Dynamic Role and Context-Based Access Control for Grid Applications[1]

Hanbing Yao, Heping Hu, Baohua Huang, Ruixuan Li
College of Computer, Huazhong University of Science and Technology, Wuhan 430074, China
hustyhb@163.com

Abstract

Despite the recent advances in access control approaches applicable to grid computing, there remain issues that impede the development of effective access control for grid applications. Amongst them are the lack of context-based models for access control, and reliance on identity or capability-based access control schemes. In this paper, we propose RCBAC model which extends the RBAC with context constraints. The RCBAC mechanisms dynamically grant and adapt permissions to users based on a set of contextual information collected from the grid environments, while retaining the advantages of RBAC model.

1. Introduction

The term "Grid" refers to systems and applications that integrate and manage resources and services distributed across multiple control domains [1]. The GSI has been accepted as the primary authentication mechanism for the Grid computing [2]. It is widely used and has been integrated into a number of Grid applications. However, the authorization and access control challenges are not fully addressed by existing approaches. While many research efforts address important aspects of the access control problem in grid environment, these efforts focus on relatively static scenarios where access depends on the user's identity [3,4,5]. They do not address access control issues for grid applications where the access capabilities and privileges of a subject not only depend on its identity but also on its security-relevant contextual information, such as time, location, or environmental state available at the time the access requests are made. These context parameters are critical to the effectiveness of the resulting access control scheme.

The remainder of the paper is organized as follow: Section 2 presents a formal definition for RCBAC. Section 3 describes RCBAC for grid application. Section 4 concludes this paper.

2. RCBAC Model

The RCBAC dynamically grant and adapt permissions to users based on a set of contextual information collected from the grid environments. The model extends the RBAC [6] with context constraints, while retaining its advantages (i.e. ability to define and manage complex security policies). RBAC addresses many other issues such as role activation, revocation, role hierarchies and separation of duty constraints. These issues apply to RCBAC as well.

2.1 A formal definition for RCBAC

Based on the formalization of the RBAC model, we present a precise description of RCBAC model that includes security-relevant contextual information [7]. Both role hierarchies and separation of duty in RBAC are meaningful in the RCBAC, though they are omitted here in our description. We only consider flat user and security-relevant contextual information. This formalization can be extended to hierarchies and constraints similar to the RBAC1 and RBAC2 models. An overview of the RCBAC is presented in Fig 1. We keep USERS, ROLES, OBS, OPS, PRMS and SESSIONS in the RBAC.

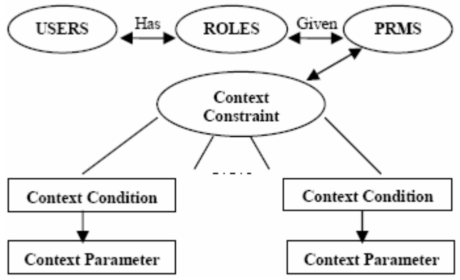


Fig 1. RCBAC

Definition 1. Context Parameter (CP): A context parameter is represented by a data structure p, having the following fields: name ∈CN, type ∈CT, and a function getValue(). The CN is a set of the possible

[1] This work is supported by National Natural Science Foundation of China under grant 60403027.

names of context parameters, and the CT is a set of types of context parameters, and the function of getValue() get the runtime value of CP. For example, the set CN may be defined as: CN = {time, location, duration, system_load}, with the corresponding set CT defined as: CT = {Time, String, Long, Integer}.

Definition 2. Context Set (CS): A context set CS consists of n context parameters {CT1, CT2 ⋯, CTn}, $n \geqslant 0$, for any CTi, CTj, with $i \neq j$ and $1 \leqslant i, j \leqslant n$, we have that CTi.name ≠ CTj.name (i.e. the parameter names must be distinct). By analyzing the grid application security requirements, application designers determine which context types will be used to specify access policy. Although the context set is determined before the application implementation, system administrators can dynamically add new ones when needed.

Definition 3. Context Condition (CN): CN = <CT > <OP> <Value>, CT∈CS, OP is a standard comparison and logical operator, VALUE is a specific value, and the type of VALUE is CP.type.

Definition 4. Context Constraint (CC): CC = CL1 ∪ CL2 ⋯ ∪ CLn, CL = CN1 ∩ CN2 ⋯ ∩ CNn. CN1,CN2 ⋯,CNn are context condition. Based on this format, our access control schema is capable of specifying any complex context related constraint to describe all kinds of security requirements. System administrators can dynamically adapt context constraint.

Definition 5. RCBAC: RCBAC = {USERS, ROLES, OBS, OPS, PRMS, SESSIONS, CC}. The USERS, ROLES, OBS, OPS, PRMS and SESSIONS are defined in RBAC, the CC is context constraint.

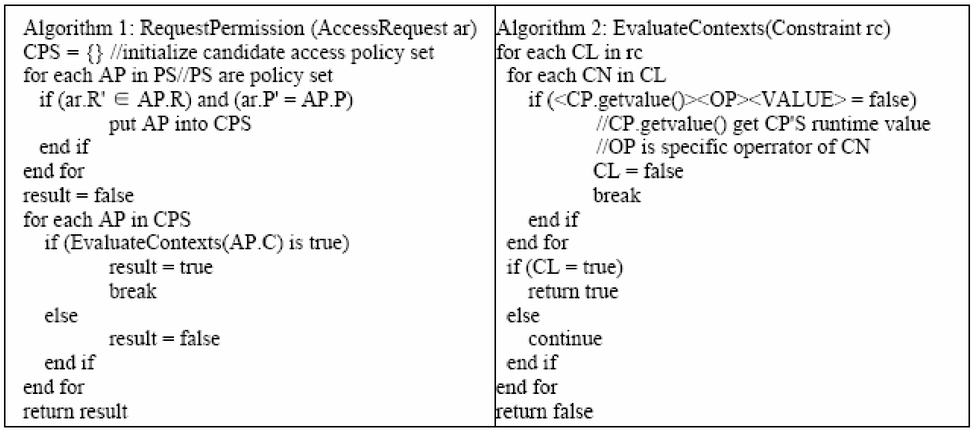
Definition 6. Access Policy (AP): We define an access policy as a triple, AP = (R, P, C), R∈ROLES, P ∈PRMS, C∈CC. If C is empty then this policy reverts to simple RBAC.

Definition 7. Access Request (AR): We define access request as a triple, AR = (R′ , P′ , RC), R′ ∈ ROLES, P′ ∈PRMS, RC(runtime context) is a set of values for every context type in the Context Set. That is, RC = {CT1.getvalue(), CT2.getvalue(), ⋯ , CTn.getvalue()}, {CT1, CT2 …, CTn} is the context set (CS) of the grid application.

An access request is granted only if there exists an access policy AP (R, P, C), such that R′ ∈R, P′ = P, and C evaluates to true under RC (that is, when all CPi in context constraint C are replaced with their values in RC, then the resulted Boolean expression is true).

2.2 Dynamic Context Evaluation Algorithms

We can design the basic algorithm to determine whether an access request is authorized or not based upon the context parameter in our model. Fig 2 illustrates the algorithm.



```
Algorithm 1: RequestPermission (AccessRequest ar)
CPS = {} //initialize candidate access policy set
for each AP in PS//PS are policy set
   if (ar.R' ∈ AP.R) and (ar.P' = AP.P)
            put AP into CPS
   end if
end for
result = false
for each AP in CPS
   if (EvaluateContexts(AP.C) is true)
            result = true
            break
   else
            result = false
   end if
end for
return result
```

```
Algorithm 2: EvaluateContexts(Constraint rc)
for each CL in rc
  for each CN in CL
     if (<CP.getvalue()><OP><VALUE> = false)
              //CP.getvalue() get CP'S runtime value
              //OP is specific operrator of CN
              CL = false
              break
     end if
  end for
  if (CL = true)
     return true
  else
     continue
  end if
end for
return false
```

Fig 2. Algorithms for RCBAC

The application passes an Access Request (ar) to the algorithm RequestPermission, and receives a Boolean value in return - indicating whether the attempted operation should be allowed, or not. The ar contains the caller's roles and permissions and context constraints. The access control system first checks whether the application's Access Policy contain the user's Access Request. Then the context constraints are populated by plugging in values from the application's runtime environment.

3. RCBAC for Grid Applications.

An overview of the RCBAC for Grid applications is presented in Fig 3. The RCBAC model ensures the users can access Grid resources only if they have appropriate privileges and capabilities. As the Grid environment is dynamic, this requires dynamic context aware access management. In our implementation, users entering the Grid application using the portal are assigned a set of roles when they log in. A context agent is then locally set up for each user, which dynamically adjusts the user context. Similarly, the context agents are set up at the application for each role that will access it, which similarly dynamically adjusts the application context.

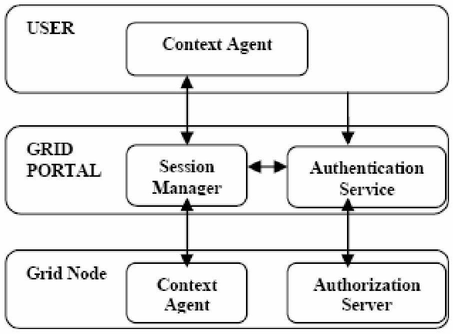


Fig 3. RCBAC framework for grid applications

As an illustration, assume that the following access request is submitted for evaluation to the grid application: <R="guest", P="view", C={p1{time, Time}, p2{location, String}, p3{duration, Long}, p4{system_load, Integer}>.

The context recorded at the time of access request is captured by context agent, and provided to the system as part of the request. Now, assume that the following AP is applicable to the permission P:

<R="guest", P="view", C=CC>

CC = CL1 ∩ CL2 ∩ CL3 ∩ CL4 s.t.

CL1: {time > 9:00} AND {time < 17:00)

CL2: {location = "admin1"} OR location = "admin2"}

CL3: {duration≤600s}

CL4: {system_load != "high">

Based on this information, the system would return an authorization decision for this access request. The available contextual information indicates that the access conditions are satisfied.

4. Conclusion and Future Work

In this paper we described the model that extends the traditional RBAC model to gain many advantages from its context-aware capability. Our new access control infrastructure is dynamic and distributed with these advantages:

1. The RCBAC model extends traditional RBAC by associating access permissions with context-related constraints. Every constraint is evaluated dynamically against the current context of the access request. Therefore, the model is capable of making authorization decisions based upon context information in addition to roles.

2. The context-based access control is applied dynamically. At design time, administrators have great flexibility to specify complex context-aware authorization policies. At run-time, our authority service can enforce context-based policy automatically because it is not statically bound to any application.

3. Context information is separated from the main business logic of target applications. Since every context type definition is independent of the specification of the access rules, any change to them has no effect on other parts of the system. Our security infrastructure is flexible and permits easy extensibility.

We plan to explore the interplay of contextual conditions in the presence of separation of duty constraints and role hierarchies. In these situations, it is critical to ensure that the access to grid resources based on inherited permissions do not violate any separation of duty constraints.

References

[1] Foster, I., Kesselman, C. and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, 15(3). 200-222. 2001.

[2] Foster, I., Kesselman, C., Tsudik, G. and Tuecke, S. A Security Architecture for Computational Grids. ACM Conference on Computers and Security, 1998, 83-91.

[3] Pearlman, L., Welch, V., Foster, I., Kesselman, C. and Tuecke, S. "A Community Authorization Service for Group Collaboration", Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, p.0050, Monterey, CA, 2002.

[4] Alfieri, R., Cecchini, R., et. al., VOMS, an Authorization System for Virtual Organizations. European Across Grids Conference, 2003, 33-40.

[5] Chadwick, W., Otenko, A. The PERMIS X. 509 role-based privilege management infrastructure. Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies, Monterey, Califorinia, USA. 2002.

[6] Sandhu, R., Coyne, E., Feinstein, H. and Youman, C. Role-Based Access Control Models, IEEE Computer, 29(2): pp.38-47, 1996.

[7] McDaniel, P. On Context in Authorization Policy. Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies, June 2003, Como, Italy.