMatch MatchId="string-match">
          <AttributeValue>Submit</AttributeValue>
        </ActionMatch>
      </Actions>
    </Target>
  </Rule>
</PolicySet>
```
The above policy sets have been implemented using XACML and an application built using Python to investigate the performance of RBAC in the campus grid VO setup described in the previous section by replacing MAC with RBAC. The VO consists of five colleges participating and sharing resources as shown in Figure 4.4. From the Figure it is seen that 4 colleges are participating in the VO with resources shared as shown in Table 4.1.

Table 4.1 Resource snapshot provided by college

<table>
<thead>
<tr>
<th>VO member</th>
<th>Number of Resources Provided</th>
</tr>
</thead>
<tbody>
<tr>
<td>College 1</td>
<td>6</td>
</tr>
<tr>
<td>College 2</td>
<td>4</td>
</tr>
<tr>
<td>College 3</td>
<td>3</td>
</tr>
<tr>
<td>College 4</td>
<td>1</td>
</tr>
</tbody>
</table>
Figure 4.4 The proposed grid network

The grid network operates in the Master / Slave mode with one node from college 1 acting as the Resource allocator and the network consisting of 7 switches. Policies are implemented using XML (Eastlake et al., 2002) with basic Discretionary Access control. The task deployment file consists of the following parameters

- Total number of tasks to run
- Computation size of task
- Communication size of task
- The resources to use based on the access policy.
- RBAC policies for authentication and authorization
4.5.3 Results and Discussion

All users are assigned 200 jobs with computation size of 5000000 and communication size of 100000. The results obtained to compute these jobs by each user is shown in Table 4.2 and Figure 4.5

Table 4.2 Computation time of tasks after implementing RBAC

<table>
<thead>
<tr>
<th>VO member</th>
<th>Time in seconds to complete with RBAC Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>College1</td>
<td>17.522</td>
</tr>
<tr>
<td>College2</td>
<td>17.933</td>
</tr>
<tr>
<td>College3</td>
<td>19.206</td>
</tr>
<tr>
<td>College4</td>
<td>23.705</td>
</tr>
</tbody>
</table>

Comparing the obtained results with MAC based deployment, it is seen that the time taken to complete the job increases from 1.34 to 2.79 times the time taken by DAC. This could be a serious disadvantage when a large number of small jobs are executed in the VO.

Figure 4.5 Performance comparison of RBAC and MAC
4.5.4 Conclusion

RBAC was implemented and the performance of the grid compared using regular access control mechanism. For a large number of small tasks, it is found the time taken for Authentication and Authorization is high compared to DAC.

To overcome the timing issue a centralized authorization model using Local Community authorization service (L-CAS) with a novel authentication model using SAML for assertion is proposed. The fine grained policies can be maintained in the L-CAS server while site level policies can be maintained on each host. Though the administrative and maintenance burden of site policies in local host could be higher, considerable time is saved in communicating with the resource user and resource provider.