CHAPTER III

ROLE BASED GRID AUTHORIZATION MODEL

3.1 Introduction

The previous chapter dealt with a literature survey on grid computing research with specific emphasis on grid security models and mechanisms. In this chapter we address one of the core security issues of grids namely protection of resources from unauthorized access. We provide solutions for direct as well as indirect authorization issues. Our solutions include the proposed architectural frameworks for single-domain (intra-domain) role-based authorization and cross-domain (inter-domain) authorization. These designs are based on the standard Role Based Access Control Model (RBAC) [124] and have been supported by implementation done on the Globus middleware platform. RBAC is yet to be fully utilized in grids.

Apart from authorization, secure inter-operability has been a growing concern for multi-domain computing systems like the grids [183]. This is due to the fact that a grid comprises diverse local administrative domains collaborating as a virtual domain. Through the proposed cross-domain (inter-domain) authorization framework, we solve the issue of secure inter-operability. As a first step in this direction, we propose a role-mapping architecture. A role in one administrative domain may have a different meaning in another domain as roles reflect the responsibilities of the users in the organization [171], [71]. We establish role-equivalence among the domains by mapping a local domain role to its equivalent global role.

This chapter is organized as follows. The first section presents the grid authorization and access control framework. We propose the system architecture for single domain (intra-domain) authorization in the next section. Section 3 depicts the cross-domain (inter-domain) role-mapping and authorization architecture. In the last section, we summarize the contributions made in this chapter.
3.2 Grid Authorization and Access Control Framework

We address the problem of authorization in a grid environment using RBAC. Mandatory authorization schemes were initially used to define access control policies [194], [80]. The evolution of RBAC as a reliable standard for single enterprises [19], [91], [78] motivated researchers to think of ways in which it could be incorporated into grid environment. A grid is often viewed as a logical organization formed of multiple physical organizations or enterprises and hence the integration of RBAC into grid is a logical extension of the standard RBAC implementation. For a grid system usually formed by multiple domains which are maintained by different companies, organizations or institutions, inter-operability is a major issue.

A role, though the basic unit of access control as per the RBAC, signifies different meanings in different organizational contexts. A mechanism has to be in place by which we can map the role of one enterprise into its new semantics in another enterprise. Delegation and revocation mechanisms also are based upon role. In our work, we present an architecture for implementation of access control through authorization with supportive delegation and revocation mechanisms for a single domain environment. Later, we extend this work for authorization of users across various domains.

3.2.1 Requirements

- We need an access control model which is suitable for an organization’s requirements and is also scalable for grid. Role Based Access Control is suitable because of its ability to represent real world organizational role hierarchy [41] and scalability. In order to implement an access control model we need:

1. Access control policies:

   An access control policy is a security policy consisting of a set of rules that an organization adopts to govern who can have access to what resource. In RBAC, access control is based on the role of the user and not on the user’s identity.

   XACML is used as the policy language to express the access control information. The language provides built-in features to express complex access control policies.
2. **Framework to operate with policies:**

   We need a mechanism which can make the access decision based on user’s role, the resource being accessed, the action being performed by the user on the resource

   An XACML framework is available to operate with the access control policies written in XACML.

   - We require a data store, which holds user-role, role-hierarchy, role-delegation and such information.
   - A Graphical User Interface (GUI) for the administrator to perform operations specified in the functional specifications of RBAC. (assign/delete roles to users, specify the role-hierarchy).
   - We need a GUI for the Policy Administrator to create, delete and modify the access control policies.
   - GUI for the clients to know the information about user-role relationships in the organization, support for delegation of his/her role to other users and the log information about delegation (to whom he/she delegated and what are his/her delegated roles).
   - We need a server which is populated with the details of users and resources with in the organization so that clients can know the organizational structure

3.3 **System Architecture for Single Domain Grid Authorization**

We suggest an architecture for authorization in a grid environment envisaged as single enterprise. We also design a back end policy database. The main components of our framework include the following.

- **XACML Policy Framework:** It is designed as a separate service running on remote system. It accepts the requests from multiple authorization modules of the middleware and makes access decisions. Policies are specified using XACML. These policies have been placed on the same system where the XACML framework resides
• **Globus container:** This component contains all the services and accepts requests from the grid clients and runs on a separate system

• **Database:** It contains information about user-role, role-delegation etc. The Policy Enforcement Point (PEP) of the policy framework queries the database for user, role and permission details

• **LDAP (Lightweight Directory Access Protocol) server:** It runs on a separate system and accepts requests from the LDAP clients. It contains the complete details of users and resources in the organization. It also responds to the queries from the Policy Enforcement Point (PEP) for additional user attributes

• **Admin GUI:** An interface to enable administration of the access control decisions, it resides in the same system where policies are stored

• **Client GUI:** It is an interface which enables the grid client to send access requests. It resides in the grid client system

The architectural components and their interfacing is presented in a basic framework as shown in Figure 10, which is a simplified version of the detailed system with the components for a single domain authorization and their associations shown in Figure 11.

An administrator in the proposed architecture has the following functionalities.

• **Create Organizational Hierarchy:** The administrator specifies the roles at each level

• **Assign Roles to Users:** The administrator assigns roles to users. The constraint in role assignment is that one user has at most one role only. We do not deal with the Separation of Duties (SoD) and cardinality issues of RBAC as these are not in the purview of our work

• **Modifying User-Role Information:** The administrator can delete or change user’s roles. The administrator has to maintain all the details about users and resources of the organization

• **Specify Access Control Policies:** The administrator is responsible for creating new policies, modify and delete the access control policies according to
the organizational needs. The access control policies are based on the role information of the user.

- **Check the Audit files:** The authorization module in the grid node logs information about user activities. The administrator checks these log files and takes appropriate decisions.

The administrator needs to interface with the policy database to assign/manipulate the user-role or role-permission information. It also needs to have another interface for extracting additional user attributes from the Lightweight Directory Access Protocol (LDAP) server. Thus, the two interfaces for the Administrator include:

- **Interface to Database:** With this interface, the Administrator can do all the user-role, role-permission assignments.

- **Interface to LDAP server:** Support for adding, deleting and modifying entries in the directory at the LDAP server.
Figure 11: High Level System Architecture for Single Domain Authorization
• **Policy Interface:** This interface helps the administrator to create new access control policies or update the existing policies.

**Client Side**
A client in the grid environment must have a valid credential in order to access the resources. A grid client is presumed to have obtained a user certificate from the Certificate Authority (CA). Generally clients initiate proxies for most of the operations in the grid. The proxy-certificate has limited life time, generally 12 hours. We do not deal with the authentication aspects of grid client as it is not in the purview of our work. We provided two interfaces for the client:

- **Interface to LDAP server:** With this interface the client can query the LDAP server and get information regarding other users and resources in the organization.
- **Interface to Database:** Using this interface the client can find user-role information, log information about past delegations by the client and to the client

**Server side:** When the Globus container is started, it displays all the services which are currently available. Whenever a request for a service comes,

1. The container first checks for the user’s valid credential (identity) using the GSI mechanism available with it
2. The custom authorization module controls access to the service with the help of the decision engine (XACML Framework). It also takes account of the system’s dynamic attributes (free memory, cpu load) to take final authorization decision. It logs information about the request, the response and other attributes.

3.3.1 **Access Control Mechanism**
The access control mechanism which we implement takes the following inputs: the user’s selected role (by default, user’s original role is activated), the services the user requests and the operations the user intends to perform. The outcome of the access control decision i.e. *deny* or *permit* will be logged. The access control mechanism involves three modules namely Custom Authorization Module,
Policy Enforcement Point (PEP) and Policy Decision Point (PDP) according to the XACML [1] standard.

3.3.1.1 Custom Authorization Module

The custom authorization module works in one of the following ways. If the service is not configured to use custom authorization plug-in, then the global client is mapped to its local identity and the client can then access the service. It is a form of coarse-grained access control. Otherwise if the request comes to a service which is configured to use custom authorization plug-in, then the corresponding module will be called. This module extracts the details of the client from the client-request.

- client’s identity (distinguished name)
- client-requested service (set of operations)
- particular operation in the service

The authorization module sends the request (user’s Identity, service name, operation) to the XACML framework (decision engine) via the network.

3.3.1.2 Policy Enforcement Point

The decision engine which runs on a separate system accepts the request. The Policy Enforcement Point (PEP) tries to get the client’s selected role, provided the user explicitly selects one of its delegated roles. Otherwise, it tries to get the client’s default role (original role) from the database, using client-identity (distinguished name) as the key. If PEP does not get any meaningful role for the client, it sends response to the authorization module with Deny as the decision. Otherwise, the PEP gets the role for the client which may be one of the user’s delegated roles or the original role. If PEP gets the delegated role of user, then it tries to get the information about the delegated role’s expiry period (date, time). If PEP gets the user’s original role, it returns the ResponseContext. In case of delegated role, PEP takes current date and time, if \( \text{current}(\text{date, time}) \leq \text{expiry}(\text{date, time}) \) of delegated role, then through the Policy Decision Point (PDP), the query will be sent to find out the applicable policies. If expiry date and time is less than the current date and time, then it returns Deny to the authorization module. If all the above conditions
are satisfied, the PEP may query the LDAP server to get more attributes about the user i.e, for more fine-grained access control policies. Then the PEP formulates the RequestContext with the available information like

- user’s role
- resource/service in request
- action/operation
- other attributes of the user/resource (optional)

and sends the RequestContext to Policy Decision Point (PDP).

3.3.1.3 Policy Decision Point

The Policy Decision Point (PDP) parses the RequestContext and extracts the attribute values of elements

- subject (user’s role)
- resource (service name)
- action (operation)

The PDP notes the time when it received the RequestContext, checks whether it is within the time-range for accessing a service or not. The time-range for resource access is specified in XACML. If the time when the RequestContext is received is not in the time-range of the service, then PDP sends the ResponseContext to PEP with Deny as the final decision. Otherwise, it continues with the next steps. The PDP uses the extracted attributes (from RequestContext) to find the set of applicable policies. For this the PDP matches

- Policy target or
- Rule target
- may be both with the extracted attributes

If any policy or rule target matches with these attributes, then that policy is said to be applicable for this Requestcontext. If there are no policies applicable for the
RequestContext, the PDP sends the ResponseContext to PEP with decision as "Not Applicable". If more than one rule or policy is matched with the RequestContext, the PDP uses the Rule Combining Algorithm or the Policy Combining Algorithm to combine the individual decisions and draw the final decision. The rule or policy combining algorithm is specified in the policy or policy set. Some of the policy/rule combining algorithms are:

- **Deny-overrides**: It allows a single evaluation of *Deny* to take precedence over any number of permit, not applicable results

- **Permit-overrides**: It allows a single evaluation of *Permit* to take precedence over any number of deny, not applicable or indeterminate results

If any final decision is drawn, then PDP prepares the ResponseContext with the final decision either *Deny* or *Permit* and sends the decision to PEP.

The sequences given in Figure 11 work as follows. In step 1, a query for a user’s role is sent to the database and once the results are fetched, the grid client sends a request for a particular service to the grid container as shown in step 2. In step 3, the authorization module deployed as part of the Globus container sends the user’s identity and the required resource action (operation) to the Policy Enforcement Point (PEP) [Appendix B]. The PEP then sends the user identity to the *rbac* database and fetches the corresponding role as in step 5. In step 6, the PEP queries the LDAP server for additional user attributes. Step 7 shows the reply for this query. After this the PEP sends a RequestContext to the policy Decision Point (PDP) [Appendix B] as shown in step 8. In step 9, a query for applicable policies is sent to the access policy module and the set of applicable policies fetched in step 10. Next in step 11, the PEP receives the ResponseContext from PDP. Based on the ResponseContext, the PEP now sends the access decision to the authorization module. The final authorization decision is made after incorporating the dynamic attributes of the server like the available memory, CPU load etc as shown in step 14. These sequences have been consolidated in the sequence diagram for the single domain authorization mechanism as in Figure 12.
Figure 12: Sequence Diagram for Single Domain Authorization
3.3.2 Database Design

The access control mechanism works in close association with the data base. The data base contains user-role and role-permission mappings. Apart from this, the database also contains information regarding other mappings like role-level, role-delegate, and usr-password. Accordingly, we have designed a database named rbac with the following tables.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>role_level</td>
<td>(role, level)</td>
</tr>
<tr>
<td>usr_role</td>
<td>(usr, role)</td>
</tr>
<tr>
<td>usr_delegate</td>
<td>(fusr, tusr, role, cdate, ctime, edate, etime, flag)</td>
</tr>
<tr>
<td>role_act</td>
<td>(role, service, act1, act2, act3, act4, act5)</td>
</tr>
<tr>
<td>usr_pass</td>
<td>(usr, pass)</td>
</tr>
<tr>
<td>usr_selected</td>
<td>(usr, role)</td>
</tr>
</tbody>
</table>

The details of each of the tables in the database is as follows.

- **role_level**: It contains details about roles at each level within the organization. *role* is the primary key for this table

- **usr_role**: contains information about the roles of the users. We have restricted to single role per user. *usr* is the primary key for this table

- **usr_delegate**: It gives the delegation information of the user, who is delegated *(fusr or from user)* what role *(role)* to whom *(tusr or to user)*, when it is delegated *(cdate or commencing date, ctime or commencing time)* and for how much time it will be valid *(edate or end date, etime or end time)*, whether the role is further delegatable or not (flag)

- **role_act**: Contains details about what are the services and the operations of that service available to a particular role
• **usr_pass**: This contains information about user’s login name and password details

• **usr_selected**: Contains the details of the user’s selected role

### 3.4 System Architecture for Cross-Domain Authorization

A grid system usually consists of more than one domain in a hierarchical/nested fashion. Therefore, cross domain authorization is an essential factor for grid-wide multi domain access control. The first step is to map the role of a given domain to equivalent role in another domain. If the request is from a client in a different domain, we map the roles using the concept of role ranking. Normally, the roles in a domain follow a hierarchical relationship. We make use of this hierarchy, to give the roles a rank on a scale of 10. The request for the resource is passed on to the central authorization server which passes the request to the Authorization Server (AS1) of the domain in which the user is a part of.

The AS1 retrieves the user’s role and ranks it in its domain, creates a token and passes it on to central authorization server. It adds the rank for the domain and normalizes the rank of the client on a scale of 1 and passes the token to AS2. Here, the subsequent rank of the source is added, normalized and compared with the rank of the client. If the required rank is greater, then access is denied, otherwise it is granted. If there are more sub domains, then we find the normalized rank with respect to the first common ancestor between the client and the resource. If the user wants to delegate the role, then the user passes on the produced token with the normalized roles. If the user who is being delegated is from a different domain, then the role normalization is done again with respect to the correct ancestor and a token is recreated.

The whole grid environment is separated into different domains and sub domains. The sub domains in every domain are given ranks on a scale of 10. The roles in a local domain are also ranked on a scale of 10 based on their hierarchy. The role in a local domain is translated to its global ranking based on the value of its own ranking and also the rank of its ancestor domains up the tree. We represent the role-mapping architecture as a weighted tree and arrive at the globally mapped...
role by comparing the global rank of a role with respect to its first common ancestor. The hierarchy of domains is represented as in Figure 13.

![Figure 13: Hierarchy of Domains](image)

The details of the user’s delegated and revoked roles between two domains are stored with their common ancestor central server. As detailed in Chapter I, the RBAC standard uses role as the basic unit of authorization and incorporates features such as role hierarchy [102], static and dynamic separation of duties and so on [75]. In a typical RBAC environment, a user is assigned roles based on the responsibilities of the user in the organization. For example, in a university domain, the potential roles could be Professor, Associate Professor and so on. For an industrial domain the roles could be CEO, General Manager, Manager and so on. The semantics of roles in a given domain will not have relevance in another domain. The role in an organization has to be mapped to its corresponding meaning in another to make cross-domain resource sharing possible.

We address this issue with a ranking based weighted role approach. Our architecture enables mapping of a local role to a global ranking. We consider a nested and hierarchical domain architecture reflecting the real life grid scenario. The roles in a particular domain follow a local role hierarchy. The cross domain architecture consists of the following components.
• At the organizational level we consider two Domains A and B

• Domains A and B consist of sub-Domains A_a and B_b

• Further, Domains A_a and B_b have grid nodes as their constituents

• Authorization Server1 (AS1) is the local Authorization server for grid nodes from Domain A_a

• Authorization Server2 (AS2) plays a similar role in Domain B_b

• Ranking servers RS1 and RS2 for the two respective Domains A and B store the rating of the subdomains

Figure 14 depicts the role mapping architecture which forms the basis for ranking of roles across domains. A simple view of the individual authorization server architecture is shown in Figure 15.

Figure 16 shows the user-role ratings for authorization.

The grid user is granted/denied access to the requested resource through the following procedure.

1. The user from Domain A_a sends his identity, path, the requested resource and also requested operation to Domain B_b

2. The user in domain b forwards the details to its Authorization Server (AS2) and awaits a deny or grant

3. The Authorization Server AS2 executes the algorithm Authorize

4. The credentials are passed up the hierarchy for role mapping as the user is from a different domain

5. The credentials reach AS1 where the attributes like the user’s role, rating etc are retrieved and a Token is created.

6. The Token follows the same path in reverse and at every stage, the rating of the domain gets weighted

7. AS2 gets the final version of the Token. The general expression by which the user rating is computed is as follows.
Figure 14: Cross Domain Role Mapping Architecture
Figure 15: Authorization Server Architecture

Figure 16: User-Role Ratings for Authorization
The normalized user rating = (local rank of the role * rank of the sub domain * rank of the domain) / 10 * 10 * 10
For example, for a grid user A3 in Domain A the normalized user rating will be 7*5*8 / 10*10*10 = 0.280

8. AS2 finds the minimal rating of the role needed to access the resource. The normalized rating of the resource is similarly found as 10*8*5 / 10*10*10 = 0.400

9. AS2 can integrate this ranking comparison with other local policies to either deny or grant access to the user

Here, we give a generic algorithm for mapping the role of a domain to its equivalent role in another domain as shown in Algorithm 1. We also describe an algorithm for authorization which is presented in Algorithm 2.

---

**Algorithm 1 Rolemap**

- **Input**: User Credentials (Token)
- **Output**: Token containing updated values

```
begin
  if(user is from different domain or sub-Domain)
    Token T = call Rolemap(credentials)
    Add rating of the sub-Domain to the Global rating of the resource in the Token
    return T
  if(user is from same Domain)
    Token T = call Authorize(Credentials)
    Add Rating of Sub-Domain to T
    Return T
end
```
Algorithm 2 Authorize

**Input**: User Credentials  
**Output**: Grant/Deny a Token  

```
begin
    If(user is from a different domain)
        call Rolemap(credentials)
        find minimum rated role to access resource in the domain
        find the normalized global rating of that role GLR
        retrieve the Normalized global rating of user from certificate GLU
        if GLU ≥ GLR
            return accept
        else
            return deny
    else
        retrieve user roles, rating and other credentials from database
        if(resource is from other domain)
            create token T containing the user details
            return T
        else
            find minimum rated role MR to access resource in the domain
            if(MR > user’s role-rating) return accept
            else return deny
end
```

### 3.5 Chapter Summary

The framework proposed in this chapter contributes to the enforcement of grid computing security in a major way. First, we propose architectural frameworks for authorization within a grid enterprise. Later we extend it to suit the requirements of authorization across domains, where inter-operability and security go hand in hand. The role mapping architecture helps the grid nodes across domains to map the roles and thereby interact and authorize users for resource access. This architecture also supports reusability of role-ranking mechanism, as the token once created between two domains can be used for future interactions between them. Domains can also formulate additional access control policies like giving access to only semantically sensible organizations for that particular resource, apart from just comparing the global ranking of the user. Both the architectures have also been represented through sequence diagrams. In total, these architectures form the core of the grid access control and authorization systems.