A Request-Driven Role Mapping for Secure Interoperation in Multi-Domain Environment

Zhuo Tang, Ruixuan Li, Zhengding Lu
College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, Hubei, China
E-mail: hust_tz@126.com, {rxli, zdlu}@hust.edu.cn

Abstract

This paper proposes a request-driven role mapping framework for secure interoperation in multi-domain environment. To support flexible policy expression and inter-domain policy mapping, we propose a more convenient and effective method to complete the privilege query in a general hybrid role hierarchy for all special external requests. To describe the practical relationship between roles, we partition the role-mappings into three types. These mappings describe the forms of the hierarchy between the mapping roles respectively. With our analysis, for the conflicts which arise from the role-mapping among multi-domains, the effective way is to choose a suitable type of role mapping.

1. Introduction

In a typical multi-domain environment [1], we partition the domains into external and local domains. The external is responsible for mediating accesses to individual systems by maintaining a global policy. When an external subject requests the local resources, we assume this process is established on the trust negotiation between the external domain and the local domains. The external subject accesses the local domains through the authentication. However, the existing works on trust negotiation does not address the policy mapping. The current software for the single sign-on (SSO)[2], such as central authorization service (CAS)[3], does not support the smaller granularity access control for the local domains.

The interoperation is established between the domains through role mappings. In this paper, when the local domain received a request, it will search the role hierarchy for the suitable nodes to be mapped for the external. The permission searching is founded on the set uniquely activable set (*UAS*) [4]. We propose an algorithm to compute the *UAS* for a general hybrid hierarchy in local domains. For a given external request, we can work out the most minimal role set for the requested privilege.

The role-mappings describe the relationships between two roles in different domains. These relationships are partitioned into three types: *I-hierarchy*, *A-hierarchy* and *IA-hierarchy* following the GTRBAC [4]. From this partition, we can resolve the conflicts for the role mapping between different domains effectively.

We propose the theories based on that all the security domains employing the RBAC police and the external subjects trying to access some local domains have passed the authentication. Finally, we suppose that the domains trying to interoperation have undergone the trust negotiation.

The contributions of this paper are as follows:

(1) For ascertaining which roles should be mapped in the local domains, this paper proposes a simpler method to compute the *UAS* for a general hybrid hierarchy.

(2) Three types of mappings between roles are proposed. This paper firstly formalizes these three types of role mappings. Through the analysis, selecting the adaptive type of the role mapping can be helpful for solving the conflicts in the interoperations between different domains.

The rest of the paper is organized as follows. Section 2 describes the basic concepts about interoperation policies. Section 3 presents our approach and the algorithms for the privilege query in general hybrid hierarchy. Section 4 describes the establishment for the role mapping, and focus on the resolutions of conflicts in the multi-domain environment. The related work is presented in Section 5, followed by the conclusion in Section 6.

2. The Basic Concepts

0-7695-2943-7/07 $25.00 © 2007 IEEE
DOI 10.1109/NPC.2007.33

This section formally describes the syntax of our policy for the privilege request.

The basic definitions of policy algebra are as follows.

Definition 1. *Authorization Term.*

Authorization terms are 2-tuple of the form: <object, access mode >, which is shortening as <O, A>. It is the basic form of the permission. The set of authorization terms is denoted as *P*. That is *P*= {<O, A>}.

Definition 2. *Permission Set. Permission set* represents all permissions of some subject, which is the set of the authorization terms. We can formulize it as *PS*.

For example, we can describe a role r_1's all permission as: *PS* (r_1) = {<file1, + read>, <file2, -write>}. That is to say the users, which are assigned to r_1, can read file1 and write file2.

The BNF definition for permission set as: $PS= PS| PS\cup PS| PS\cap PS| PS\text{-}PS| SoD (PS, PS)$;

SoD (PS_1(r), PS_2(r)) returns PS_1(r) or PS_2(r), but it can return the PS_1(r) and PS_2(r) concurrently.

In this interoperation framework, the communications between two domains are mainly created by role mappings. The formalized definition about role mappings as follows:

Definition 4 *Role Mapping* .We can formalize the role mapping as a 5-tuple :< r_1, d_1, r_2, d_2, *I/A/IA*>, r_1 is a role in domain d_1, r_2 is a role in domain d_2 respectively, in general, that d_1 is the external domain, and d_2 is the local domain. The fifth parameter is the mapping mode, which denotes the hierarchy relation of the two roles r_1 and r_2.The denotation *I* denotes the *I-hierarchy*, which means that the permissions belongs to the role r_2 is also belongs to the role r_1. The denotation *A* denotes the *A-hierarchy*, which means that if the role r_1 is can be activated by a user, the role r_2 is also can be activated by the user. *IA* denotes the *IA-hierarchy*, which represents that the relationship between the two roles is *IA-hierarchy.* This parameter divides the mapping into two cases, which are called as I-mapping and A-mapping. In following section, we will discuss it more detailed.

As Definition 4 represents, we add a temporal constrain to the role mappings. In this way, we can formalize that a role in a special domain can be enabled or activated by a role in another domain for special duration.

Table 1. The predicates in multi-domain environment

Predicate	Notation	Formalization
I-mapping	$x\rightarrow_I y$	$\forall p, p\subseteq PS(y)\rightarrow p\subseteq PS(x)$
A-mapping	$x\rightarrow_A y$	$\forall u, can\_activate(u,x) \rightarrow can\_activate(u,y)$
IA-mapping	$x\rightarrow_{IA} y$	$(x\rightarrow_{IA}y) \leftrightarrow (x\rightarrow_I y) \wedge (x\rightarrow_A y)$

In Table1, we summarize the predicates of the role mapping in multi-domain environment with temporal constrain. The denotation x, y represent the roles in different domains. The predicate $x\rightarrow_I y$ indicates that the user which assigned with role x can acquire the permissions of the role y.

3. The privilege query in general hybrid hierarchy

When a external subject requests accessing the objects in local domains, we can formalize the request as a temporal sequence: *REQ* (*req*(1), *req*(2),……*req*(n)). In this section, we propose a method for the privilege query in general hybrid hierarchy. Comparing to the existing algorithms such as [7], this method is more convenient and effective.

A special request of an external subject can be denoted as *REQ* = {({<*O*, *A*>})}. The element of the set, *req*(*i*)={<*O*, *A*>}, means that the subject wants the permission set {<*O*, *A*>} in the local domain.

3.1. The Privilege Request in Multi-Domain Environment

Firstly, we consider only one request *req*(*i*). The external request does not conclude the temporal constrains. We assume the *req*(*i*) includes a set of request for the permissions. That is *req*(*i*)= {<*O*, *A*>}.

The following is to discuss how the local domain to create a role to satisfy the request. In general, for a external access request, the requested permission set *req*(*i*), the following situations may arise in a local domain:

i.There is a set of roles in the local domain can be supported for the external domain, possibly hierarchically related, through which the *req*(*i*)can be exactly acquired.

ii.There is a set of roles in the local domain can be supported for the external domain, for which *req*(*i*)is a subset of permissions that can be acquired. That is, the set of roles' permission set include the permissions in the *req*(*i*) as well.

iii.There is a set of roles in the local domain can be supported for the external domain, through which only a sub-set of *req*(*i*) can be acquired, as in the above two cases.

We can create a *1-to-n* mapping from the external entity to the roles set in local domain for a special request. Figure.1 illustrates a simple example for the first case. The real line represents the I-hierarchy while the broken line represents the A-hierarchy between two roles. Assume that *req*(*i*)= $\{p_1\}$. In the local domain, p_1 equals to $PS(r_1)$ exactly. It is obvious that this request can be satisfied if the external entity is mapped to the role r_1. It is the first scene we discuss above. But if *req*(*i*) equals to $\{p_1, p_2, p_3\}$, and in the local domain, $p_1=PS(r_1)$, $p_2=PS(r_2)$, $p_3=PS(r_3)$ exactly, we have to implement a one to more mapping. That is to say, the roles $\{r_1, r_2, r_3\}$ can be mapped by the same external entity. If the senior role r_4 is not assigned with any other permission, it is the I-senior role for the roles r_1 and r_2. For this case, because the permissions of r_4 are $\{p_1, p_2\}$ and the role r_5 has any other permission to be assigned, the return roles are $\{r_4, r_3\}$. Extremely, if the role r_5 has no other permissions to be assigned, the return role is $\{r_5\}$.

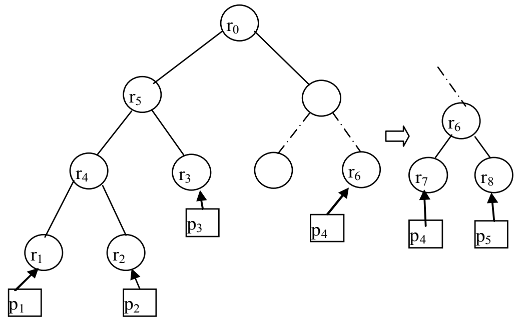

Figure. 1. a simple example for the first case

The second case is to indicate that the *req*(*i*) is a subset of permissions that can be acquired by the external domain, but we can not find a set of role whose permission is equal with the *req*(*i*) exactly. In Figure. 1, if *req*(*i*)=$\{p_1, p_4\}$, because $PS(r_6)=\{p_4, p_5\}$, we can not return the redundant permissions to the external entity. The resolution is to split the role r_6. Through creating two new junior roles r_7, r_8 for r_6, r_7 is assigned with permission p_4, while r_8 is assigned with permission p_5. So the new role set $\{r_1, r_7\}$ can satisfy the request *req*(*i*).

The third case is more complex than the above two. And only part of the requested permissions can be acquired. For the request by the *req*(*i*), the method is to return the roles which contain the requested permissions. These roles can be mapped by the external entity.

3.2. Request-driven Privilege Query Algorithm

In an arbitrary hybrid hierarchy, maintaining permission acquisition and role activation semantics can become quite challenging. Joshi *et al.* introduce the concept of uniquely activable set (*UAS*) to facilitate the analysis of hybrid hierarchies and simplify the process of determining the activation and permission acquisition sets [5]. A *UAS* is a set each element of which is a set of roles that can be activated by a specific user.

The *UAS* can be defined as an extension of the definition in [6]:

Definition 5. *Let H = (R, F) be a rooted hybrid hierarchy. Then, UAS(H) =* $\{Y_1, Y_2, ..., Y_m\}$, *where* $\emptyset \subset Y_i \subseteq R$ *for each* $i \in \{1, 2, ..., m\}$, *is the Uniquely Activable Set (UAS) of role sets of H if the following conditions hold:*

• $\forall\, i, j \in \{1, 2, ..., m\}$ *and* $i \neq j$, $PS(Y_i) \neq PS(Y_j)$, *and*

• $\forall\, Z \subseteq R$ *s.t.* $Z \notin UAS(H)$, *if* $PS(Y) = PS(Z)$ *for a* $Y \in UAS(H)$, *then* $(|Y| < |Z|)$; *where* $|A|$ *is the cardinality of set A.*

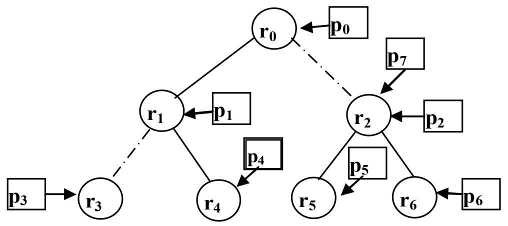

Figure. 2. An example for algorithm of *FindMappedRole*

Following the above principles, we present an algorithm *ComputeUAS* that ascertain a role set that satisfies a requested permission set *req*(*i*). In this algorithm, the breadth first searching in the hierarchy of the local roles ensures that the order of the set of local roles, which is denoted as *R*, is from up to down. Literature [7] proposes an algorithm to find a role set to satisfy the external request. But it can not ensure that its role set can follow this principle, and it did not consider that how the external map to these roles.

The computation of the *UAS* has been proofed as *NP*-complete [8].We proposes a simpler algorithm to create the role set in *UAS*. For an arbitrary general role hierarchy *H*, we can split all *I-hierarchy* (*IH*) from the hierarchy *H*. Each *I-hierarchy* contains part of roles of the hierarchy *H*. Based on these sub-hierarchy, we can compute the *UAS* for the general hierarchy *H* as algorithm *ComputeUAS*.

Firstly, we introduce some operations:

$N_1 \otimes N_2=\{\{x_1 \bigcup x_2\}|x_1 \in N_1 \text{ and } x_2 \in N_2\}$

Analogously,

$N_1 \otimes N_2 \otimes \ldots\ldots \otimes N_m= \{\{x_1 \bigcup x_2 \bigcup \ldots\ldots \bigcup x_2\} \; |x_1 \in N_1,\ x_2 \in N_2 \ldots\ldots x_m \in N_m\}$

Assuming $M= \{N_1, N_{2\ldots} N_m\}$, we have:

$\Theta\, M= N_1 \otimes N_2 \otimes \ldots\ldots \otimes N_m$

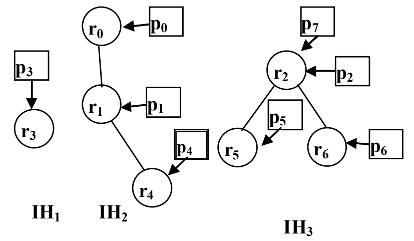

Figure. 3. The decomposition of the hybrid role hierarchy

We propose the algorithm for computing the *UAS* for an arbitrary hybrid role hierarchy *H* as Figure. 4. In this process, we firstly decompose the hybrid role hierarchy into several pure *I-hierarchies*. In the Example 1, we can split the hybrid hierarchy as Figure. 2 to three *I-hierarchies*: $\{IH_1, IH_2, IH_3\}$ as Figure. 3. Comparing to the algorithm proposed in [7], we have a simpler approach to constitute the set of *UAS*. Through producing a set of roles *R'* by selecting arbitrary roles where there is no I-hierarchy relation between them in every hierarchy IH_i, the power set of the *R'* is added to the *UAS* of the hierarchy *H*. For instance, we select $\{r_0, r_2, r_3\}$ as *R'*, then the set $\{\{r_0\},\{r_2\},\{r_3\},\{r_0,r_2\},\{r_0,r_3\},\{r_2,r_3\},\{r_0,r_2,r_3\}\}$ is must added to the *UAS*. Once we complete selecting all roles in each hierarchy IH_i, the *UAS* is formed.

Theorem 1. The set of the role set created by Algorithm *ComputeUAS* is the *UAS* for the hybrid role hierarchy.

Proof. The condition of the algorithm is that there is a general hybrid hierarchy *H*, a user *u* is assigned to the senior-most role *SH* of *H*. Without loss of generality, we assume that the permissions belong to the different role in the same *I-hierarchy* is unique. So, when we producing a set of roles set *R'* by selecting arbitrary role in every *I-hierarchy* IH_i, the permissions of all subset of the power set of the *R'* is different. It satisfies the first condition in the *Definition 5*.

Because the element in the *R'* is the role set that each role in it comes from different *I-hierarchy* (it is denoted *Yi* in the *definition 5*), the relation for "contain" does not exist among permission set of each role in a *Yi*. Assuming $Yi=\{r_i,r_j\}$, if $PS(Yi)= PS(Y_j)$,and $|Yi|>|Y_j|$, let us assume Y_j to be $\{r_j\}$,then $PS(r_i) \subseteq PS(r_j)$. In general case, if two roles belong to different *I-hierarchies* respectively or if the two roles are not the subordinate relationship in the same *I-hierarchies* (such as r_5 and r_6), ones permissions can not contain the other. So, it satisfies the second condition in the *Definition 5*. Hence, the set of the roles set produced by Algorithm 1 is the *UAS* for the hybrid role hierarchy. □

Example 1. Consider the hybrid hierarchy shown in Figure. 2. If we visit the tree following the breadth first, then $R= \{r_0, r_1, r_2, r_3, r_4, r_5, r_6\}$. The *UAS* for a user assigned to r_0 is:

$\{\{r_0\}, \{r_1\}, \{r_2\}, \{r_3\}, \{r_4\}, \{r_5\}, \{r_6\}, \{r_0, r_2\}, \{r_0, r_5\}, \{r_0, r_6\}, \{r_1, r_2\}, \{r_1, r_5\}, \{r_1, r_6\}, \{r_2, r_4\}, \{r_4, r_5\}, \{r_4, r_6\},\{r_0, r_3\}, \{r_1, r_3\}, \{r_3, r_4\},\{r_2, r_3\}, \{r_3, r_5\}, \{r_3, r_6\}, \{r_0, r_2, r_3\}, \{r_0, r_3, r_5\}, \{r_0, r_3,r_6\}, \{r_1, r_2, r_3\}, \{r_1, r_3,r_5\}, \{r_1, r_3, r_6\}, \{r_2, r_3, r_4\}, \{r_3,r_4, r_5\}, \{r_3,r_4, r_6\},\{r_0,r_5, r_6\},\{r_1,r_5, r_6\},\{r_4,r_5, r_6\},\{r_3,r_5, r_6\}\}$

The permission set of each role set in the *UAS* are: $PS(\{r_0\})=\{ p_0,p_1,p_4\}$, $PS(\{r_1\})=\{ p_1,p_4\}$, $PS(\{r_2\})=\{ p_2,p_5,p_6,p_7\}$, $PS(\{r_3\})=\{ p_3\}$, $PS(\{r_4\})=\{ p_4\}$, $PS(\{r_5\})=\{ p_5\}$, $PS(\{r_6\})=\{ p_6\}$, $PS(\{r_0, r_2\})=\{ p_0,p_1, p_2,p_4,p_5,p_6,p_7\}$, $PS(\{r_0, r_5\})=\{ p_0,p_1,p_4, p_5\}$, $PS(\{r_0, r_6\})=\{ p_0,p_1,p_4, p_6\}$, $PS(\{r_1, r_2\})=\{ p_1, p_2, p_4, p_5,p_6,p_7\}$, $PS(\{r_1, r_5\})=\{ p_1,p_4, p_5\}$, $PS(\{r_1, r_6\})=\{ p_1,p_4, p_6\}$, $PS(\{r_2, r_4\})=\{ p_2, p_4,p_5,p_6,p_7\}$, $PS(\{r_4, r_5\})=\{ p_4, p_5\}$, $PS(\{r_4, r_6\})=\{ p_4, p_6\}$, $PS(\{r_0, r_3\})=\{ p_0,p_1, p_3,p_4\}$, $PS(\{r_1, r_3\})=\{ p_1, p_3,p_4\}$, $PS(\{r_3, r_4\})=\{ p_3,p_4\}$, $PS(\{r_2, r_3\})=\{ p_2, p_3,p_5,p_6,p_7\}$, $PS(\{r_3, r_5\})=\{ p_3,p_5\}$, $PS(\{r_3, r_6\})=\{ p_3,p_6\}$, $PS(\{r_0, r_2, r_3\})=\{ p_0,p_1, p_2, p_3, p_4,p_5,p_6,p_7\}$, $PS(\{r_0, r_3, r_5\})=\{ p_0,p_1, p_3, p_4,p_5\}$, $PS(\{r_0, r_3,r_6\})=\{ p_0,p_1, p_3, p_4,p_6\}$, $PS(\{r_1, r_2, r_3\})=\{ p_1, p_2, p_3, p_4,p_5,p_6,p_7\}$, $PS(\{r_1, r_3, r_5\})=\{ p_1, p_3,p_4, p_5\}$, $PS(\{r_1, r_3, r_6\})=\{ p_1, p_3,p_4, p_6\}$, $PS(\{r_2, r_3, r_4\})=\{ p_2, p_3, p_4,p_5,p_6,p_7\}$, $PS(\{r_3,r_4, r_5\})=\{ p_3,p_4, p_5\}$, $PS(\{r_3,r_4, r_5\})=\{ p_3,p_4, p_6\}$, $PS(\{r_0,r_5, r_6\})=\{ p_0, p_1,p_4,p_5, p_6\}$, $PS(\{r_1,r_5, r_6\})=\{ p_1,p_4,p_5, p_6\}$, $PS(\{r_4,r_5, r_6\})=\{p_4,p_5, p_6\}$, $PS(\{r_3,r_5, r_6\})=\{p_3,p_5, p_6\}$.

The *UAS* supports the first case of permission request above commendably. If $req(i)= \{ p_1,p_4, p_6\}$, by searching through the permission set of each role set in the *UAS*, we can find the minimal role set for supporting these permissions: $\{r_1, r_6\}$.If we can not find a set of local roles whose permissions are equal with the $req(i)$ exactly, splitting the local role or creating the new roles is a valid method.

```
Algorithm1 ComputeUAS(H)
    Input: H—an arbitrary hybrid role hierarchy
    Output: UAS—the UAS of the role hierarchy H
    1.  initialize $UAS = \varnothing$
    2.  $IH \longleftarrow$ I-hierachyGeneration(H).
        //$IH=\{IH_1, IH_2, ...... IH_m\}$,algorithm 2
    3.  $IH' \longleftarrow$ No-hierachySetGeneration(IH)// algorithm 3
    4.  Foreach $IH'' \in 2^{IH'}$ Do
    5.      If ($|IH''|==1$) then
    6.          foreach $r' \in IH''$ Do
    7.              $UAS = UAS \bigcup \{ r' \}$
    8.      else
    9.  $UAS = UAS \bigcup \Theta IH''$
    10. Return UAS.
Algorithm2 I-hierachyGeneration(H)
    Input: H—an arbitrary hybrid role hierarchy
    Output: IH—all I-hierarchies from the hierarchy H
    1.  initialize $TempR=\varnothing$//the list for the role that has been searched
    2.  initialize $IH = \varnothing$
    3.  R=Roles(H)// R is the list of all roles in the hierarchy H
    4.  While $R \neq \varnothing$ Do
    5.      From r start DFS Search under I-hierachy, the roles passed are added
            to the set TempR
    6.      when the DFS Search completed, produce a sub- I-hierachy: $IH_i$
    7.      $IH= IH \bigcup IH_i$
    8.      $R= R—TempR$
    9.  End while
    10. Return IH.
Algorithm3 No- InheritSetGeneration(IH)
    Input: IH—an arbitrary hybrid role hierarchy
    Output: IH'—all roles-set with out inherit hierarchy from the I-hierarchy IH
    1.  initialize IH' as the power set of the roles in the IH
    2.  Foreach $R' \in IH'$ Do
    3.  if there is a inherit path between the arbitrary roles in the set R' then
    4.      $IH'= IH'—R'$
    5.  Return IH'.
```

Figure.4. The algorithms of creating the *UAS* for a hybrid role hierarchy

For the second case of the external request, we can not find an exact role set in *UAS* to satisfy the permissions requested. For the Example 1, assuming the request $req(i)=\{ p_1,p_4, p_7\}$, there are no local roles containing the permission p_7 merely. The method in this paper is to find an appropriate role to be decomposed. The result of the decomposition is to produce a new role to contain the adaptive permissions for the external request. For the instance as in the Figure.5, the roles set in the *UAS* which permissions satisfy the request are: $\{r_0, r_2\}$, $\{r_1, r_2\}$, $\{r_0, r_2, r_3\}$, $\{r_1, r_2, r_3\}$.Because the set $\{r_1, r_2\}$ owns the least permissions, and $PS(r_1)=\{p_1,p_4\} \subseteq req(i)$, $PS(\{r_2\})=\{p_2,p_5,p_6,p_7\} \not\subset req(i)$, the role r_2 is the suited role to be decomposed. For the stabilization of the *UAS*, we prefer to make the selected role as the direct senior role of the new role in each adjustment of the local hierarchy. In this instance, we can create a new role *r'*, $PS(r')=\{p_7\}$. For the request is to active some local roles to acquire the appropriate permissions, the relationship between roles r_2 and *r'* is *I-hierarchy*.

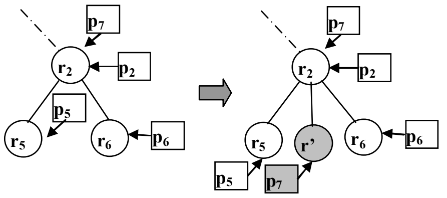

Figure. 5. A simple example for the second case

The resolution for the third case is similar with the second. we can split *req*(*i*) into two subsets *req1*(*i*)and *req2*(*i*)such that *req1*(*i*)relates to the second case and *req2*(*i*) refers to the permission set that are not available through any existing role.

4. Request-driven role mapping in multi-domain environment

When the privilege query completed in the local domain for a special external request, the follows is to create the role mappings between the two domains. This section is to discuss which type of mappings is suitable for a special external request. When the requisite roles in the local domain are confirmed for a special request, the mapping from the roles of the subject for the request to these roles must be found.

4.1. The problem for the role mapping

In the Figure.6, the hierarchy of the local domain is the same as the Example 1. The *req* of the external user *u* is $\{p_1, p_4, p_6\}$. Through searching the *UAS* of the local domain, we can know that the roles $\{r_1, r_6\}$ will be mapped by the external domain. In the external domain, the role r_c which has two *A-junior* roles is assigned to the user u_1. We have to select a role from r_c or its juniors to form a mapping with the local roles $\{r_1, r_6\}$.

When the roles in the local domain are confirmed, the problems are as follows:

1) Which roles that assigned to the request subject in the external domain must be the original of the mapping to the local roles (We call them the original roles.)?

2) Which types of mapping is to be found between the external and the local domain. That is to say, which hierarchy suits for the two roles between the external and the local domain?

As we know, secure interoperation requires enforcement of the following two principles: autonomy principle and security principle [9]. In this paper, the selection of the types of mappings and the original roles is up to these principles. The following sections are to discuss how we select the original roles and the mapping types to avoid the conflicts in role mappings.

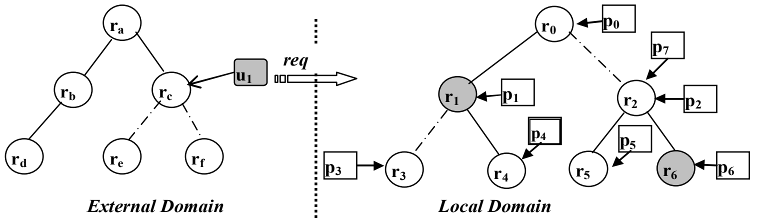

Figure.6. An example of the role mapping between two domains

4.2. Conflicts of role mapping for privilege request

The role mapping in different domains is a cause of the various types of conflicts and inconsistencies. There are various conflicts existing in role mapping among multiple domains, such as modality conflicts, cyclic inheritance, separation of duties (*SoD*)[10] and multiple management[9].Due to the limitation of the paper's space, we focus on the violation of *SoD* for the external requester.

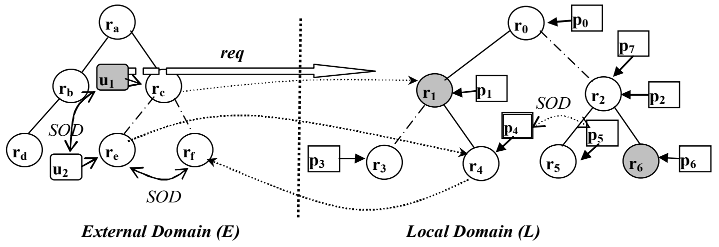

Figure. 7. An example of *SoD* in two domains.

SoD [10] prevents two or more subjects (roles or users) from accessing an object that lies within their conflict of interests or disallow a subject from accessing conflicting objects or permissions. Violations of *SoD* constraints may occur in an interoperation policy because of the interplay of various policy constraints across domains.

1) The role specific violation of *SoD*

For example, Figure. 7 shows that the roles r_e and r_f are the conflicting roles in the external domain. As the role r_c is assigned to the user u_1, the user u_1 can activate the roles r_e and r_f, but cannot active them concurrently. We assume that user u_1 active the roles r_c and r_e concurrently, that is to say, the user $u_1$can inherit all permissions of the role r_e: $PS\ (u_1) \supseteq PS\ (r_c)$, $PS\ (u_1) \supseteq PS\ (r_e)$. As mention above, if the user u_1 request the permission $\{p_1, p_4, p_6\}$, the mapping must found between the role of $\{r_c, r_e, r_f\}$ to the local roles $\{r_1, r_6\}$.Now, let us consider the *I-mapping* $< r_c, E, r_1, L, I>$. The role r_c can inherit all permissions of the role r_1 in domain L through this mapping: $PS\ (u_1) \supseteq PS\ (r_1) \supseteq PS\ (r_4)$. If there is already an *I-mapping* exist from role r_4 to role r_f, the role r_c can inherit all permissions of the role r_f through r_4. $PS\ (u_1) \supseteq PS\ (r_c) \supseteq PS\ (r_4) \supseteq PS\ (r_f)$. So, $PS\ (u_1) \supseteq (PS\ (r_e) \cup PS\ (r_f))$. Thus the user u_1 can acquire the permissions of the role r_f, which is the *SoD* with the role r_e.

Supposing denotations U and R represent the set of users and roles in a domain respectively, the violation of role-specific *SoD* can be formalized as: $\{\exists u \in U, r_1, r_2 \in R, SoD(r_1,r_2) | PS(u) \supseteq PS(r_1) \cup PS(r_2)\}$

In this case, if the user u_1 request the permission $\{p_1, p_4, p_6\}$, and the user u_1 active the roles r_c and r_e concurrently, we can select the role r_e to mapping to the local role r_1. Through the mapping $< r_e, E, r_1, L, I >$, the user u_1 can acquire the permissions $\{p_1,p_4\}$ without the violation of *SoD*. Of course, the mapping also can be an *A-mapping*: $< r_e, E, r_1, L, A/IA>$

2) The violation of user-specific *SoD*

For example as Figure.7 shows, the user u_1 and u_2 is a pair of *SoD* users. The role sets, which assigned to them, are prohibited from having a common set. As mentioned above, if the user u_1 requests the permission $\{p_1, p_4, p_6\}$, the mapping must be founded between the role of $\{r_c, r_e, r_f\}$ and the local roles $\{r_1, r_6\}$. Before the mapping establishment, we assume that there is a mapping $< r_e, E, r_4, L,$ I> between the two domains. Now, let us consider the *I-mapping* $< r_c, E, r_1, L,$ I>.Through this mapping, the user u_1, which is assigned the role r_c, can acquire all of the permissions of r_1: $PS(u_1) \supseteq PS(r_c) \supseteq PS(r_1)$. As r_1 is the senior role of r_4, so the permission set of role r_4 is the subset of the one of r_1. That is to say, the user u_1 own a role B6, $PS(u_1) \supseteq PS(r_4)$. On the other hand, there is a mapping from role r_e to role B6. As the role A6 is assigned to the role u_2, the user u_2 can also access the role r_4. So the user u_2 can acquire the permissions of the role r_4 in the same way: $PS(u_2) \supseteq PS(r_4)$. Thus, the conflicting users, u_1 and u_2, can access the same role r_4. This is a representative case of the violation of the *SoD* for user-specific.

Supposing denotations U and R represent the set of the users and roles in a domain respectively, the violation of the *SoD* for the user-specific can be formalized as: $\{\exists u_1, u2 \in U, \exists r_1, r_2 \in R \mid SoD(u_1,u_2) \wedge r_1 \in role(u_1) \wedge r_2 \in role(u_2) \wedge Containned(r, r_1) \wedge Containned(r, r_2)\}$

In this case, if the user u_1 request the permission $\{p_1, p_4, p_6\}$, when the user u_1 and u_2 are a pair of *SoD* users in the external domain, as considering the condition that there is a mapping $< r_e, E, r_4, L,$ I> exist, we can select an A-mapping $< r_c, E, r_1, L,$ A $>$. Through this mapping, the user u_1 can active the local role r_1 to acquire the permissions $\{p_1,p_4\}$. But the user u_1 can not activate r_1 and r_e at the same time. That is to say, the roles r_1 and r_e are the roles of *SoD*.

3) The violation of *SoD* for policy assignment

The *SoD* for policy assignment means that the conflicting policies are prohibited from being authorized to the same role. Generally speaking, the violation for this type of *SoD* can not be caused directly by role-mappings. For instance, as the Figure.7 shows, in the local domain, p_4 and p_5 are the conflicting policies. If the external user will request the two privileges concurrently, this request can not be satisfied completely from the *I-mapping*. This type of violation of the *SoD* usually arises from the I-hierarchies and the role mapping between two domains. We can formalize it as: $\{\exists p_1,p_2 \in P, \exists r \in R, p_1 = < r_1,PS,d >, p_2 = <r_2,PS,d> \mid SoD(p_1,p_2) \wedge Containned(r_1,r) \wedge Containned(r_2, r)\}$

We can resolve this problem through *A-mapping*. If user u_1 request the conflicting permissions $\{p_4, p_5\}$, through searching in the *UAS* of the local domain, we can confirm the roles r_4 and r_5 are the ones to be mapped. For the users in the external domain can not own the permissions p_4 and p_5 concurrently, we can create the *A-mapping* $< r_e, E, r_4, L,$ A> and $< r_e, E, r_5, L,$ A>. Through these two mappings, the user u_1 can active the local role r_4 or r_5, but can not active them concurrently.

When the conflicts occur in the interoperations, we can choose the suitable types of mappings and original roles to reduce or avoid the conflicts. The principles for the selections are as follows.

i. When the violation of role-specific *SoD* occurs, the appropriate way is to change the original role of this mapping. The lowest junior roles of the external requester are the best choices for this change.

ii. When the violation of user-specific *SoD* occurs, the appropriate way is to establish the *A-mapping* between the two domains, and the conflicting users are prohibited from activating the same junior roles in the A-hierarchy path concurrently.

iii. When the violation of policy assignment *SoD* occurs, the best way is to establish the A-mappings from external to the requested roles.

5. Related works

Time-based secure interoperation has not been addressed by earlier models. As an important extension of the model GTRBAC, James B.D. [11] presents design and implementation of X-GTRBAC Admin, an administration model that aims at enabling administration of role-based access control (RBAC) policies in the presence of constraints with support for conflict resolution in a multi-domain environment.

Several research efforts [12, 13, 14] have been devoted to the topic of policy composition and secure interoperation in multi-domain environment. In [9], an integer programming approach has been proposed to allow policy integration between multiple RBAC policies. More relevantly, [7] has tried different approaches to facilitate the administration of role hierarchy by constructing the actual *UAS* set. While the first approach is slightly better in terms of time complexity, both these approaches are non-polynomial solutions. Although our method is also non-polynomial solution (practically, finding an *UAS* for a general hybrid hierarchy is proofed as a *NP-complete*[8]), but the algorithm proposed in this paper is simpler.

In [15], Shehab *et al.* proposed a distributed secure interoperability protocol that ensures secure interoperation of the multiple collaborating domains without compromising the security of collaborating domains. In [16], the authors propose a breadth-first-search based algorithm for policy mapping between two loosely coupled interacting domains for sharing resources. However, the algorithm proposed does not do an exhausted search; instead, it creates new roles even if there is possible a combination of roles in the local domain that can satisfy the requested permissions. Other earlier work related to hybrid hierarchy that highlight its importance can be found in [17].

There are several researches [10] concerning the resolution of the conflicts between role-mapping. Through our analyses and comparison, the selection for adaptive types of role-mapping for a special external request is a most effective method to avoid the conflicts.

6. Conclusion and future work

In this paper, we propose a request-driven temporal policy framework for interoperation in a multi-domain environment. While a hybrid hierarchy is important to make an RBAC approach generic enough to capture very diverse set of access request as well as to support flexible policy expression and inter-domain policy mapping, we propose a more convenient and effective method to complete the permission query in a local domain. Base this method, we can return the suitable role set for all special external privilege requests. The role-mappings between the external and local domains are found base this role set.

The participant roles in the mappings may have three types of hierarchies. That is to say, when a role is mapped by another, the one relationship of *I-hierarchies*, *A-hierarchies* and *IA- hierarchies* may be established between the two roles. For these hierarchies, this paper proposes three catalogs of role-mappings respectively. When the conflicts occur in the role-mapping, we can choose a suitable type of mapping to avoid the conflicts.

7. References

[1] R. Power, Tangled Web: Tales of Digital Crime from the Shadows of Cyberspace. Que/Macmillan Publishing, Aug. 2000.

[2] Gang Zhao, Dong Zheng, Kefei Chen, *Design of single sign-on*, E-Commerce Technology for Dynamic E-Business. IEEE International Conference on 2004, 253-256

[3] *ITS Central Authentication Service*, [EB/OL] http://www.yale.edu/tp/cas/

[4] Joshi, J.B.D., Bertino, E., Ghafoor, A. *Temporal hierarchies and inheritance semantics for GTRBAC*, Proceedings of the seventh ACM symposium on Access control models and technologies, Monterey, California, USA, 2002, pp:74 - 83.

[5] Joshi, J.B.D., Bertino, E., Ghafoor, A. *Temporal hierarchies and inheritance semantics for GTRBAC*, Proceedings of the seventh ACM symposium on Access control models and technologies, Monterey, California, USA, 2002, pp:74 – 83.

[6] Joshi, J.B.D., Bertino, E., Ghafoor, A. *Formal Foundation for Hybrid Hierarchies in GTRBAC*. ACM Transactions on Information and System Security. (revised submission).

[7] Chandran, S.M., Joshi, J.B.D. *Towards Administration of a Hybrid Role Hierarchy*, IEEE International Conference on Information Reuse and Integration, 2005.

[8] Siqing Du, Joshi, J.B.D. *Supporting Authorization Query and Inter-domain Role Mapping in Presence of Hybrid Role Hierarchy*, SACMAT'06, June 7-9, 2006, Lake Tahoe, California, USA.

[9] Basit Shafiq, James B.D. Joshi, Elisa Bertino, *Secure Interoperation in a Multidomain Environment Employing RBAC Policies*, IEEE Transactions on Knowledge and Data Engineering, vol.17,no.11,pp:1557-1577, 2005.

[10] Emil C. Lupu, Morris Sloman. *Conflicts in Policy-Based Distributed Systems Management*, IEEE Transactions on Software Engineering, vol. 25, no. 6, pp. 852-869, Nov. 1999.

[11] James B.D. Joshi. et al. X-GTRBAC Admin: A Decentralized Administration Model for Enterprise-Wide Access Control. ACM Transactions on Information and System Security, vol. 8, no.4, pp. 388-423, Nov 2005

[12] Bonatti, P.A., Sapino, M. L., Subrahmanian, *Merging Heterogeneous Security Orderings.Esorics* 1996,183-197.

[13] Dawson, S., Qian, S., Samarati, P. *Providing Security and Interoperation of Heterogeneous Systems*. International Journal of Distributed and Parallel Databases. 2000.

[14] Gong, L., Qian, X. *Computational Issues in Secure Interoperation*. IEEE Transaction on Software and Engineering, Vol. 22, No. 1, January 1996.

[15] Shehab, M., Bertino, E., Ghafoor, A. *SERAT: Secure Role mapping Technique for Decentralized Secure Interoperability*, In Proceedings of the ACM Symposium on Access Control, Models and Technologies (SACMAT 05), Stockholm, Sweden, 2005.

[16] Piromruen, S., Joshi, J.B.D. *An RBAC Framework for Time Constrained Secure Interoperation in Multi-domain Environment*, IEEE Workshop on Object-oriented Real-time Databases (WORDS-2005), 2005.

[17] Joshi, J.B.D., Bertino, E., Ghafoor, A. *Temporal hierarchies and inheritance semantics for GTRBAC*, Proceedings of the seventh ACM symposium on Access control models and technologies, Monterey, California, USA, 2002, pp:74 -83.