Indian Journal of Science and Technology Vol. 3 No. 6 (June 2010) ISSN: 0974- 6846

Classification and formulation of role-based separation of duty constraints

Marzieh Esna-Ashari[1] and Hamid R. Rabiee[2]
[1]*Department of Computer Science, School of Mathematics, University of Tehran, Tehran, Iran*
[2]*AICTC Research Center, Department of Computer Engineering, Sharif University of Technology, Tehran, Iran*
esnashar@khayam.ut.ac.ir; rabiee@sharif.ac.ir

Abstract
Separation of Duty (SOD) is a fundamental principle in security systems and has a long history in computer security research. It is also an important characteristic in the Role-Based Access Control (RBAC) system. Various categories of this principle have been pointed out for RBAC environment by different researchers, but they have neither been classified nor formulated. This paper beside classifying and formulating all the former introduced types of SOD in Role-based environment; presents and defines formally new types of Role-based SOD.
Keywords: Separation of duty (SOD), role-based access control (RBAC)

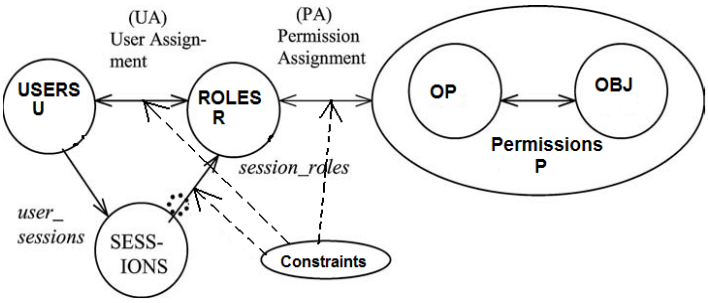
Introduction
Role-Based Access Control (*RBAC*) (ANSI, 2004; Feraiolo & Kuhn, 1992; Sandhu *et al*., 1996) has received a great deal of attention as a more flexible and promising alternative to the two conventional discretionary and mandatory access control (*DAC* and *MAC*) techniques (Joshi *et al*., 2001). One of the fundamental aspects of *RBAC* is authorization constraints (Jaeger, 1999) and one of the major classes of role-based authorization constraints is separation of duty (SOD) constraint.

Separation of duty (SOD) is considered to be a fundamental principle in security systems, both automated and manual (Clark & Wilson, 1987; Gligor *et al*.,1998; Sandhu *et al*., 1990). Although there are many variations, SOD is basically a requirement that critical operations are divided among two or more people, so that no single individual can compromise the security (Clark & Wilson, 1987; Sandhu, 1988; Zhang *et al*., 2005). With the increasing complexity in the applications of RBAC, more SOD variations have emerged. However, there is less literature to systematically classify and formulate the definitions of these SOD variations, while such a classification and formulation is necessary for developing a formal constraint language for role-based SOD. In view of the above problem, this paper systematically classifies SOD variations, presents the necessary components for each one and formulates their definitions. It also introduces some variations of SOD which are new to the literature.

A brief introduction to RBAC
RBAC is a policy-neutral and flexible, access control technique which controls the access of users to information and system resources on the basis of activities and responsibilities they have within the organization. With this technique the control of access is on the basis of roles each user has been assigned to by the organization. The core of *RBAC* is defined as a set of permissions, a set of roles, and a set of users (ANSI, 2004; Feraiolo & Kuhn, 1992; Sandhu *et al*. 1996). A role is a named job function within the organization and describes the authority and responsibility conferred on each member of the role. A user represents a human being or an autonomous agent, and a permission is an approval of a particular mode of access to one or more objects in the system. The basic concept of RBAC is that permissions are assigned to roles and individual users obtain such permissions by being assigned to roles. The permission-to-role and user-to-role assignments are both many-to-many relationships. Users establish sessions during which they may activate a subset of roles they belong to. A user might have multiple opened sessions simultaneously, each in a different window on a workstation screen.

Several *RBAC* models have been suggested and published, the most widely referenced model is RBAC96 (Sandhu *et al*., 1996) which is the basis of ANSI RBAC (ANSI, 2004). Fig.1 illustrates the general form of an RBAC system.

Fig. 1. RBAC System



The basic elements and System functions of RBAC as suggested (ANSI, 2004; Sandhu *et al*., 1996) are as follow:

- *U, R, OP,* and *OBJ* are the sets of users, roles, operations and objects respectively.
- $RH \subseteq R \times R$, a partial order on R called the role hierarchy.
- $UA \subseteq U \times R$, a many-to-many user-to-role assignment relation.

Indian Journal of Science and Technology Vol. 3 No. 6 (June 2010) ISSN: 0974- 6846

- *assigned_users*: $(r: R) \rightarrow 2^{U}$, a function for the mapping of role r onto a set of users. Formally: $assigned\_users(r) = \{u \in U \mid (u, r) \in UA\}$
- *assigned_roles*: $(u: U) \rightarrow 2^{R}$, the mapping of user u to the set of his assigned roles. Formally : $assigned\_roles(u) = \{r \in R \mid (u, r) \in UA\}$
- $P = OP \times OBJ$, the set of permissions.
- $PA \subseteq P \times R$, a many-to-many permission-to-role assignment relation.
- *assigned_permissions* $(r: R) \rightarrow 2^{P}$, the mapping of role r onto a set of permissions. Formally: $assigned\_permissions(r) = \{p \in P \mid (p, r) \in_{OP} PA\}$
- *operations* $(r: R ; obj: OBJ) \rightarrow 2$, the set of operations that role r is permitted to perform on object *obj*. Formally: $operations(r, obj) = \{op \in OP \mid p = (op, obj) \in P \wedge (p, r) \in PA\}$.
- *object* $(p: P) \rightarrow \{ob \subseteq OBJ\}$, the permission to object mapping, which gives the set of objects associated with permission p.
- S = the set of sessions
- *session_users* $(s: S) \rightarrow U$, the mapping of session s onto the corresponding user.
- *user_sessions* $(u: U) \rightarrow 2^{S}$, a function mapping each user u_i to a set of sessions.
- *session_roles* $(s: S) \rightarrow 2^{R}$, the mapping of session s onto a set of roles. Formally: $session\_roles(s_i) \subseteq \{r \in R \mid (session\_users(s_i), r) \in UA\}$
- *avail_session_perms* $(s: S) \rightarrow 2^{P}$, the permissions available to a user in a session $= \bigcup_{r \in session\_roles(s)} assigned\_permissions(r)$

Classifications of separation of duty in RBAC systems

Separation of duty (SOD) is a fundamental principle in security systems, both automated and manual. One of the *RBAC*'s great advantages is that SOD rules can be implemented in a natural and efficient way. In fact, SOD has been studied more in the context of *RBAC* than in any other field of computer security (Ferraiolo, 2003; Kong & Li, 2007; Zhang *et al*., 2005).

Although there are many variations, SOD is fundamentally a requirement that critical operations are divided among two or more people, so that no single individual can compromise the security. More generally, when a sensitive task is comprised of n-steps, an SOD policy requires the cooperation of at least k different users ($2 \leq k < n$) to complete the task (ANSI, 2004; Kuhn, 1997; Li *et al*., 2007).

Role-based separation of duty ensures SOD requirements in role based systems by controlling membership in, activation of, and use of roles as well as permission-to-role assignment (Ahn & Sandhu, 2000; Ferraiolo *et al*.,1995; Joshi *et al*., 2005). Various forms of SOD have been identified in *RBAC* (Zhang *et al*., 2005; Kong & Li, 2007). We focus on the most significant forms that are likely to be important in practice. A comprehensive survey by Simon and Zurko (Simon & Zurko, 1997) founds two broad categories of SOD methods in *RBAC* : static and dynamic.

Static SOD (*SSOD*) constraint puts the restrictions on the assignment stage that is to enforce constraints on the assignments of permissions-to-roles and roles-to-users. A common example is mutually disjoint user assignments with respect to sets of conflicting roles. For example, consider the purchasing department and the processing of invoices. Let us assume that there are four types of transaction involved in processing an invoice: create an order (Data entry clerk), approve it (Supervisor), verify its receipt (Purchasing officer) and authorize its payment (Department's Manager). The policy may require that a single individual not be assigned to three or more of these roles.

Dynamic SOD (*DSOD*) puts the restrictions on the activation time and limits the availability of the roles (and as a result the permissions) which have been assigned to a user, within or across working sessions. This model limits the availability of the permissions over a user permission space by placing constraints on the roles that can be activated within or across a user's sessions. *DSOD* allows a user to be authorized for two or more roles that do not create a conflict of interest when acted independently, but produce policy concerns when activated simultaneously. For example, a user may be authorized for both the roles of Cashier and Cashier Supervisor, but is not permitted to activate both roles simultaneously.

Each of these broad categories may further be classified into more specific types, where some of them have been previously defined in the literature. We summarize all the various types of SOD including some newly identified types in each of the two mentioned broad categories.

- ***Static SOD*** *(SSOD)* puts the restriction on the assignment stage and is further divided into the following classes.
  - Role-centric *SSOD* (*R-SSOD)*.
  - Permission-centric *SSOD* (*P-SSOD*).
  - User-centric *SSOD* (*U-SSOD*).
  - Object-based *SSOD* for **s**ensitive objects (*Ob-SSOD-S)*. This is new to the literature.
  - Object-based *SSOD* for conflicting object sets *(Ob-SSOD-C)*.
  - Operation-based *SSOD* (*Op-SSOD*).
- ***Dynamic SOD*** *(DSOD)* puts restriction on the activation time, and is further divided into the following types.
  - Role-centric *DSOD* (*R-DSOD*).
  - User-centric *DSOD* (*U-DSOD*).

Indian Journal of Science and Technology Vol. 3 No. 6 (June 2010) ISSN: 0974- 6846

– Object-based *DSOD* for sensitive objects (*Ob-DSOD-S*). This is new to the literature.
– Object-based *DSOD* for conflicting object set (*Ob-DSOD-C*). This form of SOD is a historical constraint and could be used to implement the Chinese wall model.
– Operational-based *DSOD* (*Op-DSOD*).

According to our knowledge, there are no systematic classifications and formulated definitions for all of the above identified SOD classes. Only role centric ones, i.e. *R-SSOD* and *R-DSOD* pose a formal definition (ANSI, 2004; Ferraiolo *et al*., 2003) and a somehow abbreviated formulated definition for operational based and history based exists (Kong & Li, 2007), the others lack such a definition. We pose the general formulated definitions for all of the identified SOD classes.

Formal and formulated definitions of static SODs

The general formal definitions of static SOD types are as follow:

Definition 1*: Role-centric SSOD (R-SSOD)* defines mutual exclusions among roles and requires that no single user can be assigned to *n* or more roles in one statically conflicting role set simultaneously; where *n* is a natural number ≥2 and less than the cardinality of the conflicting role set. To address this potential, a statically conflicting role collection named *SCR* is defined as follow:

SCR $\subseteq (2^R \times N)$ is a collection of pairs (*rs*, *n*), where each *rs* $\subseteq R$ is a conflicting role set and *n* is a natural number ($2 \le n < |rs|$) with the property that no single user is permitted to be assigned simultaneously to *n* or more roles from the role set *rs* in each (*rs*, *n*) $\in$ *SCR*. This definition can be formulated as follow:

$$\forall (rs, n) \in SCR, rs \in 2^R, (n \in N, 2 \le n < |rs|) \wedge \forall u \in U \Rightarrow |assigned\_roles(u) \cap rs| < n \quad (1)$$

Definition 2*: Permission centric SSOD (P-SSOD)* defines mutual exclusions among permissions and puts restriction on permission-to-role assignment. For this purpose a conflicting permission collection named *CP* is defined as follow:

CP $\subseteq (2^P \times N)$ is a collection of pairs (*ps*, *n*), where each *ps* $\subseteq P$ is a conflicting permission set and *n* is a natural number ($2 \le n < |ps|$) with the property that no single role is permitted to be authorized to *n* or more permissions from the permission set *ps* in each (*ps*, *n*) $\in$ *CP*. Formally:

$$\forall (ps, n) \in CP, ps \in 2^P, (n \in N \wedge 2 \le n < |ps|) \wedge \forall r \in R \Rightarrow |assigned\_permission(r) \cap ps| < n. \quad (2)$$

Definition 3*: User centric SSOD (U-SSOD)* defines mutual exclusions among users and states that: the union of the roles of the users in any statically conflicting user set should be consistence with the restriction of any member of the collection *SCR*. To address this, a statically conflicting user collection called *SCU* is defined as follow:

SCU $\subseteq 2^U$ is a collection of statically conflicting user sets, with the property that for any (*rs*, *n*) $\in$ *SCR*, the union of assigned roles to the users in any *us* $\in$ *SCU,* should not include *n* or more roles from the role set *rs*. This can be formulated as:

$$\forall us \in SCU \wedge \forall (rs, n) \in SCR \Rightarrow \left| \left(\bigcup_{u \in us} assigned\_roles(u) \right) \cap rs \right| < n. \quad (3)$$

Definition 4*: Object-based SSOD for sensitive objects (Ob-SSOD-S)* defines collection of statically sensitive objects and requires that no single role or single user should have permissions to do more than one operation on a single statically sensitive object. This form of SOD is new to the literature, though the idea has been taken from some informal definition in the subject of access control (Tang *et al*., 2007). To address this, the collection of statically sensitive objects is defined as follow:

SSO $\subseteq$ *OBJ* is the set of statically sensitive objects with the property that no single role or single user through his assigned roles is permitted to do more than one operation on each *ob* $\in$ *SSO*. Formally:

$$\forall r \in R \wedge \forall u \in U \wedge \forall ob \in SSO \Rightarrow |operations(r, ob)| \le 1 \wedge \left| \bigcup_{r \in assigned\_roles(u)} operations(r, ob) \right| \le 1. \quad (4)$$

Definition 5*: Object-based SSOD for statically conflicting object sets (Ob-SSOD-C)* defines mutual exclusions among objects and states that no single role or single user may be authorized to access *n* or more objects from the same statically conflicting object set, where *n* is a natural number ≥2 and less than the cardinality of that conflicting object set. For this purpose, the statically conflicting object collection *SCO* is defined as follow:

SCO $\subseteq 2^{OBJ} \times N$ is a collection of pairs (*os*, *n*), where each os $\subseteq$ *OBJ* is a conflicting object set and *n* is a natural number ($2 \le n <$ |os|) with the property that no single role or single user through his assigned roles may have the permission to access *n* or more objects from the object set *os.* Before fomulating the definition of this SOD class, we define the function *user_permissions* for returning the set of permissions a user gains through his assigned roles as:

user_permissions (*u* : *U*) = $\bigcup_{r \in assigned\_roles(u)} assigned\_permissions(r)$.

Indian Journal of Science and Technology Vol. 3 No. 6 (June 2010) ISSN: 0974- 6846

The formulated definition of *Ob-SSOD-C* is as follow:

$$\forall r \in R, \forall u \in U, \forall (os, n) \in SCO, os \in 2^{OBJ}, (n \in N \wedge 2 \le n < |os|) \Rightarrow$$

$$\left| \left(\bigcup_{p \in assigned\_permissions(r)} object(p) \right) \cap os \right| < n \ \wedge \quad (5)$$

$$\left| \left(\bigcup_{p \in user\_permissions(u)} object(p) \right) \cap os \right| < n.$$

Definition 6*: Operation-based SSOD (Op-SSOD)* requires that no single role or single user should have the authorization to be able to complete all the operations in the set of operations of a business task. If each business task be represented by at least two roles (sub tasks) then this form of *SSOD* can be easily represented by Role-centric *SSOD* (*R-SSOD*).

Formal and formulated definitions of dynamic SODs

The general formal definitions for dynamic SOD types are as follow:

Definition 7*: Role-centric DSOD (R-DSOD)* requires that no single user should be able to activate *n* or more roles from a single dynamically conflicting role set simultaneously, where *n* is a natural number ≥2 and less than the cardinality of that conflicting role set. The collection *DCR* is defined for this purpose.

DCR $\subseteq (2^R \times N)$ is a collection of pairs (*rs*, *n*), where each *rs* $\subseteq R$ is a dynamically conflicting role set and *n* is a natural number ($2 \le n < |rs|$) with the property that no single user is permitted to activate simultaneously, *n* or more roles from the role set *rs* in each (*rs*,*n*)$\in$*DCR*. This definition is formulated as:

$$\forall (rs, n) \in DCR, rs \in 2^R, (n \in N \wedge 2 \le n < |rs|) \wedge \forall u \in U$$

$$\Rightarrow \left| \left(\bigcup_{s \in user\_sessions(u)} session\_roles(s) \right) \cap rs \right| < n. \quad (6)$$

Definition 8*: User centric DSOD (U-DSOD)* states that dynamically conflicting users should not be able to activate *n* or more roles from the same dynamically conflicting role set, simultaneously (2≤*n* <cardinality of the conflicting role set). To address this, a collection named *DCU* is defined as follow:

DCU $\subseteq 2^U$ is a collection of dynamically conflicting user sets with the property that at any time for any (*rs*, *n*) $\in$*DCR* the union of the activated roles by the users in any *us*$\in$*DCU* should not include *n* or more roles from the role set *rs*. Formally:

$$\forall us \in DCU \wedge \forall (rs, n) \in DCR, ss = \bigcup_{u \in us} user\_sessions(u)$$

$$\Rightarrow \left| \left(\bigcup_{s \in ss} session\_roles(s) \right) \cap rs \right| < n \quad (7)$$

Definition 9*: Object-based DSOD for sensitive objects (Ob-DSOD-S)* defines collection of dynamically sensitive objects and requires that no single role or single user should ever be capable to do more than one operation on a single dynamically sensitive object. The collection *DSO* is defined for this purpose, as follow:

DSO $\subseteq$*OBJ* is the set of dynamically sensitive objects with the property that no single role or single user through his assigned roles can ever do more than one operation on each *ob*$\in$*DSO*. This form of SOD is a historical form and like *Ob-SSOD-S*, is new to the literature. For the formal specification of this form of SOD, we need two logs to keep the history of the accesses of roles and users to objects as follow:

- *RLOG* $\subseteq (R \times OP \times OBJ)$ is a collection of triples (*r*, *op*, *ob*), where *op* is an operation which has been performed by the role *r*, on the object *ob*.
- *ULOG* $\subseteq (U \times OP \times OBJ)$ is a collection of triples (*u*, *op*, *ob*), where *op* is an operation which has been performed by the user *u,* on the object *ob*.

After any access to any object these two logs should become updated. For extracting the required information from these two logs, we need two functions which we define them as follow:

- *role_ object_operations* (*r* : *R*, *ob* : *OBJ*) $\rightarrow 2^{OP}$, a function for mapping each role *r* and object *ob* in *RLOG* to a set of operations. Formally:

role_ object_operations (r_i, ob_i) = {*op*$\in$*OP* | (r_i, *op*, ob_i) $\in$*RLOG*}.

- *user_ object_operations* (*u* : *U*, *ob* : *OBJ*) $\rightarrow 2^{OP}$, a function for mapping each user *u* and object *ob* in *ULOG* to a set of operations. Formally:

user_ object_operations(u_i, ob_i) = {*op* $\in$ *OP* | (u_i, *op*, ob_i) $\in$*ULOG*}.

The formulated specification of *Ob-DSOD-S* is as follow:

$$\forall ob \in DSO \wedge \forall r \in R \wedge u \in U \Rightarrow |role\_object\_operations(r, ob)| \le 1 \wedge |user\_object\_operations(u, ob)| \le 1. \quad (8)$$

Definition 10*: Object-based DSOD for conflicting object set (Ob-DSOD-C)* defines dynamically conflicting object sets. For this purpose, the collection *DCO* is defined as follow:

DCO $\subseteq (2^{OBJ} \times N)$ is a collection of pairs (*os*, *n*), where each *os* $\subseteq$*OBJ* is a dynamically conflicting object set and *n* is a natural number ($2 \le n < |os|$) with the property that no single user or single role or single permission can ever access *n* or more object from the object set *os* in each (*os*, *n*)$\in$ *DCO*.

This type of SOD is also a historical constraint, and for its formal specification, we need the two logs defied in Definition 8, and for extracting the required information from these two logs, the following two functions are needed.

Indian Journal of Science and Technology Vol. 3 No. 6 (June 2010) ISSN: 0974- 6846

- *role_objects* ($r : R$) $\rightarrow 2^{OBJ}$, the mapping of role r in *RLOG* onto a set of objects. Formally:

$$role\_objects(r) = \{ob \in 2^{OBJ} \mid (\exists op \in OP)[(r, op, ob) \in RLOG]\}$$

- *user_objects* ($u : U$) $\rightarrow 2^{OBJ}$, a function mapping each user u in *ULOG* onto a set of objects. Formally:

$$user\_objects(u) = \{ob \in 2^{OBJ} \mid (\exists op \in OP)[(u, op, ob) \in ULOG]\}$$

Now we formulate the definition of *Ob-DSOD-C as:*

$$(\forall (os, n) \in DCO, os \in 2^{OBJ}, (n \in N \wedge 2 \le n < |os|) \wedge \forall r \in R \wedge \forall u \in U \wedge \forall p \in P \Rightarrow |role\_objects(r) \cap os| < n \wedge |user\_objects(u) \cap os| < n \wedge |object(p) \cap os| < n. \quad (9)$$

Definition 11*: Dynamic Operational-based SOD (Op-DSOD)* states that: no single role or single user should ever be able to complete all the operations in the set of operations of a single business task. If each business task be represented by at least two roles then this form of *DSOD* can be easily concluded from *R-DSOD*.

Conclusions and future works

Separation of duty is an important principle in computer security research and is argued to be one of the principle motivations behind RBAC (Ferraiolo *et al.*, 2003; Ahn & Sandhu, 2000). As the result of the growing complexity in the computer security mechanism, the need for new varieties of SOD is inevitable. The new varieties need to be identified, classified and defined formally. In this paper we identified all the necessary role-based SODs for today's computer security mechanism. We identified six classes of static and five types of dynamic SODs and systematically classified them. We then offered their formulated definitions and the necessary component that each one needs.

In our future work we are going to present an intuitive formal constraint language that will be able to specify all of the so far identified SOD classes.

References

1. Ahn G and Sandhu R (2000) Role-Based Authorization Constraints Specification. *ACM Transaction on Information and System Security*, 3 (4), pp: 207-226.
2. ANSI (2004) American National Standards for Information Technology- Role Based Access Control, *ANSI_INCITS 359-2004*.
3. Clark D and Wilson D (1987) A comparison of commercial and military computer security policies. *IEEE Symposium on Computer Security & Privacy*. pp: 184-194.
4. Ferraiolo D and Kuhn R, (1992) Role-based access control. The original paper on RBAC. *Proc. of 15th NIST- National Computer Security Conference*, Baltimore, MD. pp: 554-563.
5. Ferraiolo D, Cugini J and Kuhn R (1995) Role-Based Access control (RBAC): Features and Motivations. *Proceedings of the 11th Annual Computer Security Applications Conference*. pp: 242-248.
6. Ferraiolo D, Kuhn R and Chandramouli R (2003) Role-based access control. Artech House Publ., *Computer Security Series*, www.artechhouse.com.
7. Gligor V, Gavrila S and Ferraiolo D (1998) On the formal definition of separation of duty policies and their composition. *Proc. IEEE Computer Soc. Sym. on Res. in Security & Privacy*, Oakland, CA. IEEE Computer Society Press Los Alamedas, CA. pp: 172-183.
8. Jaeger T (1999) On the increasing importance of constraints. *Proc. of the 4th ACM Workshop on Role-based Access Control*, pp: 33-42.
9. Joshi J, Bertino E, Latif U and Ghafoor A (2005) Generalized temporal role-based access control model. *IEEE Transactions on Knowledge & Data Engg*. 17 (1), pp: 4-23.
10. Joshi J, Ghafoor A, Aref W, and Spafford E, (2001) Digital government security infrastructure design challenges. *IEEE Computer*. 34(2), pp: 66-72.
11. Kong G and Li J (2007) Research on RBAC-based separation of duty constraints. *J. Information & Computing Sci*. 2(3), pp: 235-240.
12. Kuhn R (1997) Mutual exclusion of roles, a means of implementing separation of duty in role- based access control systems. *Proc. of the 2nd ACM Workshop on Role-based Access Control* (RBAC' 97), Fairfax, VA. ACM Press, NY. pp: 23-30.
13. Li N, Bizri Z, and Tripunitara M (2007) On mutually-exclusive roles and separation of duty. *ACM transaction on information and system security* (ACM TISSEC), 10(2), pp: 5-36.
14. Sandhu R (1988) Transaction control Expressions for separation of duties. *IEEE 4th Aerospace Computer Security Applications Conference*, Oakland, CA. pp: 282-286.
15. Sandhu R (1990) Separation of duties in computerized information systems. *Proc. of the IFIP WG11.3 Workshop on Database Security*, Halifax,UK.pp:18-21
16. Sandhu R, Coyne E, Feinstein H and Youman C (1996) Role-based access control models. *IEEE Computer*. 29(2), pp: 38-47.
17. Simon R and Zurko M (1997) Separation of duty in role based environments. *Proceedings of 10th IEEE Computer Security Foundations Workshop*, Rockport, MA. pp: 183-194.
18. Tang Z, Li R, Lu Z and Wen Z (2007) Dynamic Access Control Research for Inter-operation in Multi-domain Environment Based on Risk. *Lecture Notes in Computer Science Information Security Applications: 8th Intl. Workshop*. pp: 277-290.
19. Zhang Z, Geng Y, Li T and Xiao J (2005) Analysis of enhanced separation of duty in role-based access control model. *Proc. of the 11th Joint Intl. Computer Conf,* Chonqing, China. World Sci. Publ. Co. pp: 69-72.