Integrating Attributes into Role-Based Access Control

Qasim Mahmood Rajpoot, Christian Damsgaard Jensen, Ram Krishnan

Integrating Attributes into Role-Based Access Control

Qasim Mahmood Rajpoot[1], Christian Damsgaard Jensen[1] and Ram Krishnan[2]

[1] Department of Applied Mathematics & Computer Science
Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark
[2] Dept. of Elect. and Computer Engg.
University of Texas at San Antonio, USA
{qara, cdje}@dtu.dk, ram.krishnan@utsa.edu

Abstract. Role-based access control (RBAC) and attribute-based access control (ABAC) are currently the most prominent access control models. However, they both suffer from limitations and have features complimentary to each other. Due to this fact, integration of RBAC and ABAC has become a hot area of research recently. We propose an access control model that combines the two models in a novel way in order to unify their benefits. Our approach provides a fine-grained access control mechanism that takes into account the current contextual information while making the access control decisions.

Keywords: Context-aware access control, RBAC, Attributes, Role-permission explosion

1 Introduction

RBAC [6] is the most popular access control model and has been a focus of research since last two decades. The RBAC paradigm encapsulates privileges into roles, and users are assigned to roles to acquire privileges, which makes it simple and facilitates reviewing permissions assigned to a user. It also makes the task of policy administration less cumbersome, as every change in a role is immediately reflected on the permissions available to users assigned to that role.

With the advent of pervasive systems, authorization control has become complex as access decisions may depend on the context in which access requests are made. The contextual information represents a measurable contextual primitive and may entail such information being associated with a user, object and environment. It has been recognized that RBAC is not adequate for situations where contextual attributes are required parameters in granting access to a user [12]. Another limitation of RBAC is that the permissions are specified in terms of object identifiers, referring to individual objects. This is not adequate in situations where a large number of objects in hundreds of thousands exist and leads to role-permission explosion problem.

Attribute-Based Access Control (ABAC) [17] has been identified to overcome these limitations of RBAC [5]. ABAC is considered more flexible as compared to

RBAC, since it can easily accommodate contextual attributes as access control parameters [12]. However, ABAC is typically much more complex than RBAC in terms of policy review, hence analyzing the policy and reviewing or changing user permissions are quite cumbersome tasks. As discussed above, both RBAC and ABAC have their particular advantages and disadvantages. Both have features complimentary to each other, and thus integrating RBAC and ABAC has become an important research topic [5], [10], [9]. Recently, NIST announced an initiative to integrate RBAC and its various extensions with ABAC in order to combine the advantages offered by both RBAC and ABAC. We take a step in this direction and present the idea of an integrated RBAC and ABAC access control model, called the Attributes Enhanced Role-Based Access Control model. The proposed model retains the flexibility offered by ABAC, yet it maintains RBAC's advantages of easier administration, policy analysis and review of user permissions.

The rest of the paper is organized as follows: Section 2 summarizes related work and compares our approach to prior work. In Section 3, we present the components of the proposed access control model and how access control decisions may be calculated. Section 4 discusses potential benefits offered by the proposed approach and identifies future directions.

2 Related Work

The limitations of RBAC led to announcement of the NIST initiative to incorporate attributes into roles [12]. The initiative identified three possible ways in which roles and attributes may be combined. The first option is *dynamic roles*, where attributes determine the roles to be activated for a user. Al-Kahtani et al [1] and Kern et al [14] explored this option for automated user-role assignment, in large organizations, using attribute-based rules. These solutions consider only user attributes and do not address the issues of role explosion and role-permission explosion. In the second approach, roles and attributes may be combined in an *attribute-centric* manner. The roles are not associated to permissions; rather they are treated as just one of many attributes. This approach is essentially the same as ABAC and does not inherit any benefit from RBAC. In the third approach, called *role-centric*, roles determine the maximum permissions available to a user, and attributes are used to constrain these permissions. Kuhn et al [12] identify this approach as a direction for future research, since it may retain the advantages of RBAC while adding the much needed flexibility.

In response to the above initiative, Jin et al [10] present first formal access control model called RABAC using the role-centric approach. They extend RBAC with user and object attributes and add a component called permission filtering policy (PFP). The PFP requires specification of filtering functions in the form of Boolean expression consisting of user and object attributes. Their solution is useful to address the role-explosion problem and as a result facilitates user role assignment. However, the approach does not incorporate environment attributes and is not suitable for systems involving frequently changing attributes,

e.g., location and time. Also, we make a fundamental modification in RBAC by using attributes of the objects in the permissions, addressing the issue of role-permission explosion. Huang et al [9] present a framework to integrate RBAC with attributes. The approach consists of two levels: underground and aboveground. The underground level makes use of attribute-based policies to automate the processes of user-role and role-permission assignment. The aboveground level is the RBAC model, with addition of environment attributes, constructed using attribute-based policies. Their work is different than ours in that it focuses on automated construction of RBAC.

Several efforts have been reported which extend RBAC to include the context of access. Examples include: environment roles [2], spatio-temporal RBAC [16], context-aware RBAC [13] and others. They add context into RBAC, however these extensions typically require creation of a large number of closely related roles, causing the role-explosion problem. Ge et al [8], and Giuri et al [7] focus on resolving the issue of role explosion by providing the mechanism of parametrized privileges and parametrized roles. However, the permissions in these solutions refer to objects using their identifiers. Few approaches propose a variant of RBAC categorizing the objects into groups or types in an attempt to resolve the role-permission explosion issue [15], [3], [11]. Grouping the objects allows us to associate a single attribute with each object. The permissions are then specified using the group attribute, where each permission refers to a set of objects in that group. Moreover, as the number of object attributes grow, the number of groups (referred to as views in [11] and object classes in [3]) increase exponentially. This makes task of policy administration complex since for every new object to be added in the system it has to be associated with all those groups to which it belongs.

3 Components of the Proposed Model

This section presents the components of the proposed Attributes Enhanced Role-Based Access Control (AERBAC) model. We also discuss briefly how an access request may be evaluated. Figure 1 depicts our access control model and its components. The entities users, roles, objects and operations have the same semantics as in RBAC. Users and objects in our model are associated with attributes too. We also incorporate the environment attribute to fully capture the situation in which access needs to be authorized. Below, we first describe these attributes and then discuss semantics of the components in our access control model, including permissions, conditions, sessions and request evaluation.

Attributes: Attributes capture the properties of specific entities (e.g. user). We define an attribute function for each attribute that returns the value of that attribute. Each attribute is represented by a range of finite sets of atomic values. For example, the range of branch attribute is a set of names of branches semantically relevant for the application domain.

User attributes capture the properties of the user who initiates an access request. Examples of user attributes are title, specialization, location, security

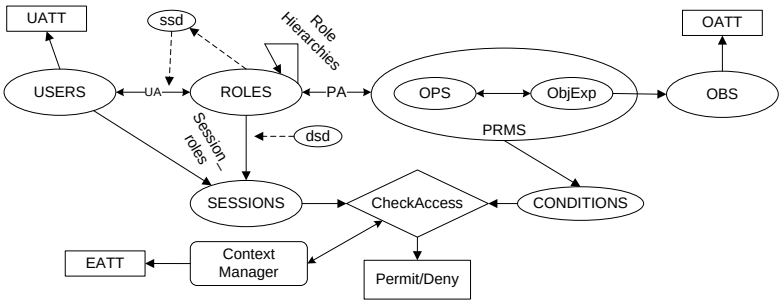

Fig. 1. Attributes extended role-based access control model

clearance etc. *Object attributes* are used to define the properties of the resources protected by the access control policy. Examples of object attributes include type, status, location, time of object creation etc. *Environment attributes* capture external factors of the situation in which the access takes place. Temperature, occurrence of an incident, system mode or other information which not only pertains to a specific object or user, but may hold for multiple entities, are typically modelled as environment attributes.

An attribute may be either static or dynamic. The values of *static attributes* rarely change e.g. designation, department, type etc. On the other hand, *dynamic attribute* values may change frequently and unpredictably, so they may well change during the lifetime of a session. This means that they may need to be checked more frequently, depending on the application requirements. Examples of such attributes include officer in command, location, occurrence of an incident etc. They are also referred to as contextual attributes in the literature [4].

Permissions and conditions: Our aim is to provide a dynamic, yet easy to manage and efficient solution to enforce the access control model. In our model, we achieve this by incorporating attributes associated with user, objects and environment in the permission. In contrast to the traditional approaches in RBAC, the permissions in AERBAC refer to objects indirectly, using their attributes, so a permission grants access to a set of objects having those attributes rather than a single object. A permission refers to a set of objects sharing common attributes, e.g. type or branch, using a single permission, in contrast to separate permissions for each unique object. This is particularly relevant in those domains where several objects share common attribute values. This helps in significantly reducing the number of permissions associated with a role, while increasing the expressiveness and granularity of access control in a role-centric fashion.

A permission consists of an object expression and an authorized operation on the object set denoted by the expression. Object expressions are formed using the attributes of objects. Each permission is associated with one or more conditions, which must be evaluated to be true in order for the user to exercise

that permission. A condition associated with a permission may contain attributes of the users, objects and environment. When a user requests a specific permission, it is granted to the user if the permission exists in the user's session and the associated condition is satisfied, i.e. the current values of attributes match the constraints given by the condition.

An example of a permission is: *p= ((oType(o) = secret oStatus(o) = active), read)* which states that a role having this permission can perform read operation on the objects denoted by the given object expression. Here *oType* and *oStatus* are object attribute functions that return the values of respective attributes for a given object. Suppose that the permission p is constrained by a condition *c= (uMember(u) = premium* $\wedge$ *time_of_day()* $\leq$ *uDutyExpire(u))* where *uMember* and *uDutyExpire* are user attribute functions that return the attribute values of a given user, whereas *time_of_day()* is an environment attribute function. This condition implies that, in order to be granted the permission *p*, the user must be a premium user and time of access must be before the end of user's duty timing.

Session: A session contains a list of permissions associated with the roles activated by the user. As described earlier, the permissions are different from standard RBAC permissions in terms of referring to the objects using their attributes and being tied with the conditions that are evaluated every time a permission is to be exercised. Hence, the CheckAccess function needs to be re-defined.

The Context Manager is responsible for propagating the updated values of dynamic attributes of the users, objects and environment. Depending on the application, some of these attribute values may also be provided by the user while placing an access request, however the application must ensure the authenticity of such information before using it.

3.1 Access Decisions

The main role of the access control mechanism is to verify whether a user *u*, requesting access to object *o*, using an operation *op*, is authorized to do so. An important consideration, in environments motivating the proposed approach, is that the user's request may also be based on the attributes of the objects. For instance, in a medical imaging application, a user might want to view all images containing specified characteristics e.g., objects with *type = tumor* and *domain = hospital-nw*. The permissions in AERBAC are constrained by conditions. For a user request to be granted, there must exist an object expression in the user's session that denotes the requested objects, and the condition tied to that object expression must be evaluated to be true, as illustrated in Fig. 2.

Request Evaluation: As mentioned above, a user request can either explicitly specify an object, by listing its identifier, or can implicitly denote a set of objects using the attributes of the objects. If the user request is not for a specific object but rather a set of objects, the system must consider the given criteria to return the requested objects. Once a user submits an access request, the request must be evaluated against the policy. The function checkAccess in RBAC needs to be modified such that it takes the user request as input, processes the request

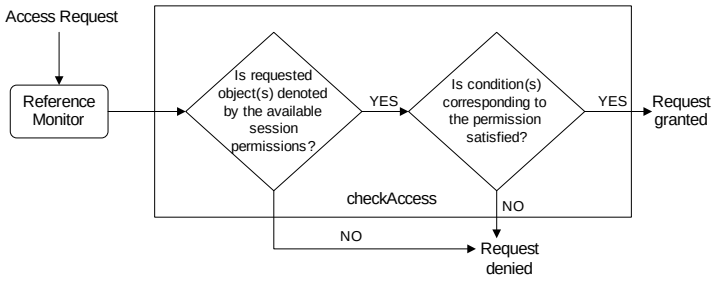

Fig. 2. Access request evaluation

as per the format of a given request, and returns the result. In the following, we elaborate on evaluation of both identifier-based and attribute-based requests.

a) Identifier-based request: In identifier-based request, the user specifies the identifier of the object to be accessed. The evaluation of such type of request is straight-forward. In this case, the input of the function checkAccess consists of a session *se*, an operation *m*, and an object *obj*. Recall that a permission consists of an object expression and an operation and is constrained by a condition. The checkAccess function returns true if and only if i) there exists a permission *p*, in the available session permissions in session *se*, that contains an object expression which evaluates to true for *obj*, ii) *m* matches *op*, and iii) the corresponding condition *c* evaluates to true.

b) Attribute-based request: Using the second form of request, the user may specify the attributes of the object in his/her request, rather than a unique identifier of the object. Specifying the object attributes in the request implies that the user wishes to access all those objects which have the specified attribute values. In this approach, an example user request could be: $Req = <se, (otype = secret \wedge odept = admin \wedge ostatus = inactive), write>$ which states that the owner of the session *se* wishes to exercise the *write* operation on the objects denoted by the given object expression. The checkAccess function receives as input the access request *Req* and returns the authorized objects to the user, if request is granted, otherwise the request is denied. The given expression is converted to a query and the resulting objects are retrieved from the resource database. The object expressions existing in the user's session are evaluated against each retrieved object and if an object expression and its corresponding condition evaluate to true for an object, the object is added into the list of authorized objects to be granted to the user. Finally, the user is granted access to all those objects for which an object expression and its corresponding condition return true. Since the object expressions are to be evaluated for each returned object, this approach may prove to be expensive in cases where several objects are returned by the query formed based on user's request. We plan to work further on it and propose a more efficient algorithm to process such a request.

4 Discussion And Future Work

Our motivation to integrate RBAC with attributes is to obtain advantages associated with both RBAC and ABAC, while addressing the limitations of RBAC and ABAC. In a pure ABAC approach, when a user request needs to be evaluated, the relevant rules are identified using the attributes associated with requesting user, requested object and current environment. These shortlisted rules are then evaluated one-by-one unless we find a rule which allows the request. In contrast, our approach requires evaluation of only those object expressions which are associated with the roles activated by a user in his/her session. Note that this would significantly reduce the number of rules to be evaluated. Compared to ABAC, our approach provides a systematic mechanism to evaluate a subset of policy rules which are determined based on the user's roles, yet retaining the advantages offered by RBAC including quick assignment and revocation of roles to users, reviewing of permissions assigned to a user or role, and reduced complexity of administration in large organizations. Our approach also provides significant advantages when compared to RBAC. First, RBAC requires that the permissions must refer to the objects based on their identity, which may lead to role-permission explosion issue in applications involving large number of objects. Our model allows to use object attributes in the permissions. Second, RBAC cannot easily handle dynamically changing attributes [5]. It typically does not support making contextual decisions unless many similar roles are created causing role-explosion problem. We provide a mechanism to incorporate these dynamically changing attributes in a role-centric manner, yet without requiring creation of a large number of roles.

The model we proposed integrates RBAC and ABAC bringing together the features offered by both models. In our model, the attributes may be associated with users, objects and environment allowing the request context to be considered in making access control decisions. In the future, we plan to work on formally defining the proposed model and provide a deeper analysis of merits offered by the proposed model as compared to existing access control model including ABAC and RBAC. Moreover, we aim to extend the model with continuous enforcement to deactivate a role or revoke a permission when context conditions fail to hold, and to develop an XACML profile of the proposed model.

Acknowledgments

The work of first two authors is supported by a grant from the Danish National Advanced Technology Foundation. The work of the third author is supported by a US National Science Foundation grant CNS-1423481.

References

1. Al-Kahtani, M. A., Sandhu, R.: A Model for Attribute-Based User-Role Assignment. In: Annual Computer Security Applications Conference, pp. 353-362. IEEE (2002)

2. Covington, M.J., Long, W., Srinivasan, S., Dev, A.K., Ahamad, M., Abowd, G.D.: Securing context-aware Applications using Environment Roles. In: Symposium on Access Control Models and Technologies. pp. 10-20. ACM (2001)
3. Chae, J. H., Shiri, N.: Formalization of RBAC Policy with Object Class Hierarchy. In: Information Security Practice and Experience, pp. 162-176. Springer (2007)
4. Covington, M. J., Sastry, M. R.: A Contextual Attribute-Based Access Control Model. In: On the Move to Meaningful Internet Systems Workshops, pp. 1996-2006, Springer (2006)
5. Coyne, E., Weil, T. R. ABAC and RBAC: Scalable, Flexible, and Auditable Access Management. IT Professional, 15(3), 14-16 (2013)
6. Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R., Chandramouli, R.: Proposed NIST Standard for Role-Based Access Control. ACM Transactions on Information and System Security (TISSEC), 4(3), 224-274 (2001)
7. Giuri, L., Iglio, P.: Role Templates for Content-Based Access Control. In: Workshop on Role-Based Access Control, pp. 153-159. ACM (1997)
8. Ge, M., Osborn, S. L.: A Design for Parameterized Roles. In: Data, Application Security and Privacy Conference, pp. 251-264. Springer (2004)
9. Huang, J., Nicol, D. M., Bobba, R., Huh, J. H.: A Framework Integrating Attribute-Based Policies into RBAC. In: Symposium on Access Control Models and Technologies, pp. 187-196. ACM (2012)
10. Jin, X., Sandhu, R., Krishnan, R.: RABAC: Role-centric Attribute-Based Access Control, In: Computer Network Security, pp. 84-96. Springer (2012)
11. Kalam, A. A. E., Baida, R. E., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miege, A., Saurel, C., Trouessin, G.: Organization based access control. In: 4th International Workshop on Policies for Distributed Systems and Networks, IEEE (2003)
12. Kuhn, D.R., Coyne, E.J., Weil, T.R.: Adding Attributes to Role-Based Access Control. IEEE Computer 43, 79-81 (2010)
13. Kulkarni, D., Tripathi, A.: Context-Aware Role-Based Access Control in Pervasive Computing Systems. In: Symposium on Access Control Models and Technologies, pp. 113-122. ACM (2008)
14. Kern, A., Walhorn, C.: Rule Support for Role-Based Access Control. In: Symposium on Access Control Models and Technologies, pp. 130-138. ACM (2005)
15. Moyer, M. J., Abamad, M.: Gener alized Role-Based Access Control. In: International Conference on Distributed Computing Systems, pp. 391-398. IEEE (2001)
16. Ray, I., Toahchoodee, M.: A Spatio-Temporal Role-Based Access Control Model. In: Data and Applications Security Conference, pp. 211-226. Springer Berlin Heidelberg (2007)
17. Yuan, E., Tong, J.: Attributed Based Access Control (ABAC) for Web Services. In: International Conference on Web Services, IEEE (2005)