Towards Role-based Authorization for OSGi Service Environments

Gail-Joon Ahn
Arizona State University
gahn@asu.edu

Hongxin Hu
Arizona State University
hongxinh@asu.edu

Jing Jin
UNC Charlotte
jjin@uncc.edu

Abstract

OSGi framework enables diverse devices to conveniently establish a local area network environment such as homes, offices, and automobiles. Access control is one of the crucial parts which should be considered in such emerging environments. However, the current OSGi authorization mechanism is not rigorous enough to fulfill security requirements involved in dynamic and open OSGi environments. This paper provides a systematic way to adopt a role-based access control approach in OSGi environments. We demonstrate how our authorization framework can achieve important RBAC features and enhance existing primitive access control modules in OSGi service environments.

1. Introduction

The Open Services Gateway Initiative (OSGi) provides an open, common architecture for service providers, developers, software venders, gateway operators and equipment vendors to develop, deploy and manage services in a cooperative fashion. In addition, OSGi service platform is an extensible integration platform used to remotely and dynamically deploy, provision, maintain and manage applications and services with multiple devices in networked environments [1]. In OSGi service platform, the User Admin service provides user authorization functionality. However, the User Admin service's authorization architecture in OSGi is not sufficient enough to support the highly dynamic and open OSGi environments. Thus, an enhanced access control mechanism is needed for achieving interoperability, extensibility, and ease of administration & management. To overcome such issues, role-based access control (RBAC) can be applied in such network environments to simplify authorization management by associating users with roles, and roles with permissions [10]. Because the roles within an organization typically have overlapping permissions, RBAC supports a hierarchical structure of roles to provide permission inheritance, where a senior role can inherit all permissions assigned to junior roles. As a fundamental aspect of RBAC, constraints can also allow us to lay out higher-level access control policies [3].

In this paper, we propose a systematic way to support a well-known access control in OSGi environments. In our proposed approach, the OSGi authorization mechanism is configured and mapped to RBAC, demonstrating that OSGi authorization requirements can be fully fulfilled by RBAC. In addition, important RBAC features are studied to enhance existing access control modules in OSGi service environments.

In the remainder of this paper, we first briefly describe current OSGi authorization mechanism. In Section 3, we formalize OSGi authorization mechanism and demonstrate how OSGi authorization can be accommodated in RBAC. The implementation of our prototype is addressed in Section 4. Several related work are discussed in Section 5. Finally, Section 6 concludes this paper.

2. OSGi Authorization Mechanism

In OSGi service platform, the User Admin service is utilized to find out if the users attempting to access are authorized or not [9]. In the User Admin service authorization architecture, three components are defined:

User - A user is a human being who can be identified by credentials such as a password and other identity attributes.

User Group - A user group is an aggregation of users based on common properties. For example, all family members belong to a user group named *Residents*.

Action Group - Every action that can be performed by a bundle is associated with an action group. For example, if a bundle could be used to control the alarm system, there should be an action group named *AlarmSystemControl*.

In OSGi authorization, the authorization decision is made based on the following two strategies:

ANY Strategy: a user could be allowed to carry out an action if s/he belongs to *Any* member of the action group. For example, the *AlarmSystemControl* action group contains two user groups *Administrators* and *Residents*. *Elmer*, *Pepe* and *Bugs* belong to *Administrators* user group, and

1071-0485/08 $25.00 © 2008 IEEE
DOI 10.1109/FTDCS.2008.43

Table 1. User Groups with Basic User Members

	Elmer	Fudd	Marvin	Pepe	Daffy	Foghorn
Residents	Basic	-	-	Basic	Basic	-
Buddies	-	-	-	-	Basic	Basic
Children	-	-	Basic	Basic	-	-
Adults	Basic	Basic	-	-	-	Basic
Administrators	Basic	-	-	Basic	-	Basic

Table 2. Action Groups with Basic and Required User Group Members

	Residents	Buddies	Children	Adults	Admins
AlarmSystemControl	Basic	-	-	-	Required
InternetAccess	Basic	-	Basic	Basic	-
TemperatureControl	Required	-	-	Required	-
WebCamAccess	Basic	Basic	-	Required	Required
PhotoAlbumView	Basic	Basic	-	-	-

Elmer, *Pepe* and *Daffy* belong to *Residents* user group as follows:

Administrators = { *Elmer, Pepe, Foghorn* }
Residents = { *Elmer, Pepe, Daffy* }
AlarmSystemControl = { *Administrators, Residents* }

This *ANY strategy* allows any of four members, *Elmer*, *Pepe*, *Foghorn* and *Daffy*, to activate the alarm system since all users are a member of user groups and those user groups belong to one action group.

ALL Strategy: a user is allowed to carry out an action if s/he belongs to *All* members of the action group. In the above-mentioned *AlarmSystemControl* example, only *Elmer* and *Pepe* would be authorized to activate the alarm system, since *Daffy* and *Bugs* are not members of both *Administrators* and *Residents* user groups.

The implementation of User Admin service in OSGi service platform supports a combination of both strategies by introducing two member sets, namely the ***basic*** member set for the *ANY strategy* and the ***required*** member set for *ALL strategy*. *Basic* membership defines a set of members that can get access and *required* membership reduces this set by requiring a user to be a required member of each action group.

To accommodate this, OSGi assigns a user to a user group then the user group is assigned to a specific action group. Table 1 and 2 show an authorization example to demonstrate the assignment relationships respectively.

The access decision is made based on the *basic* and *required* assignment relationships. For example, in Table 2 the action group *WebCamAccess* has two *basic* members, *Residents* and *Buddies*, and two *required* members, *Adults* and *Administrators*. Thus, all users belonging to at least one of the *basic* members, *Residents* and *Buddies*, and all *required* members, *Adults* and *Administrators*, are able to carry out the webcam access action. From Table 1, we notice that *Elmer* and *Foghorn* can meet this authorization requirement.

3. Construction of OSGi-compliant RBAC

We found out that the current OSGi authorization mechanism is not intuitive for authorization administration. Furthermore, it is not suitable for satisfying all security requirements for defining fine-grained access control policies in a highly dynamic and open OSGi environment. Our objective is to provide an efficient and effective authorization mechanism for OSGi enabled network environments. To achieve this, we adopt a role-based access control (RBAC) which is a powerful mechanism for reducing the management complexity, administration cost and potential configuration error within the organization [10]. Hence, it is inevitable to identify and derive RBAC components from OSGi authorization requirements.

We first attempt to formally define components in the current OSGi authorization mechanism and these formal definitions are used through the rest of this paper. There are three sets of entities: users (*U*), user groups (*UG*) and action groups (*AG*). The *basic* user-to-user group assignment (*BUA*) is a many-to-many relation between *U* and *UG*. There are two kinds of many-to-many relation between *UG* and *AG*. One is *basic* action group-to-user group assignment (*BAA*) that reflects the *basic memberships* for each action. Another is *required* action group-to-user group assignment (*RAA*) that reflects the *required memberships* for each action. The following definitions formally summarize each component:

- U is a set of users, U=$\{u_1, \ldots u_n\}$,
- UG is a set of user groups, UG=$\{ug_1, \ldots ug_n\}$,
- $AG = AG_{Basic} \cup AG_{Required}$ is a set of *basic* and *required* action groups, AG=$\{ag_1, \ldots ag_n\}$,
- $BUA \subseteq U \times UG$, is a many-to-many *basic* user-to-user group assignment relation,
- $BAA \subseteq AG_{Basic} \times UG$, is a many-to-many *basic* action group-to-user group assignment relation, $BAA[ag_i, ug_i] = \{basic\}$,
- $RAA \subseteq AG_{Required} \times UG$, is a many-to-many *required* action group-to-user group assignment relation, $RAA[ag_j, ug_j] = \{required\}$,
- $users : UG \rightarrow 2^U$, is a function mapping each user group ug_i to a set of users, $users(ug_i)$=$\{u \in U | (u, ug_i) \in BUA\}$,
- $user\_groups : U \rightarrow 2^{UG}$, is a function mapping each user u_i to a set of user groups, $user\_groups(u_i)$=$\{ug \in UG | (u_i, ug) \in BUA\}$,
- $ba\_user\_groups : AG_{Basic} \rightarrow 2^{UG}$, is a function mapping each basic action group ag_i to a set of user groups, $ba\_user\_groups(ag_i)$=$\{ug \in UG | (ag_i, ug) \in BAA\}$,

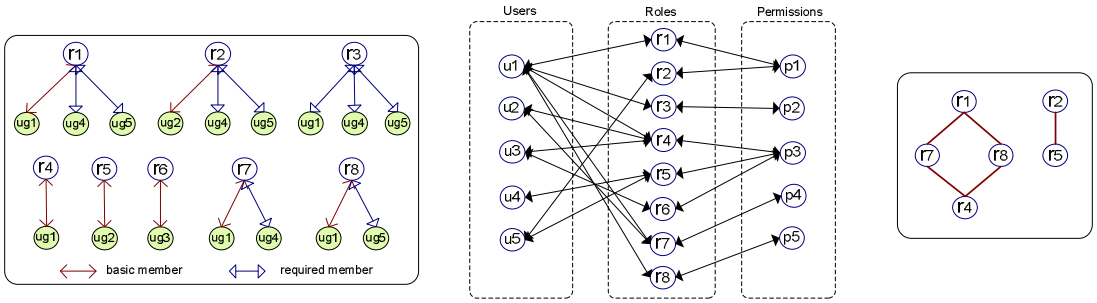
	u_1	u_2	u_3	u_4	u_5
ug_1	B	B	B	-	-
ug_2	-	-	-	B	B
ug_3	-	-	B	-	-
ug_4	B	B	-	-	B
ug_5	B	-	-	-	B

(a) USER-UG assignment

	ug_1	ug_2	ug_3	ug_4	ug_5	
ag_1	B	B	-	R	R	Case 1
ag_2	R	-	-	R	R	Case 3
ag_3	B	B	B	-	-	Case 2
ag_4	B	-	-	R	-	
ag_5	B	-	-	-	R	

(b) AG-UG assignment

Figure 1. An OSGi Authorization Example



(a) Role Construction (b) Mapping to RBAC (c) Role Hierarchies

Figure 2. Overall Process of Constructing RBAC from an OSGi Authorization.

- $re\_user\_groups : AG_{Required} \rightarrow 2^{UG}$, is a function mapping each required action group ag_i to a set of user groups, $re\_user\_groups(ag_i)$=$\{ug \in UG | (ag_i, ug) \in RAA\}$,
- $ba\_action\_groups : UG \rightarrow 2^{AG_{Basic}}$, is a function mapping each user group ug_i to a set of basic action groups, $ba\_action\_groups(ug_i)$=$\{ag \in AG_{Basic} | (ag, ug_i) \in BAA\}$, and
- $re\_action\_groups : UG \rightarrow 2^{AG_{Required}}$, is a function mapping each required user group ug_i to a set of required action groups, $re\_action\_groups(ug_i)$=$\{ag \in AG_{Required} | (ag, ug_i) \in RAA\}$.

Next, we demonstrate how RBAC components and features can be utilized to support components of OSGi authorization. We describe how we can identify basic RBAC component, role hierarchy and constraints to enforce OSGi authorization requirements.

3.1. Constructing basic RBAC components

The essence of RBAC is the notion of *roles* to abstract users and permissions. Permissions are grouped through roles, and users obtain permissions by being assigned to roles. Users, roles, permissions and the corresponding assignment relations are core components in RBAC.

We noticed that the User set (*U*) and the Action Group set (*AG*) essentially can be treated as *users* and *permissions* in the RBAC model, respectively. In addition, the User Group set (*UG*) in OSGi can be represented as *roles* in such that actions are abstracted through User Groups and those actions can be exercised by a member of the specific User Group. However, normally OSGi uses two types of assignments, namely the basic assignment (*BAA*) and the required assignment (*RAA*) that can allow us to achieve the abstraction of actions through User Groups. However, RBAC cannot accommodate this characteristic directly . So, we propose several properties to bridge this gap in the course of OSGi-compliant RBAC construction.

First, we construct the role (*R*) by introducing a concept of *Private Membership*. In particular, we treat the user groups in OGSi as the private members of each role in role construction, which can be further characterized as *basic* members and *required* members depending on the property of the OSGi *BAA* relation and *RAA* relation. The detailed role construction process is explained through an OSGi authorization example as shown in Figure 1. In this example, Table (a) reflects the assignment relation between *U* and *UG*, and Table (b) shows the assignment relation between *AG* and *UG*. In Table (b), there are two types of assignment relations, *Basic* and *Required* to reflect *BAA* and *RAA* relations respectively. For a particular action group, we identify that there exist three possible combinations on *Basic* and *Required* assignments as follows:

Case 1: Contains both *basic* assignments and *required* assignments as shown in the row of

ag_1, $ba\_user\_groups(ag_1) = \{ug_1, ug_2\}$ and $re\_user\_groups(ag_1) = \{ug_4, ug_5\}$.

Case 2: Contains only *basic* assignments as shown in the row of ag_3, $ba\_user\_groups(ag_3) = \{ug_1, ug_2, ug_3\}$.

Case 3: Contains only *required* assignments as shown in the row of ag_2, $re\_user\_groups(ag_2) = \{ug_1, ug_4, ug_5\}$.

As specified in the OGSi authorization strategies, a user must be assigned to *ALL required* user groups and *ANY basic* user groups before s/he can exercise a particular action. Also, while accommodating the identified *basic* and *required* assignment patterns, we define the following mapping rules on action group basis:

Rule 1. Introduce one role and one permission-to-role assignment for each OSGi basic action group-to-user group assignment in BAA. The role contains only one basic private member of the basic user group and ALL required members of the required user groups. The particular action group in BAA is constructed as one permission and the permission is assigned to the role.

Rule 2. In case of no basic assignment, introduce one role that contains ALL required members of the required user groups and one permission-to-role assignment by assigning the permission to the role.

Rule 3. In terms of user-to-role assignment, a user can be assigned to a role when the corresponding user in OSGi authorization is assigned to all the private members of that role.

```
Algorithm [Mapping]
Input:  OSGi Authorization Specification
Output: RBAC Authorization Specification
Method:
(1)  P ← AG;
(2)  U ← USER;
(3)  FOR each p ∈ P DO
(4)     BM = ba_user_groups(p);
(5)     RM = re_user_groups(p);
(6)     IF (BM ≠ Ø) ∧ (RM ≠ Ø) THEN
       /* Case 1: Have both Basic Member and Required Member */
(7)        FOR each bm_i ∈ BM DO
(8)           r = newRole(bm_i, RM);
(9)           IF (r ∉ R) THEN
(10)             R.add(r);
(11)             addPA(p, r);
(12)             FOR each u ∈ U DO
(13)                IF (bm_i ∈ user_groups(u)) ∧
                       (RM ⊂ user_groups (u)) THEN
(14)                   addUA(u, r);
(15)    ELSE
(16)       IF (BM ≠ Ø) ∧ (RM = Ø) THEN
          /* Case 2: Only have Basic Member */
(17)          FOR each br_i ∈ BM DO
(18)             r = newRole(bm_i);
(19)             IF (r ∉ R) THEN
(20)                R.add(r);
(21)                addPA(p, r);
(22)             FOR each u ∈ U DO
(23)                IF br_i ∈ user_groups(u) THEN
(24)                   addUA(u, r);
(25)       ELSE
(26)          IF (BM = Ø) ∧ (RM ≠ Ø) THEN
             /* Case 3: Only have Required Member */
(27)             r = newRole(RM);
(28)             IF (r ∉ R) THEN
(29)                R.add(r);
(30)                addPA(p, r);
(31)                FOR each u ∈ U DO
(32)                   IF RM ⊂ user_groups(u) THEN
(33)                      addUA(u, r);
```

Figure 3. Mapping Algorithm from OSGi Authorization to RBAC Authorization.

Now we discuss how to use these three rules to construct the RBAC for each case we have identified. First, we map all users and action groups in OSGi authorization to users and permissions of RBAC directly. As identified in Case 1, the action group ag_1 has both basic member and required member. Following **Rule 1**, we construct roles such as two roles r_1 and r_2 for the action group ag_1 in our example. r_1 contains a basic member ug_1 and two required members ug_4 and ug_5. r_2 is constructed by a basic member ug_2 and all required members ug_4 and ug_5. Formally, $members(r_1) = \{ug_1, ug_4, ug_5\}$, and $members(r_2) = \{ug_2, ug_4, ug_5\}$. Using the same rule, the permission p_1 corresponding to the action group ag_1 is assigned to roles r_1 and r_2. Following **Rule 3**, a user is assigned to a role only if the user has been assigned to all private members of this role. In the example (Figure 1 (a)), u_1 is assigned to ug_1, ug_4 and ug_5. Since ug_1, ug_4 and ug_5 are private members of r_1, u_1 should be assigned to r_1 in RBAC. For the same reason, u_5 is assigned to r_2. From Figure 2, u_1 and u_5 can hold the same action group ag_1, and u_1 and u_5 own the same permission p_1 via r_1 and r_5, respectively. Hence, the same authorization requirements are fulfilled through the process of configuring RBAC.

In Case 2, the action group has only three basic members ug_1, ug_2 and ug_3. Using **Rule 1**, three roles r_4, r_5 and r_6 are constructed by these three basic members of ag_3 as shown in Figure 2. Formally, $members(r_4) = \{ug_1\}$, $members(r_5) = \{ug_2\}$ and $members(r_6) = \{ug_3\}$. Corresponding assignment relations as a result of **Rule 1** and **3** are shown as well.

In Case 3, the action group has only required members. The second row of the OSGi authorization example (Figure 1 (b)) shows an action group ag_2 has three required members ug_1, ug_4 and ug_5. Using **Rule 2**, one role r_3 is constructed that contains all three required members of ag_2. Formally, $members(r_3) = \{ug_1, ug_4, ug_5\}$. Figure 2 depicts the role construction and assignment relationships.

To realize the basic RBAC construction from OSGi authorization, a mapping algorithm is given in Figure 3, which can be performed to map OSGi authorization to ba-

sic RBAC as we discussed above. Some elements and functions in these algorithms–such as *USER*, *AG*, *user_groups()*, *ba_user_grous()* and *re_user_groups()*–are adopted from our formal representation of OSGi authorization mechanism. Some other elements and functions are based on the formal RBAC model.

3.2. Supporting role hierarchies and constraints

*W*e construct role hierarchy to define an inheritance relationship, reducing the cost of administration. For example, all managers in the same organization may have a certain set of core "management privileges", even though they work in different departments. This commonality can be exploited through a role hierarchy that makes each department manager role to inherit a generic "management" role. Role hierarchy allows the policy designer to write generic access polices once by simplifying the complexity of access control policies.

In our mapping approach, roles are constructed based on *basic* members and *required* members of action groups of OSGi authorization mechanism. Considering the private member sets of every role, we discover inclusion relations between them. In Figure 2(a), r_1 is constructed by a basic private member ug_1 and two required private members ug_4 and ug_5, while r_7 is constructed by a basic private member ug_1 and a required private members ug_4. Formally, the following condition is *true*.

$$(ba\_member(r_7)=ba\_member(r_1)) \wedge (re\_members(r_7) \subset re\_members(r_1))$$

Since the private members of r_1 include all private members of r_7, a user who is assigned to r_1 should be assigned to r_7 as well according to **Rule 3**. Thus a user assigned to r_1 should have all permissions of r_7. In other words, r_1 should possess all permissions of r_7 and r_1 is more powerful than r_7. Therefore, a role hierarchy relation can be built between r_1 and r_7. As a senior role, r_1 inherits all permissions of r_7. Commonly, if any of the following two conditions is *true*, r_i is senior to r_j. With this methodology, two role hierarchies can be identified and shown in Figure 2(c).

1. $(ba\_member(r_j)\neq\phi) \wedge (ba\_member(r_i)\neq \phi)$
   $\wedge (re\_member(r_i)\neq\phi) \wedge (ba\_member(r_j)=ba\_member(r_i))$
   $\wedge (re\_members(r_j) \subset re\_members(r_i))$

2. $(ba\_member(r_j)=\phi) \wedge (re\_member(r_i)\neq\phi)$
   $\wedge (re\_member(r_j)\neq\phi)$
   $\wedge (re\_members(r_j) \subset re\_members(r_i))$

RBAC is a policy oriented approach and allows to specify well-known security principles, such as separation of duty and least privilege. These security principles can be defined as constraints in RBAC. Also we have proved that OSGi authorization requirements can be satisfied by RBAC in the previous discussion. We briefly elaborate how RBAC constraints can help enhance OSGi authorization in this section.

Separation of duty (SOD). SOD is a well-known principle for preventing fraud by identifying conflicting roles. In home network environments, some roles cannot be assigned to the same user, such as *Residents* and *Buddies*, *Adults* and *Children*. These roles can be defined as conflicting roles and the corresponding SOD constraints help avoid having undesired assignments.

Prerequisite constraints. This constraint is based on the concept of prerequisite roles. For example, a user can be assigned to the *Administrators* role only if the user is already a member of the *Residents* role. It ensures that only users who are already assigned to the *Residents* role can be assigned to the *Administrators* role.

4. Implementation Details

In order to prove the feasibility of our approach, we design a prototype system. A security-enhanced OSGi authorization architecture is depicted in Figure 4. This authorization architecture is composed with two distinct parts, policy management and policy enforcement.

4.1. Policy management

OSGi authorization mechanism has been defined in the standard of OSGi service platform specification [9]. However, there is lack of available tools for managing OSGi authorization. In our implementation, a web-based OSGi authorization management tool is designed to support our policy management framework. The tool is composed of two major GUIs: (1) OSGi authorization management GUI communicates with OSGi authorization engine to manage standard OSGi authorization policy [1] and (2) RBAC authorization management GUI is used to manage RBAC authorization through the RBAC authorization engine. Transformation handler is responsible for mapping OSGi authorization to RBAC components, building role hierarchies and reconstructing OSGi authorization by performing corresponding algorithms.

XACML [8] is a standard and general purpose access control policy language defined in XML, which is flexible enough to accommodate most access control policy needs. Since core and hierarchical RBAC implementation can be specified by using XACML [7], the XACML policy generator generates XACML policies from RBAC authorization

[1]The current OSGi authorization mechanism can coexist with the OSGi-compliant RBAC authorization mechanism.

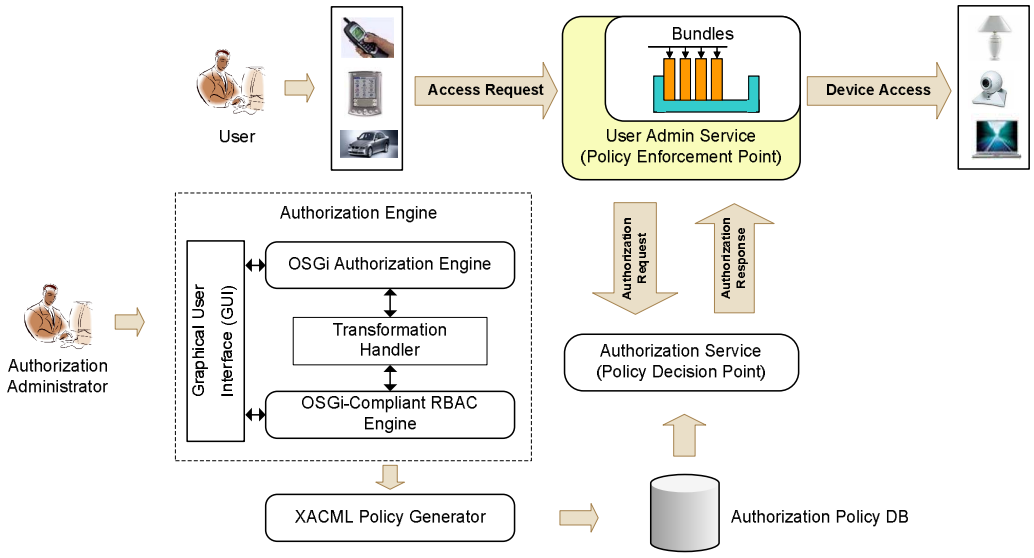


Figure 4. A Security-Enhanced OSGi Authorization Architecture.

specification. Three types of XACML policies are generated: *Role Policy Set*, *Permission Policy Set* and *Role Assignment Policy Set*. All generated policies are stored in an authorization policy DB.

4.2. Policy enforcement

A typical authorization system includes a policy decision point (PDP) and a policy enforcement point (PEP). Existing solutions in authorization management can be divided into two kinds of architectures: pull mode and push mode. In push approach, each subject presents the required information to the PDP and the decision is sent to PEP, while the PEP collects the related information of subjects and queries the PDP for policy decision in pull approach.

Our authorization architecture currently utilizes the pull mode. By integrating the Sun's XACML library [2] into OSGi service platform, the Authorization Service as the PDP module interprets XACML polices and makes access decisions. The User Admin Service as the PEP module queries the Authorization Service and enforces the relevant operations.

5. Related Work

Several related work investigated the access control mechanism for home network environments where the OSGi service platform is operated. Cho et al. [4, 5] proposed an authorization policy management framework based on RBAC for OSGi service platform. Through the comparison of several access control models, they claimed that RBAC model is more flexible than DAC model for home network environments operated by the OSGi service platform. However, they did not consider the established authorization mechanism in current OSGi standard. Also, they omitted to address how important RBAC features, such as role hierarchy and constraints, can be effectively utilized in OSGi service platform. Lim et al. [6] presented a mechanism to bundle authentication and authorization services for the OSGi service platform. Their approach uses XACML to specify RBAC polices for the authorization of service bundles. However, they also ignored existing OSGi authorization mechanism and attempted to employ RBAC in the OSGi platform directly. To the best of our knowledge, our approach illustrated in this paper is the only RBAC authorization solution that is compatible to the existing OSGi authorization standard.

6. Concluding Remarks

In order to prove RBAC can fulfill OSGi authorization requirements and to leverage important RBAC features in the OSGi environment,

We have introduced a systematic mechanism to construct RBAC-compliant OSGi authorization modules. Our approach fulfilled OSGi authorization requirements while leveraging important RBAC features in the OSGi environment. A proof-of-concept prototype has been implemented to demonstrated the feasibility of our approach as well. Currently, we seek a way to validate whether changes in the RBAC-based environment can also be reflected in OSGi authorization environment. In addition, we plan to study

the specification of complicated access control policies, and verification and testing of access control policy specification in OSGi environments.

7. Acknowledgments

This work was supported, in part, by funds provided by National Science Foundation (NSF-IIS-0242393) and Department of Energy Early Career Principal Investigator Award (DE-FG02-03ER25565).

References

[1] *OSGi Initiative*. http://www.osgi.org.

[2] *Sun's XACML implementation*. http://sunxacml.sourceforge.net/.

[3] G.-J. Ahn and R. S. Sandhu. Role-based authorization constraints specification. *ACM Transactions on Information and System Security*, 3(4):207–226, November 2000.

[4] E.-A. Cho, C.-J. Moon, D.-H. Park, and D.-K. Baik. Access control policy management framework based on RBAC in OSGi service platform. In *CIT '06: Proceedings of the Sixth IEEE International Conference on Computer and Information Technology*, page 161, Washington, DC, USA, 2006. IEEE Computer Society.

[5] E.-A. Cho, C.-J. Moon, D.-H. Park, and D.-K. Baik. An effective policy management framework using RBAC model for service platform based on components. In *SERA '06: Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications*, pages 281–288, Washington, DC, USA, 2006. IEEE Computer Society.

[6] H.-Y. Lim, Y.-G. Kim, C.-J. Moon, and D.-K. Baik. Bundle authentication and authorization using XML security in the OSGi service platform. In *ICIS '05: Proceedings of the Fourth Annual ACIS International Conference on Computer and Information Science*, pages 502–507, Washington, DC, USA, 2005. IEEE Computer Society.

[7] OASIS. *Core and hierarchical role based access control (RBAC) profile of XACML v2.0*. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf.

[8] OASIS. *XACML 2.0 Core: eXtensible Access Control Markup Language (XACML) Version 2.0*. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

[9] OSGi. *User Admin Service Specification Version 1.1*. http://www.osgi.org.

[10] R. Sandhu, E.Coyne, H.Feinstein, and C.Youman. Role-based access control model. *IEEE Computer*, 2(29), February 1996.