A Role-Based Access Control Policy Verification Framework for Real-Time Systems

Basit Shafiq
Purdue University
shafiq@ecn.purdue.edu

Ammar Masood
Purdue University
ammar@ecn.purdue.edu

James Joshi
Univ. of Pittsburgh
joshi@mail.sis.pitt.edu

Arif Ghafoor
Purdue University
ghafoor@ecn.purdue.edu

Abstract

This paper presents a framework for verifying the access control requirements of real-time application systems such as workflow management systems and active databases. The temporal and event-based semantics of these applications can be expressed using event-driven Role Based Access Control (RBAC) model. Any comprehensive access control model such as RBAC requires verification and validation mechanisms to ensure the consistency of access control specification. An inconsistent access control specification exposes the underlying system to numerous vulnerabilities and security risks. In this paper, we propose a Petri-Net based framework for verifying the correctness of event-driven RBAC policies.

1. Introduction

Role based access control (RBAC) has emerged as a promising alternative to traditional discretionary and mandatory access control (DAC and MAC) models, which have some inherent limitations [9, 15]. Several beneficial features such as policy neutrality, support for least privilege, efficient access control management, are associated with RBAC models [9, 16]. The concept of role is associated with the notion of functional roles in an organization, and hence RBAC models provide intuitive support for expressing organizational access control and are suitable for handling access control requirements of diverse organizations and emerging service-based applications such as e-commerce, healthcare-systems, etc. [5, 9]. Furthermore, use of role hierarchies and grouping of objects into object classes based on responsibility associated with a role simplifies management of access permissions. RBAC constraints allow expressing user-specific access control policies, and DAC and MAC policies, thus, increasing the applicability of RBAC models. In particular, many separation of duty (SoD) constraint can be easily specified to cater to the access control needs of many commercial applications [1, 14].

This work was supported in part by NSF award IIS 0209111.

By configuring the assignment of the least set of privileges from a role set assigned to a user when the user activates the role, inadvertent damage can be minimized in a system. RBAC models have also been found suitable for addressing security issues in the Internet environment, and show promise for newer heterogeneous multi-domain environments that raise serious concerns related to access control across domain boundaries [2, 9, 10].

RBAC has been widely researched and has been extended by several researchers [16, 14, 7]. One such crucial extension is an RBAC model with temporal constraints, which was proposed in Temporal RBAC model [4] and later generalized into Generalized-TRBAC [10]. GTRBAC distinguishes among various states of a role - such as *disabled*, *enabled* and *active* states - and extends the notion of RBAC events introduced in TRBAC. An event-based of TRBAC approach is particularly suitable for time-based access control requirements and for dynamic access control models [10, 11].

In this paper, we combine the event-based approach taken in GTRBAC with the Petri-net based modeling approach to develop a framework for modeling and analysis of non-temporal RBAC policies. The approach is particularly novel because of the intuitive way in which Petri-nets capture both system states and events, thus allowing state-based analysis for policy verification and assisting in deriving an event based execution model of an RBAC system in order to ensure safety. Furthermore, several formal tools and techniques are available for Petri-nets that can be utilized to carry out relevant analysis for correctness verification of specification.

An essential feature of RBAC is that it allows specification of various SoD constraints that are needed in many commercial applications [1,5]. SoD constraints aim at eliminating any possibility of users committing a fraud in a system by preventing a user from acquiring enough access privileges to commit fraud.

The paper is organized as follows. In section 2, we present relevant background on RBAC models on which we build our Petri-net framework. In section 3, we provide a classification of consistency rules. The colored Petri-net model of RBAC and the policy analysis framework are presented in section 4. In section 5, we discuss related work. Section 5 concludes the paper and provides future directions.

2. Overview of Role Based Access Control

In this section, we provide relevant background on the RBAC and GTRBAC models that we refer to in this paper. The RBAC model as proposed by Sandhu *et. al.* in [16], currently being used as the basis for the NIST RBAC model, consists of the following four basic components: a set of users `Users`, a set of roles `Roles`, a set of permissions `Permissions`, and a set of sessions `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 within an organization. A permission is an access mode that can be exercised on a particular object in the system. A session relates a user to possibly many roles. When a user logs in the system the user establishes a session by activating a set of enabled role that the user is entitled to activate at that time. If the activation request is satisfied, the user issuing the request obtains all the permissions associated with requested role. On `Roles`, a hierarchy is defined, denoted by $\geq$. If $r_i \geq r_j$, r_i, $r_j \in$ `Roles` then r_i inherits the permissions of r_j. In such a case, r_i is a senior role and r_j a junior role.

The RBAC model does not explicitly model different states of a role and hence does not capture various events that are typical of an RBAC system. Such event based approach was used by Bertino *et. al.* in TRBAC model [4] and later extended by Joshi *et. al.* in GTRBAC [10], primarily to capture the different transitional actions needed in the context of temporal constraints. The GTRBAC model provides a temporal framework for specifying an extensive set of temporal constraints and uses a language-based framework [10]. GTRBAC allows various types of temporal constraints such as *temporal constraints on role enabling/disabling*, *temporal constraints on user-role and role-permission assignments/de-assignments*, *role activation-time constraints, etc.* These constraints are useful in capturing the dynamic behavior of systems that employ RBAC.

Fig. 1 highlights four key components of GTRBAC model that include user-role assignment/de-assignment, role-permission assignment/de-assignment, role enabling/disabling, and role activation/deactivation. The latter two events allows one to define fine-grained access constraints based on system events as well as states. Such events, in particular, are useful in describing various dependency constraints. For instance, a role can be enabled only if some other roles are enabled, defining a precedence relation between them.

In this paper, we use an event based approach to model RBAC. This event based RBAC model corresponds to a selected set of temporal constraints of GTRBAC. The motivations for this are two fold:

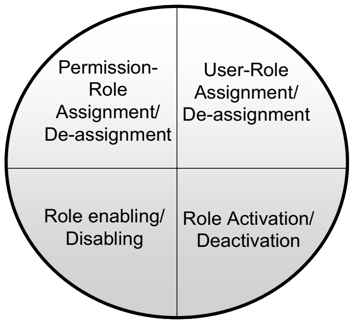

Permission-Role Assignment/ De-assignment
User-Role Assignment/ De-assignment
Role enabling/ Disabling
Role Activation/ Deactivation

Figure 1. Components corresponding to event set of GTRBAC

1. Such an event-based realization of traditional RBAC system allows capturing the dynamic properties of the system that can be used to verify the correctness of an RBAC specification.
2. Our future goal is to extend the proposed Petri-net modeling framework to model the GTRBAC system and then, to develop techniques for validating and verifying the correctness properties of GTRBAC Policies pertaining to the four components of Fig. 1.

3. A Verification Model for RBAC

Our main objective in this paper is to model RBAC using a Petri net based framework and then use this framework to verify the correctness of the underlying security policies.

3.1. Policy Considerations in RBAC

A policy is a set of rules that defines the expected behavior of the system employing that policy. The system is said to be in conformance with the underlying policy if every state of the system can be deduce from the set of rules/axioms comprising the policy. An inconsistent state or erratic system behavior can be attributed to a potential flaw in the policy specification. This flaw may be because of inconsistency in the policy itself or because of incompleteness. An inconsistent policy is the one in which two or more rules from a given set of rules comprising the policy contradict each other. Incompleteness implies that the given set of rules defining the policy is not sufficient to capture all states of the system. In this context, security verification can be stated as the process of proving that the properties or rules specified in a security policy are enforced in the information system.

Gavrila *et. al.* [7] state a set of consistency rules for information systems employing RBAC as an access control mechanism. These rules are defined as consistency rules because the information system is expected to satisfy these rules in all possible states it may take. These consistency rules although specify most of the constraints for the traditional RBAC model [7], do not capture the constraints required in modeling event-based systems. For example precedence and dependency constraints cannot be modeled using the consistency rules specified in [7]. For an RBAC system constraints can be grouped into following classes: 1) cardinality constraints, 2) separation of duties (SoD) constraints, 3) inheritance constraints, and 4) precedence and dependency constraints.

Cardinality, inheritance and SoD constraints are addressed in literature for traditional RBAC [7], however these approaches are primarily static in nature and do not take into account various authorization related RBAC events allowed in a system non-deterministically. These events include user-role assignment enabling/disabling of a role, and activation/deactivation of a role as described in [10]. A SoD constraint in an event-based environment may prohibit two conflicting roles to be enabled at the same time, or inhibit two conflicting users of some role to activate that role concurrently. Similarly, inheritance and cardinality constraints have new semantics in this (event-based) environment that may not be captured by static approaches. Precedence and dependency conditions are required to model the relative ordering of events. The following two examples describe situations where precedence and dependency constraints are required:

1. A junior employee of an office is allowed to activate the Junior_Employee role in the system *only if* the manager of the office has activated the Manager role. This condition can be modeled by the precedence constraint.
2. A trainee doctor is authorized to activate his/her role only in presence of a senior doctor. In this case the senior doctor cannot deactivate his/her role if there is an active trainee doctor role. This example represents a dependency constraint.

In [17], a set of consistency rules are proposed which are major extensions of consistency rules defined by Gavrila in [7]. This extended set of rules allows modeling of various constraints of RBAC with an event-based approach. These set of rules mainly cover the cardinality, inheritance, SoD and precedence and dependency constraints. In the context of these consistency rules, the correctness of a system state is verified against these rules. Due to space limitations, we do not list all the consistency rules, however, for clarity in presentation, we briefly describe a few of them; this will also facilitate the reader in understanding the examples given in section 4.2.

a. *User activation cardinality constraint*: The number of roles activated by any user u does not exceed the maximum number of roles the user is entitled to activate at any time.

$$\forall u \in USERS,\ |active\_roleset(u)| \le activation\_user\_card(u).$$

Where, *active_roleset(u)* describe the set of roles currently being activated by user u, and *activation_user_card(u)* gives the total number of roles the user is authorized to activate in concurrent sessions.

b. *Activation time user-based SoD constraint*: Role r cannot be concurrently activated by users u_1 and u_2, if they are activation time conflicting users for role r. Formally:

$$\forall r \in ROLES, \forall u_1, u_2 \in conflict\_user\_activeset(r)$$
$$\heartsuit\ r \notin active\_roleset(u_1) \bigcap active\_roleset(u_2)$$

Where, *conflict_user_activeset(r)* returns the set of activation time conflicting users for role r.

c. *Activation time dependency constraint:* A role r_z having an activation time dependency on role r_y can be activated by user u only if role r_y is in the active role set of some user u'. Furthermore, role r_y cannot be deactivated if role r_z is in active role set of any user. Formally:

$$r_z \in dep\_activeset(r_y)\ \heartsuit\ [r_z \in active\_roleset(u) \rightarrow r_y \in \bigcup_{u' \in USERS} active\_roleset(u')]$$

Where, *dep_activeset(r_y)* gives a set of roles that have activation time dependency on role r_y. The second example, given above, describes the activation time dependency between the trainee doctor and senior doctor. The above constraints give a synopsis of a comprehensive set of constraints described in [17].

3.2. Colored Petri-Net model of RBAC

In this section, we present a Colored-Petri-net (CPN) based framework to model RBAC. We first present a brief background on CPNs followed by the detailed description of the RBAC components and its CPN representation.

CPN Formulation of RBAC: *A CPN* [8] *is a tuple* CP = (Σ, *P, T, A, N, C, G, E, I*), *where*:

a. Σ is a finite set of non-empty types, called color sets;

b. P is a finite set of places;

c. *T is a finite set of transitions;*

d. *$A = NA \cup RA \cup IA$ is a finite set of arcs such that: $P \cap T = P \cap A = T \cap A = \emptyset$; where NA is a set of Normal Arcs, RA is a set of Read Arcs and IA is a set of Inhibitor Arcs.*

e. *N is a node function. $N:A \rightarrow P \times T \cup T \times P$.*

f. *C: is a color function. $C:P \rightarrow \Sigma$.*

g. *G is a guard function. It is defined from T into expressions such that:*
$\forall t \in T$: [Type(G(t)) = B and Type(Var(G (t)) $\subseteq \Sigma$].

h. *E is an arc expression function. It is defined from A into expressions such that:*
$a \in A$: [Type(E(a)) = $E(G(a))_{MS}$ and Type(Var(E(a)) $\subseteq$ Σ]. Here p(a) is the place of N(a).

i. *I is an initialization function. It is defined from P into closed expression such that:*
$\forall p \in P$: [Type(I(p)) = $C(p(a))_{MS}$].

In the following, we elaborate the above elements of CPN within the context of RBAC.

Color set Σ: For the RBAC formulation, the elements of the color set Σ with the corresponding data type are listed below.

Color USER = integer, Color ROLE = integer.
Color SESSION = integer.
Color COMMAND = {assign, de-assign, enable, disable, activate, deactivate}
Color UR = product USER * ROLE * ROLE; Color URS = product USER * ROLE * SESSION;
Color CMD = product COMMAND * USER * ROLE * SESSION.

Based on the above set of colors, following tokens are defined for RBAC mode:

- *User token*: <*u*>::color USER
- *Role token*: <*r*>:: color ROLE
- *User-role assignment token*: <*u,r,r'*>::color UR.
- *User-role activation token*: <*u,r,s*>::color URS.
- *Command token*: <*cmd, u, r, s*>:: color CMD.

Places P: Following CPN places are used to capture the state information for RBAC modeling:

1. *Event token generator* (ETG): This place stores command tokens for user-role assignment and de-assignment, role enabling and disabling, and role activation and deactivation. For any transition to get enabled, there must be a corresponding token in the place ETG. In this sense, this place act as a transition firing controller that helps in analyzing all possible system states against a given command list.
2. *Disabled Roles* (DR): This place can only store role tokens (C(DR) = ROLE). A token <r_y> in this place implies that role r_y is in disable state.
3. *Enabled Roles* (ER). This place can only store role tokens (C(ER) = ROLE). A token <r_y> in ER place implies that role r_y is in enable state.
4. *User Role Assignment/Authorization* (UR). This place contains tokens of color UR (C(UR) = UR). A token <u,r_y,r_x> in this place means that user u is authorized for role r_y. This authorization can be as a result of direct assignment of role r_y to user u ($r_x = r_y$), or because of assignment of role r_x to user u such that r_x inherits r_y ($r_x \geq r_y$ and $r_x \neq r_y$).
5. *User Role Session activation* (URS). This place stores tokens of color URS. . Each <*u,r,s*> token stored in this place implies that session s is being activated by user u who has assumed role r.
6. *Role Cardinality* (RC): This place contains role tokens only (C(RC) = ROLE). It enforces assignment time role cardinality constraint, i.e., limits the number of users authorized for a given role. If there are n_i number of <r_y> tokens in place RC then at most n_i number of users can be authorized for r_y.
7. *User Cardinality* (UC): This place contains user tokens only (C(UC) = USER). It enforces assignment time user cardinality constraint, i.e., defines an upper bound on the number of authorized roles for a given user.
8. *Role Activation cardinality* (RAC): Place RAC stores token of type ROLE (C(RAC) = ROLE). It enforces activation time role cardinality constraint, i.e., limits the number of concurrent activations of a given role. If there are n_i number of <r_y> tokens present at RC, then at most n_i more copies of role r_y can be activated concurrently.
9. *User Activation cardinality* (UAC): Place UAC stores token of type USER (C(UAC) = USER). This place enforces activation time user cardinality constraint, i.e., limits the number of concurrent activations of roles for a given user. If there are m_j number of <u_z> tokens present in the place UAC, then user u_z can make m_j activations concurrently. These activations may involve activating same role multiple times or multiple roles for any number of times provided that the total number of such concurrent activation of roles by user u_z do not exceed the user activation cardinality m_j.

Arcs and arc expression: Arc, arc expressions and guard functions are used to model constraints including cardinality, SoD, inheritance, precedence and dependency constraints as discussed in section 3.1.

Transitions:Transitions in this framework represent all four components of Fig. 1 including user-role assignment/de-assignment, role-permission assignment/de-assignment, role enabling/disabling and role activation/deactivation. In this CPN representation, each role r_y has the following six transitions:

1. Assignr_y: assigns user $u \in$ USERS to role r_y. By virtue of this role assignment user u is authorized for all roles inherited by role r_y.
2. De-assignr_y: Cancels all the user role assignment between user u and role r_y. It also nullifies u's authorization for all junior roles that are on u's authorization list by virtue of its assignment to role r_y.
3. Enabler_y: This transition enables role r_y. Upon firing, a token r_y is inserted in place ER from DR, implying that role r_y is enabled and can be activated by a user who is authorized for role r_y. Fig. 2 shows enable transition for role r_y.
4. Disabler_y: This transition disables role r_y. Upon firing, r_y is removed from place ER and inserted in place DR, implying that role r_y can not be activated by any user.
5. Activater_y: This transition establishes an active session between user u and role r_y.
6. Deactivater_y : This transition deactivates role r_y from the an active session between user u and role r_y.

Firing of any of the above transitions changes the state of the system. A transition can fire anytime after its enabling. Enabling of a transition implies that all the constraints associated with the event, the transition is modeling, are satisfied. For brevity in presentation, we list the enabling/firing rules for assignment of roles only. For the remaining transitions, interested readers are referred to [17].

***Enabling/firing rules of transition assignr_y*:** This transition upon firing inserts the set of tokens $\{<u, r_x, r_y>: r_y \geq r_x\}$ in the place UR which implies that the role r_y is assigned to user u, and user u is authorized for role r_y and all roles r_x junior to role r_y. The transition *assignr_y* and its connecting places are shown in Fig. 2 and the corresponding arc expressions and guard functions are listed in Table 1.

This transition gets enabled if the following constraints are satisfied:

- There is a token $<assign, u_z, r_y>$ in place ETG implying that role r_y be assigned to user u_z.
- Assignment time role cardinality constraint specified by the arc expression E3: $r_y+r_{y1}+\ldots r_{yn}$, where, all $r_{yi} < r_y$ and $i \leq n$, is satisfied. Alternatively, tokens $r_y,r_{y1},\ldots,r_{yn}$ are present in place RC.
- Assignment time user cardinality constraint specified by the arc expression E4: $(n+1)u_z$ is satisfied, where n is the number of roles that are junior to r_y in the role hierarchy.
- Assignment time conflicting roles constraint specified by the arc expression (inhibitor) E6: $<u_z, r_c, any\ r>$ and the transition guard function G2: *conflict_role_assign*(r_y r_c) is satisfied. That is Place UR does not contain any token $<u_z, r_c, any\ r>$ for which the above guard function evaluates true.
- Assignment time conflicting users constraint specified by the arc expression (inhibitor) E7:$<u_c, r_z, any\ r>$ and the transition guard function G3: *conflict_user_Assign*(r_y,u_z,u_c) is satisfied. That is Place UR does not contain any token $<u_c, r_z, any\ r>$ for which the above guard function is true.
- Place UR does not contain any token $<u_z, r_y, any\ r>$. This is specified by the inhibitor arc expression E3 and guard function G1.
- The following two constraints are optional and are only defined for roles which have assignment time precedence constraint(s). Assignment time precedence constraint can be of two types: same user assignment constraint and any user assignment constraint. A given role may have one, both or none of these precedence constraints.
  1. Same user assignment constraint requires that a user u_z can be assigned role r_y only if role $r' \in$ *prec_su_assignset*(r_y) is assigned to user u_z. This constraint is specified by the read arc expression E8 and the transition guard function G4: *prec_su_assign*(r_y,{r}).
  2. Any user assignment constraint requires that a user u_z can be assigned role r_y only if role r''.$\in$ *prec_au_assignset*(r_y) is assigned to some user. This constraint is specified by the read arc expression E9 and the transition guard function G5: *prec_au_assign*(r_y,{r}).

The guard functions and arc expressions corresponding to Fig. 2 are listed in Table 1. Fig. 2 shows a CPN representation of user to role assignment/de-assignment with transition *assignr_y* and d*eassignr_y* modeling the assignment and de-assignement events for role r_y respectively. The set of places in Fig. 2 shows the current state of the system in terms of number of users assigned to role r_y, the number of active sessions associated with role r_y etc. The arcs and guard expressions specify the assignment time cardinality, Sod, precedence and dependency constraints.

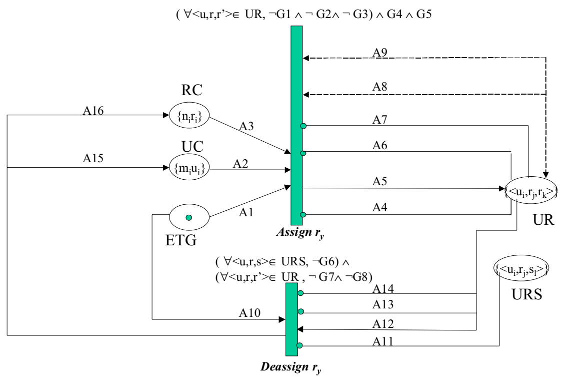


Figure 2. CPN construction for user-role assignment/de-assignment.

Based on the discussions in Section 3, we formalize the notion of a consistent RBAC state in the following definition. This notion of consistency is used to capture the dynamic property of the CPN in Theorem 1.

Definition: The state of an RBAC system is said to be consistent if all the cardinality, inheritance, SoD, precedence and dependency constraints are satisfied in that state.

Theorem 1: Given a P_{RBAC} (CPN structure for RBAC) structure with an initial consistent state M_0, all states M, reachable from M_0 are consistent.

The proof for this theorem requires enumerating all the consistency rules and is omitted because of the space limitation. Interested readers are referred to [17].

3.3. Reachability analysis for consistency verification of RBAC policy

In this section, we elaborate the process of verifying the consistency of RBAC policy constraints. The verification is based on the reachability analysis of CPN proposed in the previous section. We use occurrence graph method [8] to enumerate all reachable states of a system employing a given RBAC policy. Lemma 1 states that our Petri net representation of RBAC system is bounded and so its occurrence graph will have finite number of nodes. However, the exhaustive nature of this method implies that the problem of verifying that a given state is reachable from some initial state takes exponential space and time [13]. Since policy verification can be done offline and is performed before the deployment of actual system, so complexity is not a major issue in using this proposed Petri-net approach.

The following example illustrates the use of occurrence graph for security policy verification.

Example: Consider three roles r_0, r_1, and r_2 and a single user u_0. Let r_1 be junior to r_0 ($r_1 \leq r_0$ and $r_1 \neq r_0$). Also let r_1 and r_2 be assignment time conflicting roles, i.e., r_1 and r_2 cannot be assigned to the same user implying that roles r_1 and r_2 cannot be activated by the same user concurrently. Fig. 2 shows the sub-graph of the occurrence graph of the RBAC system. In this sub-graph all roles (r_0, r_1, and r_2) are considered to be in enable state and the SoD constraint is only defined between roles r_1, and r_2. Note that in Fig. 3, user u_0 who is assigned role r_0 and r_2 is able to activate roles r_1 and r_2 concurrently. This is a violation of the SoD constraint defined on these two roles. This inconsistency arises because of the fact that in the original specification roles r_0 and r_2 do not have any SoD constraint while r_1 and r_2 are assignment time conflicting roles. As r_0 is superior to role r_1 and any user assigned to role r_0 is authorized for role r_1, the SoD constraint must also be defined between roles r_0 and r_2.

Table 1. Arc and guard expressions

Arc Expression E_i for corresponding arc A_i, where			
E1	<assign,u_Z,r_y>	E9	{<any u , r'', any r>}
E2	(n+1)u_Z	E10	<de-assign,u_Z,r_y>
E3	r_y+r_{y1}+…r_{yn} (r_{y_i}<r_y. for all	E11	<u_Z r_k, any r>
E4	<u_Z,r_h>	E12	<u_Z,r_y, r_y> + <u_Z, r_{y1},r_y>+….+<u_Z,
E5	<u_Z,r_y,r_y> + <u_Z, r_{y1},r_y>+….+<u_Z, r_{yn},r_y>	E13	<u_Z,r_i, r_{dsu}> ($r_i \leq r_{dsu}$)
E6	<u_Z r_C, any r> A7: <u_C r_Z, any r>	E14	<any u, r_j, r_{dau}> ($r_j \leq r_{dau}$)
E7	<u_C r_Z, any r>	E15	(n+1)u_Z
E8	{<u_Z , r', any r>}	E16	r_y+r_{y1}+…r_{yn} (r_{y_i}<r_y. for all 1≤i≤n)
Guard functions associated with transition *Assignry* and *De-ssignry*			
G1	:($r_h \leq r_y$) ∨ ($r_y \leq r_h$)	G5	prec_au_assign(r_y r',})
G2	conflict_role_assign(r_y, r_c)	G6	$r_k \leq r_y$
G3	conflict_user_assign(r_y, u_Z, u_C)	G7	dep_su_assign(r_{dsu}, r_y)
G4	prec_su_assign(r_y,{r'})	G8	dep_au_assign(r_{dau}, r_y)

4. Related work

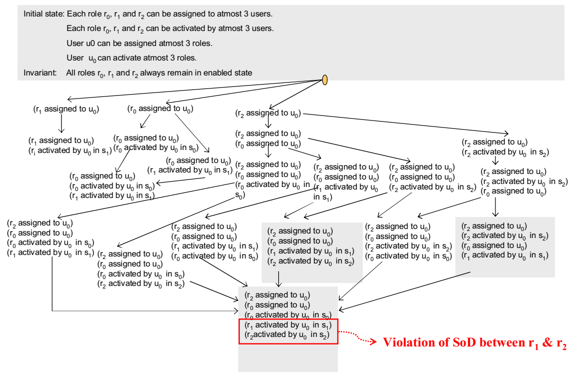
RBAC models have been proposed and extended by several researchers [14, 16, 6], and the efforts in this direction have resulted in the proposal of a standard model – the NIST RBAC model [6]. Need for supporting constraints in an RBAC model has been addressed by many researchers. In particular, the attention has been in supporting *separation of duties* (SoD) constraints [1, 7]. In [1], Ahn *et. al.* propose *RCL2000* – a role based constraint specification language. Bertino *et. al.* have proposed a logic based constraint specification language that can be used to specify constraint on roles and users and their assignments to workflow tasks [3].

Various work address policy analysis and verification issues related to RBAC models. Nyanchama *et. al.* [14] present a graph based RBAC model, where graphs are used to mainly represent hierarchies of users, roles and permissions. It does not address the issue of policy verification. Koch *et. al.* [12] present a graph transformation based formalism for RBAC model and model the SoD constraints identified in the literature. The model provides a graph transformation based specification of static and dynamic consistency conditions of RBAC.

Ahmed *et. al.* [18] have proposed a model checking based methodology for verification of the security requirements of computer supported cooperative work (CSCW) systems. They use role-based policies to specify coordination and security constraints of the CSCW systems. In this sense our work is similar to [18].

5. Conclusion

We have presented a colored Petri-net based framework for verifying the consistency of RBAC policies. The Petri-net model can capture all the cardinality and separation of duty constraints that have been previously identified in the literature. Moreover, the model also allows specification of the precedence and dependency constraints that we introduce in this paper. We use the Petri-net reachability analysis technique for RBAC policy verification. A set of consistency rules is used as the basis for detecting undesirable states representing erratic behavior of the system due to the flaws in policy specification. The analysis framework can be used by security administrators to generate correct specification iteratively.



Initial state: Each role r_0, r_1 and r_2 can be assigned to atmost 3 users.
Each role r_0, r_1 and r_2 can be activated by atmost 3 users.
User u0 can be assigned atmost 3 roles.
User u_0 can activate atmost 3 roles.
Invariant: All roles r_0, r_1 and r_2 always remain in enabled state
(r_1 assigned to u_0)
(r_0 assigned to u_0)
(r_2 assigned to u_0)
(r_1 activated by u_0 in s_1)
(r_0 activated by u_0 in s_0)
(r_2 activated by u_0 in s_2)
(r_1 activated by u_0 in s_1)
Violation of SoD between r_1 & r_2

Figure 3. Occurrence graph of the example with incomplete specification

6. References

[1] G. Ahn, R. Sandhu, "Role-Based Authorization Constraints Specification", *ACM TISSEC*, Vol. 3, No. 4, Nov. 2000.

[2] J. Barkley, A. Cincotta, D. Ferraiolo, S. Gavrila, and D.R. Kuhn, "Role Based Access Control for the World Wide Web," in *proc. of the 20th National Information System Security Conference*, 1997.

[3] E. Bertino, E. Ferrari, and V. Atluri, "The Specification and Enforcement of Authorization Constraints in Workflow Management Systems," *ACM TISSEC*, Vol. 2, No. 1, Feb. 99, pp. 65-104.

[4] E. Bertino, P. A. Bonatti, E. Ferrari, "TRBAC: A Temporal Role-based Access Control Model," *ACM TISSEC*, Vol. 4, No. 3, Aug. 2001, pp. 191-233.

[5] D. F. Ferraiolo, D. M. Gilbert, N Lynch, "An examination of Federal and Commercial Access Control Policy Needs," in *Proc. of the National Computer Security Conference*, Baltimore, MD, Sept. 1993, pp. 107-116.

[6] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, R. Chandramouli, "Proposed NIST Standard for Role-based Access Control,"*ACM TISSEC,* Vol. 4, No. 3, Aug 2001.

[7] S. I. Gavrila , J. F. Barkley, "Formal Specification for Role Based Access Control User/role and Role/role Relationship Management," in *Proc. of the 3rd ACM Workshop on Role-Based Access Control*, Oct., 1998, pp. 81-90.

[8] K. Jensen, Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use Volume 1, *Springer Verlag*, 1997.

[9] J. B. D. Joshi, W. G. Aref, A. Ghafoor and E. H. Spafford, "Security Models for Web-based Applications," *Communications of the ACM*, Vol. 44, No. 2, Feb. 2001, pp. 38-72.

[10] J. B. D. Joshi, E. Bertino, U. Latif, A. Ghafoor, "A Generalized Temporal Role Based Access Control ," *IEEE Transaction on Knowledge and Data Engineering*, Vol. 17, No. 1, Jan 2005, pp. 4–23.

[11] J. B. D. Joshi, E. Bertino, A. Ghafoor, "Temporal Hierarchies and Inheritance Semantics for GTRBAC," in *Proc. of the* 7th *ACM Symposium on Access Control Models and Technologies*, June 2002, pp. 74-83.

[12] M. Koch, L. V. Mancini, F. Parisi-Presicce, "A Graph-based Formalism for RBAC," *ACM Transactions on Information and System Security,* Vol. 5, No. 3, Aug. 2002, pp. 332 – 365.

[13] T. Murata, "Petri Nets: Properties, Analysis and Application", *Proceedings of IEEE*, Vol. 77, No. 4, 1989, pp. 541-580.

[14] M. Nyanchama and S. Osborn, "The Role Graph Model and Conflict of Interest," *ACM TISSEC*, Vol. 2 No. 1, 1999, pp. 3-33.

[15] S. Osborn, R. Sandhu, Q. Munawer, "Configuring Role-based Access Control to Enforce Mandatory and Discretionary Access Control Policies," *ACM* TISSEC Vol. 3, No. 2, May 2000, pp. 85-106.

[16] R. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, "Role-Based Access Control Models," *IEEE Computer* Vol. 29 No. 2, 1996, pp. 38-47.

[17] B. Shafiq, J. Joshi, A. Ghafoor, "A Petri-net Model for Verification and Validation of Role-based Access Control Model", *CERIAS TR 2002-33*, Purdue University.

[18] T. Ahmed and A. R. Tripathi, "Static Verification of Security Requirements in Role Based CSCW Systems," in *Proc. of the 8th ACM Symposium on Access Control Models and Technologies*, Jun. 2003, pp. 196-203.