http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 8 (5): 726-734, 2009
ISSN 1812-5638

Security Policy Management for Systems Employing Role Based Access Control Model

Chao Huang, Jianling Sun, Xinyu Wang and Yuanjie Si
College of Computer Science, Zhejiang University, Zheda Road 38, Hangzhou, Zhejiang, 310027, China

Abstract: In this study, we propose the redundancy and inconsistency checking algorithms to support the policy management of systems employing role based access control model. Present method is based on the formal definition of the policy redundancy and policy inconsistency. Via constructing the role graph, we analyze the redundancy and inconsistency one by one. According to the features of each type of redundancy and inconsistency, present algorithm checks all the possible violations and generates the related policy elements to help the security administrator to amend the policy afterwards. The performance test demonstrates that the approach is efficient enough for practical usage. Present approach could guarantee the conciseness as well as consistency of the access control policy, at same time reduce the burden of access control administration significantly.

Key words: Policy inconsistency, policy redundancy, separation of duty, cardinality constraint, role hierarchy, policy checking

INTRODUCTION

The prominent progress of Internet and related technologies has promoted tremendous information and data sharing. There is a growing concern for security and privacy of data and numerous studies have shown that unauthorized access can cause great losses, especially for the financial software (Ferraiolo *et al*., 2003; Schaad *et al*., 2001). Access control becomes more and more essential for safe and secure access to the software and hardware resources. Role Base Access Control (RBAC) model, which was proposed by Ferriaiolo and Sandhu (2001) at the 90s of last century, provides a powerful way to satisfy the access control needs (Li and Tripunitara, 2006). RBAC96 is currently the most widely used access control model in enterprise area since its fine grained control over the privilege (Ferraiolo *et al*., 2001; Li and Mao, 2007). Furthermore, RBAC is capable of modeling a wide range of access control policies including Discretional Access Control (DAC) and Mandatory Access Control (MAC). An access control policy is a statement which specifies the rules about how to setup the process for grating or denying authorizations to the users (Coyne, 1996; Beznosov and Deng, 1999). The access control redundancy means there are unwanted specifications about the same access rule and the access control inconsistency means there are access control rules conflicting with each other in the policy, for example rule A claims that granting role R to user Bob, however rule B claims that denying the request of assigning R to Bob, when the above access control rules coexist in one policy, the access control policy inconsistency occurs, which may conceal the security danger or even breakdown the entire access control system (Schaad *et al*., 2001; Nyanchama and Osborn, 1999). Thus, it is extremely important to make sure that there is no inconsistency of an access control policy. For the access control system employing RBAC model, it is much more complicated for the policy inconsistency management, since RBAC supports more advanced constraints.

Centonze *et al*. (2006) present a theoretical foundation for correlating an operation-based RBAC policy with a data-based RBAC policy via., a location consistency property. The research shows how to infer whether an operation-based RBAC policy is equivalent to any database RBAC policy and a static analysis tool called SAVES is introduced to facilitate the consistency validation. However, the inconsistencies concerning role hierarchies and cardinality constraints are not discussed. Chang and Hoh (2005) discussed the inconsistency detection in distributed component environment. Via analyzing the server-centric RBAC system, they proposed an inconsistency detection method between two access control policies and also investigated the conditions of guaranteeing consistency. However, there is no the discussion of the inconsistency in each individual distributed system, which is actually the pre-condition of the consistency in the entire distributed system. Strembeck (2004) presents the experiences with the implementation of conflict-checking methods for

Corresponding Author: Chao Huang, College of Computer Science, Zhejiang University, Zheda Road 38, Hangzhou, Zhejiang, 310027, China

separation of duty constraints in the XORBAC access control service and only provides a concise method to check the separation of duty conflicts, which is obviously not sufficient to make sure the consistency of system. All of the above work addressed little on the policy redundancy, which is obviously not safe and efficient enough for the policy management.

The redundancy and inconsistency management need a systematic way (Li *et al*., 2007). In this study, pointing to the problems listed above we take all of the elements of a RBAC policy including role hierarchies, separation of duty constraints and cardinality constraints into consideration and present the formal definition of the policy redundancy and inconsistency. We also show the algorithm to check the redundancy and inconsistency of RBAC access control policy which is based on the Tarjan's SCC algorithm (Tarjan, 1972). Compared to the methods by Chang and Hoh (2005) and Strembeck (2004), present will provide much more confidence for the consistency in RBAC policy.

The purpose of this study is to develop a comprehensive algorithm to check the policy redundancy and inconsistency to support the management of RBAC policy. The algorithm should take all the policy constraints into account and due to the frequent policy checking in daily management, the algorithm should be efficient enough for practical application.

BACKGROUND

RBAC is widely used for the specification of security requirement of most commercial applications (Yao *et al*., 2001; Park *et al*., 2001; Essmayr *et al*., 2004). RBAC96 model currently being used consists of the following four basic components: a set of users, a set of roles, a set of permissions and a set of sessions. A user is a human being or a process within a system. A role is a collection of permissions associated with a certain job function. Permission defines the access rights that can be exercised on a particular object in a system. A session relates a user to several roles. The whole picture of RBAC96 model is as shown in Fig. 1.

Definition 1

RBAC policy: An RBAC policy is a 7-tuple (U, R, P, RH, RP, UR, C):

- U, R, P are respectively, finite sets of users, roles and permissions, where, P is defined as $P = 2^{Operations \times Objects}$
- **RH:** Role hierarchy is a partial order relation defined on role sets, $RH \subseteq (R \times R)$, if $(r_1, r_2) \in RH$, r_1 is a senior role whereas r_2 is junior role and r_1 inherits all the permissions of r_2. The relation can also be expressed as: $r_1 > r_2$

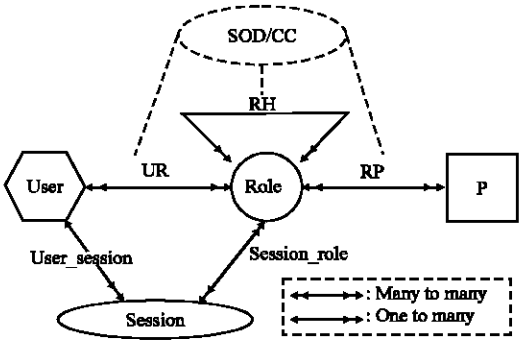

Fig. 1: RBAC96 model

- **RP:** Role-permission is a relation defined on role set and permission set, $RP \subseteq (R \times P)$
- **UR:** User-pole is a relation defined on the user set and role set, $UR \subseteq (U \times R)$
- **C:** The constraint set defined on the above entity and relation sets, including the separation of duty and cardinality constraint

RBAC global functions:

- **senior:** $R \rightarrow 2^R$, mapping each role to a set of roles, which are senior to the role
- **role_u:** $R \rightarrow 2^U$, mapping each role to a set of users who own the role, including the ones who own the role via., role hierarchy
- **role_p:** $R \rightarrow 2^P$, mapping each role to a set of permissions which are assigned to the role, including the ones which are inherited from senior roles
- **user_p:** $U \rightarrow 2^P$, mapping each user to a set of permissions which are owned by the user
- **user_r:** $U \rightarrow 2^R$, mapping each user to a set of roles which are assigned to the user
- **pms_u:** $P \rightarrow 2^U$, mapping each permission to a set of users who owns the permission
- **pms_r:** $P \rightarrow 2^R$, mapping each permission to a set of roles, which have been assigned the permissions

Separation of Duty (SOD) is a fundamental technique to prevent fraud and errors, known and practiced long before even the existence of computers. SOD reduces the possibility of fraud or significant errors which may cause damage to an organization. By partitioning the tasks and the associated privileges, the accomplishment of a sensitive task requires the cooperation of multiple users imperatively. SOD constraint is not only the most important constraint in RBAC policy but also the most complicated part which causes the management of RBAC policy more difficult. There are three kinds of separation of duty in RBAC, permission separation SOD-P, role separation SOD-R and user separation SOD-U.

Definition 2
Separation of duty:

- **SOD-P:** SOD-P is the finest grained control of separation of duty, which specifies the exclusive set of permission

SOD-P$\subseteq 2^{P}\times N$, which satisfies the following conditions:

$$\forall n \geq 2, \forall (ps,n) \in SOD-P, \forall p \subseteq ps \mid p \mid > n \Rightarrow \bigcap_{per \in p} pms\_u(per) = \varnothing$$

If n = 1, we can use $(p_1, p_6) \in$SOD-P for simplicity.

- **SOD-R:** SOD-R is the constraint over the role set, which specifies the user could not be assigned the exclusive roles in the meanwhile

SOD-R$\subseteq 2^{R}\times N$, which satisfies the following: $\forall n \geq 2$, $\forall$(rs, n) $\in$ SOD-R

$$\forall s \subseteq rs \mid s \mid > n \Rightarrow \bigcap_{res} role\_u(r) = \varnothing$$

The simplified expression can be written as $(r_1, r_2) \in$SOD-R, when n = 1

- **SOD-U:** SOD-U is the third form of separation of duty, which specifies that only one of the two users could be assigned the certain role. SOD-U$\subseteq 2^{U}\times R$, which satisfies the following conditions:

$$\forall (\{u_1, u_2\}, r) \in SOD-U \Rightarrow r \notin user\_r(u_1) \cap user\_r(u_2)$$

Besides the above SOD constraints, Cardinality Constraint (CC) is also one of the most important constraints in RBAC. Cardinality constraint regulates that at the given time, there is only one can be assigned as the certain role. For example in a company, at any time, there should be only one employee can be assigned as the general manager, otherwise there must be confusion of the management.

Definition 3
Cardinality Constraint (CC):

- **CC-P:** CC-P constraint specifies the restriction that the certain permissions which can be assigned to role only once. CC-P$\subseteq$P, which satisfies the following condition:

$$\forall p \in CC-P, \Rightarrow \mid pms\_r(p) \mid \leq 1$$

- **CC-R:** CC-R constraints specifies the restriction that the specified role can only be assigned to one user. CC-R$\subseteq$R, which satisfies the following condition:

$$\forall r \in CC-R, \Rightarrow \mid role\_u(r) \mid \leq 1$$

POLICY MANAGEMENT

The major problems encountered in the management of RBAC policy are redundancy and inconsistency. Redundancy means there are unwanted specifications to describe the control rules and inconsistency means there are conflicting rules for the access control.

Take following policy pl for example:

$$pl = (U, R, P, RH, UR, C)$$

Where:

U = $\{u_1, u_2\}$
R = $\{r_1, r_2, r_3, r_4, r_5, r_6, r_7\}$
p = $\{p_1, p_2, p_3, p_4, p_5, p_6\}$
RH = $\{(r_1, r_2), (r_2, r_3), (r_1, r_3), (r_5, r_6), (r_6, r_4), (r_4, r_5), (r_7, r_3), (r_7, r_4)\}$
RP = $\{(r_3, p_1), (r_3, p_2), (r_4, p_6)\}$
UR = $\{(u_1, r_1), (u_2, r_5)\}$
SOD-P = $\{\{p_2, p_4\}\}$
SOD-R = $\{\{r_3, r_4\}\}$
SOD-U = $\{(\{u_1, u_2\}, r_5)\}$, CC-R(r_5)

There are several redundancies and inconsistencies in the above RBAC policy. Depicting the policy pl in the Fig. 2, the redundancy of the role hierarchies among r_1, r_2 and r_3 is obvious, since the hierarchy relation of $r_1 \succ r_3$ can be deducted from $r_1 \succ r_2$ and $r_2 \succ r_3$. Redundancy in most situations has no dramatically negative effect on the access control. However, it will threaten the security as the evolution of the policy. For policy pl, if the security administrator wants to remove the hierarchy relation

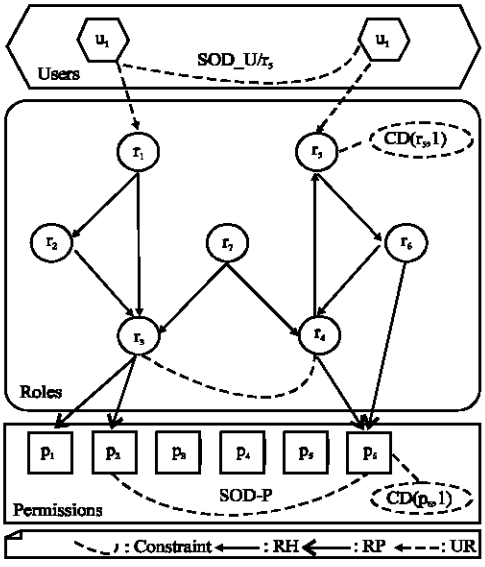

Fig. 2: Example RBAC policy

between r_1 and r_2, the intention of the administrator is also to cancel all the hierarchy relation which inherits from r_2, however, since $r_1 \succ r_3$ exists as a redundant relation which cause the policy unconformable.

Above RBAC policy also suffers from policy inconsistency. For the hierarchy relations among r_4, r_5 and r_6, it is obvious that there should be no cycle in the role graph as hierarchy is a kind of partial order relation.

Boolean matrix for relations: The above policy example demonstrates that the management of RBAC policy is fundamental and challenging. Boolean matrix is a good way to express the relations among the users, roles and permissions.

Definition 4
Multiplication of Boolean matrix: Boolean matrix multiplication between Boolean matrices $A \in \{0, 1\}^{l \times m}$ and $B \in \{0, 1\}^{m \times n}$ and $A \otimes B = C$, where $C \in \{0,1\}^{l \times m}$ and

$$c_{ij} = \bigvee_{k=1}^{m} (a_{ik} \wedge b_{kj})$$

Definition 5
MRH, MUR, MRP and MUP: MRH, MUR, MRP and MUP are the matrices to represent the RH, UR, RP and user-permission relation in the policy. Setting $l = |U|$, $m = |R|$, $n = |P|$

- **MRH:** An m×m Boolean matrix, satisfying:

$$\forall r_1, r_2 \in R,\ MRH(r_1, r_2) = \begin{cases} 1, & \text{if } (r_1, r_2) \in RH \\ 0, & \text{if } (r_1, r_2) \notin RH \end{cases}$$

- **MUR$^+$:** The transitive closure of MRH, representing the complete hierarchy relations on role set
- **MUR:** An l×m Boolean matrix, satisfying:

$$\forall r \in R, u \in U,\ MUR(u,r) = \begin{cases} 1, & \text{if } (u,r) \in UR \\ 0, & \text{if } (u,r) \notin UR \end{cases}$$

- **MUR$^+$:** An l×m Boolean matrix, satisfying:

$$MUR^+ = MRU \otimes MRH^+$$

- **MRP:** An m×n Boolean matrix, satisfying:

$$\forall r \in R, p \in P,\ MRP(r,p) = \begin{cases} 1, & \text{if } (r,p) \in RP \\ 0, & \text{if } (r,p) \notin RP \end{cases}$$

- **MRP$^+$:** An m×n Boolean matrix, satisfying:

$$MRP^+ = MRH^+ \otimes MRP$$

which indicate the whole role and permission relation

- **MUP:** l×n Boolean matrix, representing the relations between users and permissions, satisfying:

$$MUP = MUR \otimes MRP^+$$

RBAC policy redundancy: Redundancy means there are unwanted specifications to describe the control rules, following lists the types of the redundancy:

- Redundancy between role hierarchies

For $r_i \succ r_j$, if $\exists r_k \ldots r_{k+l} \in R$ satisfying:

$$r_i \succ r_k \wedge r_k \succ r_{k+1} \wedge \ldots \wedge r_{k+l-1} \succ r_{k+l} \wedge r_{k+l} \succ r_j$$

then $r_i \succ r_j$ is redundant.

The proof is simple, since $\succ$ is a transitive relation, via., $r_k \ldots r_{k+l}$, $r_i \succ r_j$, can be deducted. Figure 3a shows the role hierarchy redundancy.

- Redundancy between SOD-P and SOD-R

For sod-r(r_1, r_2) $\in$ SOD-R,
if $\exists$sod-p(p_1, p_2)$\in$SOD-P, satisfying:

$$p_1 \in role\_p(r_1) \wedge p_2 \in role\_p(r_2) \text{ or}$$

$$p_1 \in role\_p(r_2) \wedge p_2 \in role\_p(r_1)$$

then sod-r(r_1, r_2) is redundant. Figure 3b shows this type of redundancy.

- Redundancy between SOD-U and CC-R

For sod-u({u_1, u_2}, r)$\in$SOD-U, if $\exists$cc-r(r), $\in$CC-R, then sod-u({u_1, u_2},r) is redundant. Figure 3c shows this type of redundancy

Following is the algorithm to check the redundancies:

Algorithm 1: RBAC policy redundancy checking
Input: Policy pl = (U, R, P, RH, UR, C)
Output: Return false indicating there is no redundancy in the policy, otherwise return true indicating redundancy exists.

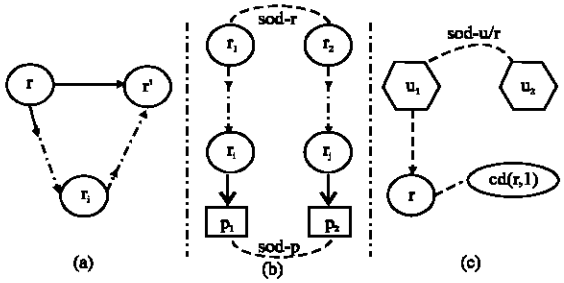

Fig. 3a-c: RBAC policy redundancy

- For policy pl, construct the relation matrices, MRH, MUR, MRP and MUP and assuming the redundant set is RSet←⌀
- Retrieve next unchecked rh = (r_1, r_2) from RH, set MRH(r_1, r_2)←0, then using the Warshall algorithm compute the transitive closure of MRH, which is MRH^+
- If $MRH^+(r_1, r_2) = 1$, then (r_1, r_2) is a redundant role hierarchy relation, add (r_1, r_2) to the redundant set RSet, mark it as role hierarchy redundancy
- If $MRH^+(r_1, r_2) = 0$, then (r_1, r_2) is not redundant, reset MRH (r_1, r_2)←1 and continue step 2 until all the role hierarchy relation are checked
- Retrieve next unchecked sod-r(r_1, r_2), for each sod-p(p_1, p_2), if $MRP^+(r_1, p_1) = 1$, $MRP^+(r_2, p_2) = 1$ or =1 or MRP^+ (r_1, p_2) = 1 and $MRP_+(r_2, p_1) = 1$, then sod-r(r_1, r_2) is redundant, add it to the redundant set and mark it as SOD-R and SOD-P redundancy. Continue step 5 until all the sod-r constraints are checked
- For each sod-u({u_1, u_2}, r), if cc-r(r) = 1, then sod-u({u_1, u_2}, r) is redundant, add it to the redundant set
- If redundant set is empty, return false, otherwise return true

For redundancy checking, the redundancy between SOD-P and SOD-R and the redundancy of SOD-U and CC-R are easy to do, which can be accomplished according to the corresponding relation matrix, via., verifying whether the unwanted condition is true. The redundancy among the role hierarchy is more complicated than the former two, since it involves the transitive relation among roles, which cause the computation more time-consuming.

We utilize the Warshall algorithm (Floyd, 1962) to computer the research-ability between two role nodes in the policy graph. The time complexity is:

$$O(m^3), \text{ where, } m = |R|$$

m is the role number in the policy.

The efficiency can be improved via., using DFS (Depth First Search) based reach-ability algorithm, which can reduce the time complexity to:

$$O(m^2 + mn)$$

$$m = |R| \text{ and } n = \sum_{i=1}^{m}\sum_{j=1}^{m} MRH[i, j]$$

where, n is the role hierarchy relation edge in the role policy graph. In practice, usually, $n<<m^2$, so DFS based reach-ability algorithm offers an advantage here.

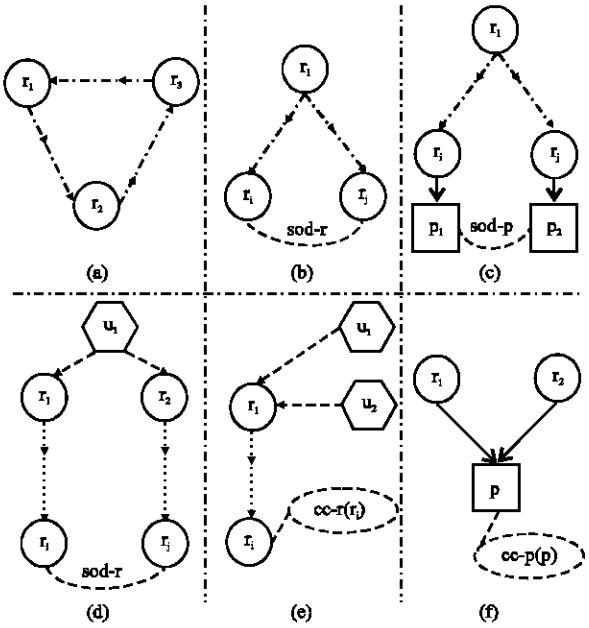

Fig. 4a-f: RBAC policy inconsistency

RBAC policy inconsistencies: A security and safe access control policy should give the indicated responses of an access request. Any confusion or uncertainty may conceal the security danger or even breakdown the entire system. Inconsistency is the major cause of the uncertainty. RBAC policy is more complicated than DAC and MAC, since RBAC supports role hierarchy relation and more advanced constraints which facilitate the fine grained control. Figure 4 shows the inconsistencies in an RBAC policy:

- Inconsistency between role hierarchies
  For r_1, r_2, $r_3 \in R$, if
  $r_1 \in$ senior(r_2)$\wedge r_2 \in$ senior (r_3) $\wedge$ $r_3 \in$ senior(r_1). Then there is inconsistency of the role hierarchies among r_1, r_2 and r_3. In Fig. 4a, r_1 is senior to r_2, r_2 is senior to r_3 and r_3 is senior to r_1, which makes all of the three roles involve the conflicts, since the junior should not get the permissions of the senior roles
- Inconsistency between RH and SOD-R
  For r_1, r_2, $r_3 \in R$ and sod-r(r_2, r_3), if
  $r_1 \in$ senior(r_2)$\wedge r_1 \in$senior(r_3)

Then there is inconsistency between the role hierarchy and SOD-R constraint. In Fig. 4b, r_1 is senior to role r_i and r_j, which are restricted by the SOD-R constraint. This inconsistency will expose the system under the danger, since any user who is assigned the role r_1 can do the operations of r_i and r_j at the same time which is supposed to be prohibited.

- Inconsistency between RP and SOD-P
  For $r_1 \in R$ and $p_1, p_2 \in P$, sod-p(p_1, p_2)
  if $p_1 \in$role_p(r_1)$\wedge p_2 \in$ role_p(r_1)

Then there is inconsistency between the role-permission assignment and the SOD-P constraint. In Fig. 4c, role r_1 gains the mutual exclusive permissions via., the role hierarchy, which may cause the security danger similar to the 1 in 2.

- Inconsistency between UR and SOD-R
  For $u \in U$, r_1, $r_2 \in R$ and sod-r(r_1, r_2) if $r_1 \in$ user_r(u)$\wedge r_2 \in$user_r(u)

Then there is inconsistency between the user-role assignment and the SOD-R constraint. In Fig. 4d user u_1 is assigned with role r_1 and r_2, which are involved in the SOD-R relation via., role hierarchy from r_i and r_j. Thus, u_1 gains the exclusive permissions which is not allowed.

- Inconsistency between UR and CD-R
  For $r \in R$ and cc-r(r) if
  $\exists u_1, u_2 \in U$, $u_1 \neq u_2$ and
  $r \in$role_u(u_1)$\wedge r \in$role_u(u_2)

Then there is inconsistency between user-role assignment and the cardinality constraint. In Fig. 4e, user u_1 and u_2 are assigned with role r_1, which has the cardinality constraint or inherits the cardinality constraint via., role hierarchy from role r_i, which cause the inconsistency.

- Inconsistency between RP and CD-P
  For $p \in P$ and cc-p(p) if
  $\exists r_1, r_2 \in R$, $r_1 \neq r_2$ and
  $p \in$ role_p(r_1)$\wedge p \in$ role_p(r_2)

Then there is inconsistency between role-permission assignment and CD-P constraint. In Fig. 4f , role r_1 and r_2 are assigned with the same permission p, which is restricted by the cardinality constraint. Such kind inconsistency will turn the system into complete confusion.

The above inconsistency definitions are the basic forms. Since, role hierarchy relation is transitive and other RBAC relations are transitive-kind relations, like the UP can be delivered via., UR and RP, there are more complicated inconsistencies which can be composed of the above basic forms.

Based on the analysis of RBAC policy inconsistency, the inconsistency checking algorithm is given as following:

Algorithm 2: RBAC policy inconsistency checking
Input: Policy pl = (U, R, P, RH, UR, C)
Output: Return false indicating there is no inconsistency; otherwise return true indicating inconsistency exists.

- Construct role graph RG = (V, E), where, V = R and E = RH and set the inconsistency set ISet$\leftarrow\varnothing$
- Find the Strong Connected Components (SCC) in RG via Tarjan's SCC algorithm, if the output is empty set then there is no role hierarchy inconsistency, otherwise, there are inconsistencies among the role hierarchy and the output set contains the roles which cause the inconsistency
- Retrieve next unchecked sod-p(p_1, p_2), for each $r \in R$, if $MRH^+(r, r_1)$ and $MRH^+(r, r_2) = 1$, then there is inconsistency of r, r_1, r_2 and sod-r constraints, add them to the ISet. Continue step 3 until all the sod-r constraints are checked
- Retrieve next unchecked sod-p(p_1, p_2) from the policy, for each $r \in R$, if $MRP^+(r, p_1) = 1$ and $MRP^+(r, p_2) = 1$, then add r and sod-p(p_1, p_2) to the inconsistency set. Continue step 4 until all the sod-p constraints are checked
- Find the next unchecked sod-r(r_1, r_2), for each $u \in U$, if $MUR^+(u, r_1) = MUR^+(u, r_2)$ 1, then sod-r(r_1, r_2) and u cause the inconsistency, add them to the ISet. Continue this step until all the sod-r constraints are checked.
- For each cc-r(r), if

$$\sum_{i=1}^{l} MUR(u_i, r) > 1$$

  (l is the user number), then there is inconsistency, add the user set $\{u \in U | MUR(u, r) = 1\}$ and cc-r(r) to the ISet
- For each cc-p(p), if

$$\sum_{i=1}^{n} MRP(r_i, p) > 1$$

  then there is inconsistency, add the role set $\{r \in R | MRP(r, p) = 1\}$ and cc-p(p) to the ISet
- If ISet is empty, return false, else return true

After calculate the relation matrix MRH, MRH^+, MUR, MRP, it is easy to check the inconsistencies via., the algorithm except the role hierarchy inconsistency.

The essential reason of the role hierarchy inconsistency is that there are strongly connected roles (i.e., role cycle) in the role graph. Tarjan's SCC algorithm is based on DFS, which can be used for the role hierarchy inconsistency checking. It begins from a starting role and the roles involved in a cycle will form a sub-tree of the whole DFS search tree and the root of the sub-tree is the root of the strongly connected roles.

The time complexity of Tarjan's SCC algorithm is:

$$O(m+k)$$

where, m = |R| and k = |RH|, so the complexity of inconsistency checking can be calculated as following:

$$O(m+k)+O(m*C_1)+O(m*C_2)+O(l*C_1)+O(C_3+C_4)$$

where in n=|P|, l=|U|, C_1=|SOD-R|<m^2, C_2=|SOD-P|<mn, C_3=|CC-R|<m and C_4=|CC-P|<n and since m<<*l* and n<<*l* so the time complexity is: $O(lm^2)$.

Although, we separate the steps to check the inconsistency for clarity, the steps can be combined to improve efficiency.

RESULTS AND DISCUSSION

The purpose of the experimental evaluation is two fold. First, we would like to validate that present algorithm works in a wide variety of environments ranging from small to large organization, with a few to a large number of roles and comprising of complex role hierarchy. Secondly, we would like to examine the performance of present approach with complex access control constraints. However, it is hard to find the real data of complex access control constraints that we would like to evaluate. Therefore, we created a test data generator that would allow us to verify the effectiveness of this algorithm under various conditions.

The test data generator performs as follows: first a set of roles are created, the number of which is set to a certain maximum number and the hierarchy relationships are constructed randomly under a certain ratio. Next for each role a random number of permissions are chose to form the role. Next the constraints among roles and permissions are generated randomly. Finally, the redundancy and inconsistency are generated according to the specified amount the generating algorithm is shown in Fig. 5. It is obvious that the generator uses a simple randomization strategy in order to generate the test data. This allows us to test in a completely unbiased manner with several different situations.

The input parameters of the test data generator algorithm are as follows:

NR = No. of roles
NRH = No. of role hierarchies
NU = No. of users
NRd = No. of redundancies
NI = No. of inconsistency

The experimental data is collected under the following test environment:

- **CPU:** Intel Core 2 T7500 @ 2.20 GH
- **Memory:** 2.00 GB
- **OS:** Windows XP, SP4
- **Program language:** Java
- **JRE Version:** 1.6.0_02

Table 1 shows the CPU time records of the redundancy checking method based on Warshall algorithm and DFS algorithm, in the table the column head RNum is row number and role hierarchy ratio is defined as:

$$\text{Ratio} = \frac{|RH|}{|R|}$$

According to the data in Table 1, we can get the conclusion that the Warshall based redundancy checking algorithm is much more sensitive to the role number. Obviously, the DSF based redundancy checking algorithm offers the significant performance advantage and DSF based method is actually the one we choose in our real application.

Table 2 shows the CPU time records of the inconsistency checking algorithm, as the role hierarchy ratio increases, the CPU time which is taken to perform inconsistency checking also increase. This makes sense since more hierarchies involved in the checking algorithm indicates that there are more edges in the role graph which makes the graph traverse more time consuming.

Table 1: CPU time records of redundancy checking based on Warshall algorithm and DSF

Role No.	Based on Warshall role hierarchy ratio					Based on DSF role hierarchy ratio				
(RNum)	0.1	0.2	0.3	0.4	0.5	0.1	0.2	0.3	0.4	0.5
100	0	15	0	16	0	3	0	0	2	0
200	48	47	48	32	31	13	13	14	11	10
300	125	141	141	140	141	25	27	28	27	27
400	312	313	328	312	313	46	46	48	46	46
500	610	625	625	610	625	72	73	72	72	73
600	1047	1047	1062	1078	1063	102	103	105	105	104
700	1656	1687	1687	1688	1687	140	141	141	142	143
800	2500	2532	2531	2562	2547	184	186	188	187	186
900	3578	3641	3609	3625	3688	234	237	235	236	239
1000	5000	5015	5047	5031	5047	292	293	292	292	293

```
Input: NU, NR, NRH, NRd, NI
Output: role system with redundancy and inconsistency
1     //generate permissions
2     randomly generate the permission set PS
2     //Create roles
3     while NR>0
4     NR ← (NR-1)
5     choose permissions from PS and assign to role tr randomly and add
      tr to role set RS
6   //generate the role hierarchies
7   while NRH>0
8     select two roles r1 and r2 randomly
9     if (r1, r2)∉RH and (r2, r1)∈RH
10    add (r1, r2) to role hierarchy set RH
11    NRH ← (NRH-1)
12  //create users
13  while NU>0
14    NU←(NU-1)
15    choose roles from RS and assign to user tu randomly and add tu to
      user set US
16  //create the constraints
17  generate the constraints randomly, including SOD and cardinality
    constraints randomly
18  //generate the role redundancy and inconsistency
19  computer MRH and MRH+
20  for (i = 0; i<|RS|; i++)
21    for (j = 0; j<|RS|; j++)
22      if (MRH(i, j) = 0 && MRH+(i, j) = 1 && (NRd > 0))
23        add (ri, rj) to RH
24        NRd ← (Nrd -1)
25      if (MRH+(i, j) = 0 && (NI > 0))
26        add (rj, ri) to RH
27        NI ← (NI-1)
```

Fig. 5: Test data generator

Table 2: CPU time records of inconsistency checking

Role No. (RNum)	Role hierarchy ratio 0.1	0.2	0.3	0.4	0.5
100	0	2	1	2	2
200	16	32	51	72	77
300	27	56	87	120	137
400	49	98	150	204	242
500	78	150	228	308	372
600	107	212	321	432	528
700	146	286	432	580	712
800	191	374	565	757	933
900	239	474	714	956	1182
1000	301	591	889	1190	1474

Compared to the methods by Chang and Hoh (2005) and Strembeck (2004), present approach is different in several ways: Firstly,present approach takes all the RBAC constraints into consideration, including three types of separation of duty and two types of cardinality constraints. Constrains, which provide the advanced manipulation of the access control policy, are the most important feature of RBAC model. Thus, the policy management methods with no support for the constraints are of no practical use. Present approach is the first attempt to take all the RBAC constraints, which makes this approach the one which could provide the maximum management confidence. Secondly, the access control policy management is the daily basic work of security administrator, thus the policy checking algorithm should be efficient enough due to frequent calls. From the above experimental evaluation, we can see that for an access system with role number up to 1000, present approach still maintain a high performance. According to present experience and the case study (Ferraiolo *et al*., 2003), the role number of enterprise system is much less than 1000, so the method is efficient enough for practical usage. Another advantage of present approach is that we reduce the redundancy and inconsistency to the graph problems, for which there have been a lot of existing well-known algorithms. Another point worth mentioning is the role hierarchy ratio, which is a kind of description of the complexity of the hierarchy relations. Similar to the concepts in graph theory, it could be considered as the graph complexity. Since, the role graph is a diagraph and in most circumstances is acyclic diagraph, the ratio of the number of edges and the number of vertexes will be relatively small. According to the experience, the role hierarchy ratio is usually less than 0.2, in such kind of situation, our approach demonstrates excellent performance. Although, the above work has not discussed the dynamic constraints, there is no significant difference compared to the static constraints except the checking happens at runtime, which requires the algorithm should be much more efficient and our approach can be extended easily to provide such kind of support.

CONCLUSIONS

In this study, we analyze the redundancy and inconsistency in role based access control policy. We give the formal definitions for each type of policy redundancy and inconsistency. Based on the analysis, the algorithms are given to check both redundancy and inconsistency. The redundancy checking algorithm is based on Warshall and depth first search algorithm and the redundancy can be identified via., detecting whether there is the redundant path in the role graph. The inconsistency checking algorithm is based on Tarjan's SCC algorithm and the inconsistency can be identified via checking whether the role cycle exists. Compared to the others similar studies, present methods covers all the elements of RBAC policy and the checking algorithm is efficient enough for the practical usage, which could guarantee the conciseness and consistency of the RBAC policy and prevent the system from concealing security danger. The future study includes the study of redundancy and inconsistency resolution. And the severity of different inconsistencies and the automated method of inconsistency resolution will be the focus of the future research.

REFERENCES

Beznosov, K. and Y. Deng, 1999. A framework for implementing role-based access control using CORBA security service. Proceedings of the 4th ACM Workshop on Role-Based Access Control, RBAC'99, ACM, New York, pp: 19-30.

Centonze, P., G. Naumovich, S.J. Fink and M. Pistoia, 2006. Role-based access control consistency validation. Proceedings of the 2006 International Symposium on Software Testing and Analysis, July 17-20, Portland, Maine, USA., pp: 121-132.

Chang, J. and P. Hoh, 2005. Inconsistency detection of authorization policies in distributed component environment. LNCS., 3325: 39-50.

Coyne, E.J., 1996. Role engineering. Proceedings of the 1st ACM Workshop on Role-Based Access Control, Nov. 30-Dec. 02, Gaithersburg, Maryland, USA., pp: 132-140.

Essmayr, W., S. Probst and E. Weippl, 2004. Role-based access controls: Status, dissemination and prospects for generic security mechanisms. Elect. Commun. Res., 4: 127-156.

Ferraiolo, D.F., R. Sandhu, S. Gavrila, D.R. Kuhn and R. Chandramouli, 2001. Proposed NIST standard for role-based access control. ACM Trans. Inform. Syst. Secur., 4: 224-274.

Ferraiolo, D.F., R. Chandramouli, G. Ahn and S. Gavrila, 2003. The role control center: Features and case studies. Proceedings of the 8th ACM Symposium on Access Control Models and Technologies, SACMAT'03, ACM, New York, pp: 12-20.

Floyd, R.W., 1962. Algorithm 97: Shortest path. ACM Commun., 5: 345-345.

Li, N. and M.V. Tripunitara, 2006. Security analysis in role-based access control. ACM Trans. Inform. Syst. Secur., 9: 391-420.

Li, N. and Z. Mao, 2007. Administration in role-based access control. Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security ASIACCS '07, Mar. 20-22, Singapore, ACM, New York, pp: 127-138.

Li, N., J. Byun and E. Bertino, 2007. A critique of the ANSI standard on role-based access control. IEEE Secur. Privacy, 5: 41-49.

Nyanchama, M. and S. Osborn, 1999. The role graph model and conflict of interest. ACM Trans. Inform. Syst. Secur., 2: 3-33.

Park, J.S., R. Sandhu and G. Ahn, 2001. Role-based access control on the web. ACM Trans. Inform. Syst. Secur., 4: 37-71.

Schaad, A., J. Moffett and J. Jacob, 2001. The role-based access control system of a European bank: A case study and discussion. Proceedings of the 6th ACM Symposium on Access Control Models and Technologies, SACMAT '01, ACM, New York, pp: 3-9.

Strembeck, M., 2004. Conflict checking of separation of duty constraints in RBAC implementation experiences. Proceedings of the Conference on Software Engineering, Mar. 2004, IEEE, pp: 21-27.

Tarjan, R., 1972. Depth-first search and linear graph algorithms. SIAM J. Comput., 1: 146-160.

Yao, W., K. Moody and J. Bacon, 2001. A model of OASIS role-based access control and its support for active security. Proceedings of the 6th ACM Symposium on Access Control Models and Technologies, SACMAT '01, Chantilly, Virginia, United States, pp: 171-181.