CERIAS Tech Report 2005-08

AN OPTIMAL CONFLICT RESOLUTION STRATEGY FOR EVENT-DRIVEN ROLE BASED ACCESS CONTROL POLICIES

by Basit Shaiq, Elisa Bertino, and Arif Ghafoor

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

An Optimal Conflict Resolution Strategy for Event-Driven Role Based Access Control Policies

Basit Shafiq[1], Elisa Bertino[2], and Arif Ghafoor[1]
[1] School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907
{shafiq, ghafoor}@ecn.purdue.edu
[2] CERIAS and Department of Computer Sciences, Purdue University, West Lafayette, IN 47907
bertino@cerias.purdue.edu

Abstract

Role based access control (RBAC) has generated great interest in the security community for its inherent richness and flexibility in modeling a wide range of access control policies. Any comprehensive access control model such as RBAC requires verification tools to support consistency analysis and identify possible policy conflicts. These conflicts, if remain undetected and unresolved, expose the underlying system to numerous vulnerabilities and security risks. In this paper, we propose a verification framework for detection and resolution of inconsistencies and conflicts in event-driven RBAC policies. The framework uses an integer programming based approach for optimal resolution of policy conflicts. The proposed approach is generic and can be tuned to a variety of optimality measures.

Portions of this work were supported by Grant IIS-0209111 from the National Science Foundation and by sponsors of the Center for Education and Research in Information Assurance and Security.

1 Introduction

Role based access control (RBAC) has generated great interest in the security community for its inherent richness and flexibility in modeling a wide range of access control policies [11, 17, 19]. Several beneficial features such as policy neutrality, support for least privilege, efficient access management, are associated with RBAC models [19, 13]. 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 policies [10]. Various extensions to RBAC have been proposed to incorporate the temporal and event-based semantics required in many business processing and workflow based applications, including: e-commerce, digital government, supply-chain management, health-care, distance learning, and many others [4, 21, 22, 5, 6]. Most of these emerging applications require specification and enforcement of access control policies at a very fine granularity which are difficult to implement even with the extended RBAC models, mainly because of their inefficiency in modeling permission-centric constraints. The current RBAC approaches attempt to model such constraints at the role-level which either restrict the semantics of permission-centric constraints or introduce considerable overhead by creating separate roles for the permissions associated with such constraints [12]. The latter may introduce a disparity between the abstraction of role in RBAC and the notion of role in an organizational hierarchy. Another important requirement common to many such application environments is the support for event-based access control, according to which certain roles or permissions may be automatically enable or disabled depending on the occurrence of some specified events. In this paper we propose two important extensions to current RBAC models that address those requirements. In particular, we introduce the concept dynamic permissions in RBAC. Acquisition of a dynamic permission not only requires assumption of the role to which such permission is assigned but also the satisfaction of the corresponding permission-centric constraints. In addition we enhance RBAC with supports for event-based role or permission enabling/disabling by introducing two new types of event triggers for modeling stricter form of dependencies that often occur in many task-oriented and work-flow based systems. These triggers can be used to model both role-level and permission-level dependency constraints.

Any comprehensive access control model requires tools to support consistency analysis and identify possible conflicts. The interplay of various RBAC constraints such as hierarchy, separation of duty (SoD), dependency and cardinality may introduce inconsistencies and conflicts in the underlying access control policy. These conflicts, if remain undetected and unresolved, exposes an organization to numerous vulnerabilities and risks pertaining to security and privacy of organizational data and resources. The problem of conflict detection has been extensively studied in literature in the context of rule-based systems [15, 16]. Most of the security related research work uses the reachability and model-checking based techniques developed for analyzing rule-based systems, to verify the consistency of access control policy specification [1, 2]. However, the crucial issue of conflict resolution has not been adequately addressed in literature in the context of access control policies. Typically, resolution of policy conflicts involves manual intervention of policy administrator. Incase there are multiple policy administrators, a consensus on the resolution needs to be obtained. This is a slow and ad hoc process and provides no guarantee on the quality of solution in terms of the system behavior after resolution. There are some dynamic techniques for resolution of access control policy conflicts [14, 7, 6, 13]. However, these techniques either assume a hierarchical relationship of objects and subjects, or consider authorizations to be independent. These assumptions may not hold in access control polices derived from RBAC models.

In this paper, we propose an integer programming (IP) based technique for the optimal resolution of conflicts in an event-driven RBAC policy. The event-driven RBAC model, discussed in Section 2, uses trigger based mechanism to capture the dependency and event constraints. It is important to note that even though the proposed conflict resolution technique is discussed in the context of RBAC, our results are also relevant to a large variety of existing and next generation access control models [18, 23].

The main contributions of this paper are as follows:

1. Two types of event-triggers, *strong dependence and weak dependence* triggers are introduced. The strong dependence semantics of the event triggers is a novel addition and has not been considered in existing RBAC models. These triggers can be used to model a variety of dependence and workflow based constraints. In addition various separation of duties (SoD) constraints defined in literature can be composed from these constraints and other basic RBAC SoD constraints. Another novel feature of our model is the introduction of dynamic permissions that allow specification of access constraints at the permission level.
2. An integer programming (IP) based approach is used to resolve policy conflicts in an optimal manner. Unlike other conflict resolution strategies, our approach does not assume any object-oriented hierarchy defined over objects and subjects. Moreover, the structural dependence semantics of RBAC models can be easily captured

in the proposed approach. The authorizations produced by the resulting RBAC policy are always deterministic. The proposed approach is generic and can be tuned to a variety of optimality measures such as maximizing accessibility, minimizing set of relaxed constraints, and maximizing prioritized accesses.

The remainder of this paper is organized as follows. In Section 2, we present the event-driven RBAC model. In Section 3, we describe the proposed IP-based conflict resolution technique. In Section 4, an example is provided to illustrate the effectiveness of proposed conflict resolution technique, and in Section 5, we formally prove the correctness of proposed approach. The related work is presented in Section 6. Section 7 concludes the paper and provides some future directions.

2 Event-Driven Role Based Access Control

The RBAC model [19], currently being used as the basis for the NIST RBAC model, consists of the following four basic components: a set U of *users*, a set R of *roles*, a set P of *permissions*, and a set S 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 within an organization. Permission defines the access rights 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 roles 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 the requested roles. One of the most important aspects of RBAC is the use of role hierarchies to simplify management of authorizations. The original RBAC model supports only *inheritance* or *usage* hierarchy, which allows the users of a senior role to inherit all permissions of junior roles. In order to preserve the *principle of least privilege*, RBAC model has been extended to include *activation hierarchy* which enables a user to activate one or more junior roles without activating senior roles [20]. From this point onward, we will use the notations I and A, to refer to inheritance and activation hierarchies respectively. The symbols $\underset{I}{\geq}^{*}$ and $\underset{A}{\geq}^{*}$ are used to express I, and A hierarchy relationship between two roles respectively. Accordingly, $r_i \underset{f}{\geq}^{*} r_j$, where $f \in \{I, A\}$, implies that role r_i is senior to r_j and the hierarchical relationship between them can be either *inheritance* only, or *activation* only. If role r_i is immediately senior to role r_j then the superscript * is omitted from the relation symbol $\underset{f}{\geq}$.

2.1 Dependency Constraint

The event dependency relationship semantics is incorporated in the RBAC formalism by introducing event triggers. We define two types of dependency relations, namely: *strong dependency* relation represented by the symbol $\rightarrow\rightarrow$ and *weak dependency* relation denoted by the symbol $\rightarrow$. In the following, we first define different types of event expressions and then introduce event triggers for implementing dependency constraints.

Simple Role Event Expression: A simple role event expression (SREE) is of the form *activate*(u_i, r_j), where $u_i \in$ U, $r_j \in$R, specifying that user u_i has activated role r_j. All the role-related events can be specified in terms of activation of a role by some user. In order to represent the event corresponding to enabling (disabling) of a role r_j, a special user u_{ej} (u_{dj}) is assigned to role r_j. Activation of r_j by u_{ej} (u_{dj}) corresponds to the enabling (disabling) of role r_j. To prevent simultaneous activations of r_j by u_{ej} and u_{dj} a *user-specific separation of duty* constraint (discussed in Section 2.2) is specified between users u_{ej} and u_{dj}.

Simple Permission Event Expression: A simple permission event expression (SPEE) is of the form *access*(u_i, p_k), where $u_i \in$ U, $p_k \in$P, specifying that user u_i has accessed permission p_k. Like role related events, permission related events can also be expressed by a user access to the permission. A permission can be accessed by a user if the permission is in the enabled state and the user acquiring the permission has activated the role to which the permission is assigned, or a senior role. A permission p_k assigned to role r_j becomes enabled (disabled) if it is being accessed by the special user u_{ej} (u_{dj}) of role r_j.

Event Triggers: Event triggers are used to model the dependence and ordering relationship between events. There are two types of triggers distinguishing the strong and weak dependency semantics.

Strong dependence Trigger: A strong dependence trigger is of the form: $E_1*...*E_n \rightarrow\rightarrow E$, where E_i (i = 1,..,n) and E are simple role event or simple permission event expressions and * represents logical conjunction or disjunction. Event E occurs only if the body of the trigger (left hand side of the trigger) is true.

Weak dependence Trigger: A weak dependence trigger is of the form: $E_1*...*E_n \rightarrow E$, where E_i (i = 1,..,n) and E are simple role event or simple permission event expressions and * represents logical conjunction or disjunction. In case of weak dependence, event E can occur independently of E_is; however, when the body of the trigger becomes true, event E is triggered.

2.2 Separation of Duty Constraint

Separation of duty (SoD) policies have been found to be crucial for securing many commercial and business applications. Role-based models provide a convenient way for expressing and enforcing such policies. In the event-based RBAC formalism presented in this paper, majority of the SoD constraints identified in the literature [2, 13] can be composed from the following four basic SoD constraints:

i) *Role-specific SoD*: a role-specific SoD disallows activation of conflicting roles by same user in the same session or in concurrent sessions.
ii) *Permission-specific SoD*: a permission-specific SoD prevents same user to access conflicting permissions in the same session or in concurrent sessions.
iii) *Role-level-user-specific SoD*: a role-level-user-specific SoD prohibits conflicting users of a role from assuming that role in concurrent sessions.
iv) *Permission-level-user-specific SoD*: a permission-level-user-specific SoD prevents conflicting users of a permission from accessing that permission concurrently.

2.3 Permission Access Semantics in Event-Driven RBAC

In traditional RBAC models, permissions are considered static in a sense that any user assuming a role acquires all the permissions assigned to that role. A limitation of this permission acquisition semantics is that no permission-level access constraints can be defined. For instance, permissions may have separation of duty constraints or constraints regarding the order in which permissions are accessed. Permission centric constraints can be modeled in RBAC by creating separate roles for all permissions with such constraints and defining these constraints at the role level. However, creating a separate role for each permission, introduces a disparity between the abstraction of a role in RBAC and the notion of role in an organization hierarchy. For defining constraints at the permission level, we distinguish permissions into two types: i) *static permission*, and ii) *dynamic permission*.

Static Permission: A static permission does not have any permission level constraint and becomes enabled when the corresponding role to which it is assigned is enabled. For a user u_i to access a static permission p_s assigned to role r_j, user u_i needs to activate at least one role in the set of roles $\mathrm{R}_{\mathrm{ps}} = \left\{ r : \left(r = r_j \right) \vee \left(r \underset{I}{\geq^*} r_j \right) \right\}$.

Dynamic Permission: Permission level constraints can be defined on dynamic permissions. For a user u_i to access the dynamic permission p_k assigned to role r_j, following conditions must hold:

i) p_k is in the enabled state.
ii) u_i has activated at least one role in the set of role R_{pk}. $\mathrm{R}_{\mathrm{pk}} = \left\{ r : \left(r = r_j \right) \vee \left(r \underset{I}{\geq^*} r_j \right) \vee \left(r \underset{A}{\geq^*} r_j \right) \right\}$
iii) u_i has not accessed any permission p_l that conflicts with p_k in the same session or in concurrent sessions.
iv) p_k is not being accessed by any user u_c that conflicts with user u_i for permission p_k (permission-level user-specific SoD).

Note that unlike static permission, a user can also acquire a dynamic permission p_k assigned to role r_j by only activating a role senior to r_j in the *A-hierarchy* semantics.

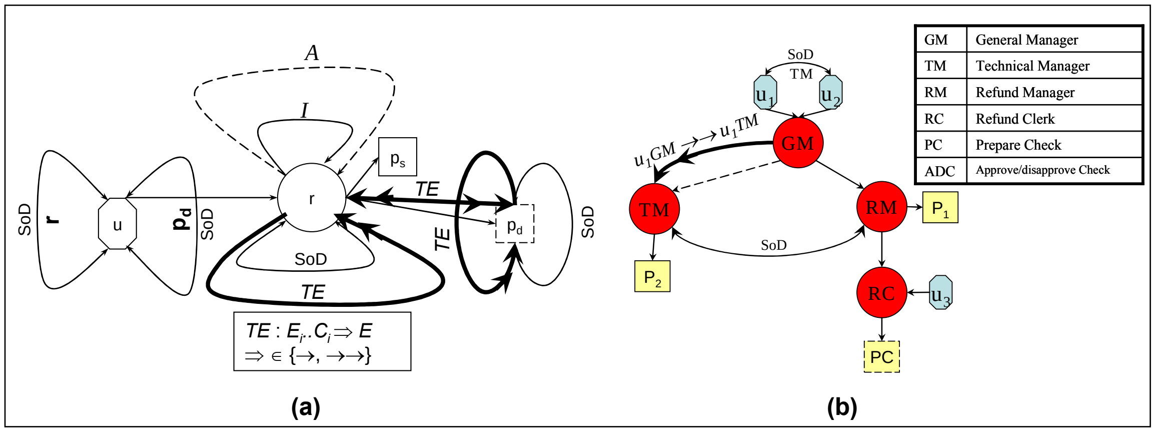


Fig. 1 (a) Type graph for event-driven RBAC. (b) Example of an event-driven RBAC policy

2.4 Graph-Based Specification Model for Event-Driven RBAC

A graph based formalism can be used to specify the RBAC policy and associated event-based constraints. In the graph based model, user, roles and permissions are represented as nodes and the edges of the graph describe the association and constraints between different nodes. Nodes in a RBAC graph cannot be connected in an arbitrary manner. The type graph shown in Fig. 1(a) defines all possible edges that may exist between different nodes. An edge between a user node *u* and a role node *r* indicates that role *r* is assigned to user *u*. The hierarchy relationship between roles are modeled by self-edges labeled with *I* and *A*. In the type graph, *I-hierarchy* and *A-hierarchy* are represented by *solid* and *dashed* edges respectively. There can be edges between role and permission nodes. A permission is a pair (*object*, *access mode*), which specifies which objects can be accessed and in which mode (read, write, execute, approve, etc). In order to distinguish between the two permission types, *static* permissions are represented as a solid square and dynamic permissions as *dashed* square. The graph model also supports specification of various separation of duty (SoD) constraints. A Role-specific or permission specific SoD constraint between two roles or permissions is represented by a double headed arrow between corresponding nodes. To represent conflicting users u_i and u_j for role r_k (permission p_l), a double headed edge with a label r_k (p_l) is drawn between the user nodes u_i and u_j. The label r_k (p_l) specifies that the corresponding users are conflicting for role r_k (permission p_l) and cannot access r_k (p_l) simultaneously. An event trigger in the graph-based model is represented by a bold edge labeled with the trigger expression TE. The trigger edge originates from all role or permission nodes, listed in the body of the trigger, and terminates at the role or permission node defined in the head of the trigger. A single trigger may have more than one edge because of the presence of multiple terms in the body of the trigger; however, all such edges are labeled with the same trigger expression.

Fig. 1(b) shows an instance of the event driven RBAC policy graph consisting of four roles: General Manager (GM), Technical Manager (TM), Refund Manager (RM), and Refund Clerk (RC). The policy graph has three users: u_1, u_2, and u_3, and three permissions: P_1, P_2, and PC (prepare check). Users u_1 and u_2 are assigned the role GM and u_3 is assigned the role RC. The static permissions P_1 and P_2 are assigned to role RM and TM respectively, while the dynamic permission PC is assigned to role RC. The inheritance hierarchy relationship $GM \underset{I}{\geq} RM \underset{I}{\geq} RC$ enables user u_1 and u_2 to acquire the permissions of junior roles RM and RC by assuming the senior role GM. On the contrary, assumption of role GM does not entitle $u_1$or u_2 to inherit all the permissions of role TM without activating TM. Role TM is junior to GM in the A-hierarchy semantics, $GM \underset{A}{\geq} TM$. This *A*-hierarchy relationship permits u_1 and u_2 to activate role TM. A role-specific SoD constraint is defined between role TM and RM, implying that these roles cannot be accessed by same user simultaneously. The double headed arrow between the user nodes u_1 and u_2 defines the user specific SoD constraint between these two users for role TM, meaning that users u_1 and u_2 cannot activate role TM concurrently. The strong dependency constraint, $u_{1GM} \rightarrow\rightarrow u_{1TM}$, defined between the role nodes GM and TM, implies that user u_1 can activate the role TM only if u_1 has activated the role GM.

3 Consistency Analysis and Conflict Resolution

Formal specification of security requirements and access constraints is the first step in designing access control policy. The next step is to identify and resolve any policy conflicts. Conflicts in an event-driven RBAC policy may not be explicit and may occur because of the interplay between various constraints embedded in the policy. For instance, the RBAC policy shown in Fig. 1(b) becomes inconsistent when user u_1 assumes the role of GM. The strong dependency trigger defined between GM and TM constrains u_1 to either assume both roles GM and TM or none at all. By activating the role of GM, u_1 also acquires permission over the junior role RM because of the inheritance hierarchy relationship. This leads to a violation of the role-specific SoD constraint defined between TM and RM. One of the following solutions resolves this conflict: i) Disallow u_1 to activate the role GM. ii) Drop the strong dependence constraint between GM and TM. iii) Relax the SoD constraint between TM and RM. iv) Remove the hierarchical relationship between GM and RM. The first one restricts the accessibility of u_1 to GM and all other junior roles and may cause deadlock if there is no other user to access these roles. The last three solutions correspond to conflict resolution by constraint relaxation. However, relaxing a constraint in an arbitrary manner may produce significant deviations from the original policy and may not yield optimal resolution.

In the following, we describe a 0-1 integer programming (IP) based approach that resolves policy conflicts in an optimal manner. The authorizations produced by the resulting policy are always deterministic. The proposed approach primarily uses constraint relaxation strategy. All the constraints that can be relaxed are assigned a weight according to their importance and a conflict free policy is generated by selecting all the non-conflicting constraints that yield an optimal value of the objective function. The proposed approach is generic in the sense that it can work for a variety of optimality measures such as maximizing accessibility, minimizing the set of relaxed constraints, and maximizing prioritized accesses and constraints. Changing the optimality measure in our formulation only requires changing the weight in the objective function.

3.1 IP Formulation of Event-based RBAC Policy

The event-driven RBAC policy can be formulated as the following 0–1 integer programming problem.

$$\text{maximize } c_1^T a + c_2^T u_r + c_3^T u_{pd}$$

$$\text{subject to } A[a \quad u_r \quad u_{pd} \quad pu_r \quad pu_{pd}] \le b$$

$$\forall a_i \in a,\ a_i = 0 \text{ or } 1,\ \forall u_{ir_j} \in u_r,\ u_{ir_j} = 0 \text{ or } 1,\ \forall u_{ip_j} \in u_{pd},\ u_{ip_j} = 0 \text{ or } 1,$$

$$\forall pu_{ir_j} \in pu_r,\ pu_{ir_j} = 0 \text{ or } 1,\ \forall pu_{ip_j} \in pu_{pd},\ u_{ip_j} = 0 \text{ or } 1$$

Where, $c = [c_1\ c_2\ c_3]$ is the cost function defining the optimality criterion. 'a' is a constraint vector whose elements correspond to the policy constraints including role assignment, role-hierarchy, SoD, and event dependency constraints. 'u_r' is a vector defining the user-role authorizations and 'u_{pd}' is a vector defining the user-permission authorizations. The vectors 'pu_r' and 'pu_{pd}' define the role and permission authorizations for proxy users. Proxy users (discussed in next section) are not the actual users specified in the original access control policy and are included in the IP formulation to create a problem instance in which all the constraints can be evaluated. In the IP formulation of an RBAC policy, all the constraints are defined using algebraic equations. The elements of matrix 'A' correspond to the coefficients of terms used in the equations/inequalities defining the constraints. All the variables used in above IP formulation are binary variables, i.e., they can only take a value of '0' or '1'.

In the solution to the IP problem, if the value of a constraint variable 'a_i' equals one then the corresponding constraint is retained in the final policy; otherwise, it is dropped. The user role authorization variable 'u_{ir_j}' defines the authorization of user u_i over role r_j. If $u_{ir_j} = 0$ is specified as an IP constraint, then user u_i has no authorization over role r_j and cannot access r_j by any means. Similarly, the user permission variable u_{ip_j} defines the authorization of user u_i over the dynamic permission p_j. Note that the user-permission authorization variable is only defined for dynamic permissions. Static permissions are automatically acquired by a user when the user accesses the role to which such static permissions are assigned. For instance, a user u_i by activating role r_j acquires all the static permissions of r_j and all the static permissions of roles that are junior to r_j in the *I-hierarchy* sense.

3.1.1 IP constraint Transformation Rules

The rules for transforming the policy constraints into IP constraints are listed in Tables II – IV. The predicates and functions used in these transformation rules are described in Table I. The transformation rules are grouped into following categories: hierarchy and assignment, enabling, SoD and dependency triggers. Rules for each of these categories are separately defined for actual users specified in the original event-driven RBAC policy and for proxy users created to evaluate all possible authorizations and constraints in the underlying IP problem.

Table II lists the rules for the users defined in the original policy. Rules 1-4 ensure that in any feasible solution of the IP, if a user accesses a role or permission then the user should have proper authorization for the role or permission being accessed. A user *u* is authorized for a role *r* if either *u* is assigned to *r* or *u* is assigned to senior role *r'* such that there is an access path from *r*' to *r*. For accessing a dynamic permission, a valid authorization is required for the role to which such permission is assigned. The enabling rules (5 and 6) imply that a role or permission can only be accessed in the enabled state. Rules 7 and 8 represent the four basic SoD constraints in mathematical form using the corresponding user-role and user-permission binary decision variables. Rules 9 and 10 defines the event trigger dependency implying that whenever the body of the event trigger becomes true, the event listed in the head of the corresponding dependency constraint is triggered. For a strong dependency constraint the dependent event cannot occur if the body of the corresponding trigger is false.

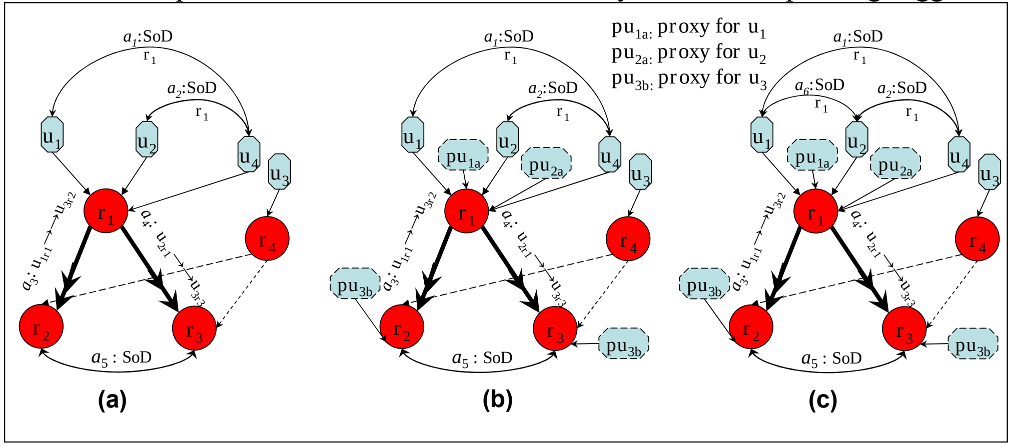

a_1:SoD r_1
a_2:SoD r_1
a_6:SoD r_1
u_1
u_2
u_3
u_4
r_1
r_2
r_3
r_4
a_3: $u_{1r1} \to\to u_{3r2}$
a_4: $u_{2r1} \to\to u_{3r3}$
a_5 : SoD
pu_{1a}
pu_{2a}
pu_{3b}
pu_{1a}: proxy for u_1
pu_{2a}: proxy for u_2
pu_{3b}: proxy for u_3
(a)
(b)
(c)

Fig 2. (a) RBAC graph without proxy assignment leading to undetected conflicts. (b) RBAC graph with proxy assignment to evaluate all possible policy conflicts. (c) RBAC graph with SoD constraint a_6 preventing any conflicting authorizations due to the conflicting constraints a_3, a_4, and a_5

Table I. Description of functions and predicates used in IP transformation rules

Function/predicate	Description
reachable(*u*,*x*)	Returns True if the role or permission node *x* in the RBAC graph can be reached from the user node *u*. This reachability implies that user *u* can access the role or permission *x*.
uassign(*u*,*r*)	Returns True if user *u* is assigned to role *r*.
passign(*p*,*r*)	Returns True if permission *p* is assigned to role *r*.
active-proxy(*u*,*r*)	Returns the active proxy-user of user *u* for role *r*.
passive-proxy(*u*,*r*)	Returns the passive proxy-user of user *u* for role *r*.

The solution to the IP problem with original user-role/permission and constraint variables may yield an instance in which not all policy constraints are evaluated. Omission of these constraints from IP evaluation can be attributed to SoD constraints and event triggers that derive the policy to a single instance out of many other legal policy instances. Because of this omission, it is quite possible that some conflicting constraints may remain undetected and unresolved in the final policy derived from the IP solution. For example in the RBAC policy graph shown in Fig. 2(a), the dependency constraints a_3:$u_{1r1} \to\to u_{3r2}$ and a_4:$u_{2r1} \to\to u_{3r3}$ jointly conflict with the SoD constraint a_5 defined between roles r_2 and r_3. An IP problem derived from the rules given in Table II for the RBAC graph of Fig. 2(a) may yield a solution in which this conflict may remain unresolved. For instance, the assignment $a_1 = a_2 = a_3 = a_4 = a_5 = u_{4_{r1}} = 1$, and $u_{1_{r1}} = u_{2_{r1}} = 0$ is a feasible solution to the IP problem corresponding Fig. 2(a). In this solution, the policy is evaluated without considering the activation of role r_1 by u_1 and u_2, and so the conflicting constraints a_3, a_4, and a_5 remain undetected.

Table II. IP Transformation rules for actual users specified in the original policy

Category	ID	Rule	Meaning
Hierarchy and assignment	1	$\neg \mathit{reachable}(u_i, r_j/p_k) \Rightarrow u_{ir_j/p_k} = 0$	If there is no access path from a user node u_i to role or permission node r_j/p_k then u_i is not authorized to access r_j/p_k .
	2	For an *I-hierarchy* constraint $a_m : r_j \underset{I}{\geq} r_k$, $a_m u_{ir_j} - u_{ir_k} \leq 0$	Any user u_i assuming role r_j also assumes r_k if the constraint $a_m : r_j \underset{I}{\geq} r_k$ is retained in the final policy.
	3	Let $U_{Ik} = \left\{ u \mid \neg \mathit{uassign}(u,r) \wedge (r = r_k \vee r \underset{A}{\geq}^{*} r_k) \right\}$ and $R_{Ik} = \left\{ r_j \mid a_j : r_j \underset{I}{\geq} r_k \right\}$. $\forall u \in U_{IK}$, $\sum_{r_j \in R_{IK}} a_j u_{r_j} - u_{r_k} \geq 0$	Any user u not assigned to r_k or any of its senior roles in the A-hierarchy sense, can access r_k only if u is able to access at least one role in the set R_{Ik}.
	4	Let $R_{Ak} = \left\{ r \mid (r = r_j) \vee (r \underset{A}{\geq}^{*} r_j) \right\}$ and $\mathit{passign}(p_k, r_j)$ · $\sum_{r \in R_{Ak}} u_{ir} - u_{ip_k} \geq 0$	A user u_i can acquire a dynamic permission p_k assigned to role r_j by accessing at least one role in the set R_{Ak}
Enabling	5	$\forall\, u \in \mathsf{U}$, $\forall\, r \in \mathsf{R}$ and all dynamic permissions p_k assigned to r_j, $u_{r_j} - u_{ejr_j} \leq 0$, and $u_{p_k} - u_{ejp_k} \leq 0$.	Any role or permission can only be accessed in enable state. For defining the event corresponding to a enabling of role r_j, a special user u_{ej} is assigned to r_j. $u_{ejr_j} = 1$ implies that r_j is in enable state. Any dynamic permission p_k assigned to r_j becomes enable if $u_{ejp_k} = 1$.
	6	Let $A_u = \{u \mid \mathit{reachable}(u, r_j/p_k)\}$ · $\sum_{u \in A_u} u_{r_j/p_k} - u_{ejr_j/p_k} \geq 0$. Where, p_k is a dynamic permission assigned to r_j.	If a role or permission is enabled, then at least one of the authorized users must access that role or permission in any feasible solution of IP.
SoD	7	For a role (permission specific) SoD constraint a_m between r_j and r_k (p_j and p_k), $a_m (u_{r_j/p_j} + u_{r_k/p_k}) \leq 1$	Conflicting roles or permissions cannot be accessed by same user concurrently.
	8	Let U_c be the set of conflicting users for role r_k (dynamic permission p_k) and a_m be the corresponding SoD constraint. $a_m \sum_{u \in U_c} u_{r_k/p_k} \leq 1$	Conflicting users cannot access same role/permission concurrently.
Dependency triggers	9	For a strong dependency trigger, $a_m : \underset{C_i}{\vee} \left(\underset{u_x \in C_i}{\wedge} u_x \right) \rightarrow\rightarrow u_y$, $\forall C_i$, $a_m \prod_{u_x \in C_i} u_x - u_y \leq 0$, and $a_m \sum_{C_i \in C} \prod_{u_x \in C_i} u_x - u_y \geq 0$, where, $C = \cup C_i$.	User u accesses the dependent role/permission y if and only if the body of the trigger is true.
	10	For a weak dependency trigger, $a_m : \underset{C_i}{\vee} \left(\underset{u_x \in C_i}{\wedge} u_x \right) \rightarrow u_y$, $\forall C_i$, $a_m \prod_{u_x \in C_i} u_x - u_y \leq 0$, where, $C = \cup C_i$.	Whenever the body of the trigger becomes true, user u is forced to access the dependent role/permission y.

In order to include all the legal authorizations and constraints, we expand the IP problem to include proxy user-role and proxy-permission role variables. Additional constraints are defined between the actual users and proxy users in such a way that all access rules and constraints are evaluated in the policy instance generated by the expanded IP problem. There are two types of proxy users: *active proxy* and *passive proxy*. An *active proxy user* upon activation of a role or acquisition of a dynamic permission triggers the activation of another role or permission for a *passive proxy user,* provided an event trigger is defined for such activation. Table III lists the rules for defining IP constraints involving proxy users.

Table III. IP transformation rules for proxy users

Category	ID	Rule	Meaning
Proxy User assignment	11	For a dependency constraint $u_{ix} \Rightarrow u_{jy}$, where $\Rightarrow \in \{\rightarrow, \rightarrow\!\!\rightarrow\}$, assign an active proxy user *pu*pu_{ia} to role *x*. Incase *x* is dynamic permission, assign the active proxy user pu_{ia} to the role to which permission *x* is assigned. Similarly, if y is a role assign a passive user pu_{ib} to role *y* and if *y* is a dynamic permission assign the passive proxy pu_{ib} to the role containing permission *y*.	
	12	If a user u_i is assigned to role r_j then assign an active proxy user pu_{ia} to role r_j.	
	13	Let user u_i be assigned to role r_j or any role r_k such that $r_j \underset{A}{\geq}^* r_k$ and $r_j \underset{I}{\ngeq}^* r_k$, assign an active proxy user pu_{ia} to role r_k.	
Hierarchy and assignment	14	$\neg reachable(pu_i, r_j/p_k) \Rightarrow pu_{ir_j/p_k} = 0$	If there is no access path from a user node pu_i to role/permission node r_j/p_k then pu_i is not authorized to access r_j/p_k .
	15	Consider a user u_i with an active proxy pu_{ia}. If both u_i and pu_{ia} are authorized to access role r_j and all the dynamic permissions p_k, then $u_{ir_j} + pu_{iar_j} = 1$ and $u_{ip_k} + pu_{iap_k} = 1$	Either an authorized user u_i or its active proxy (but not both) must access r_j / p_k in any feasible solution of the underlying IP problem
	16	Consider a user u_i with a passive proxy pu_{ib}. If both u_i and pu_{ib} are authorized to access role r_j and all the dynamic permissions p_k, then $u_{ir_j} + pu_{ibr_j} \leq 1$ and $u_{ip_k} + pu_{ibp_k} \leq 1$	Either an authorized user u_i or its passive proxy (but not both) may or may not access r_j / p_k in a feasible solution of the underlying IP problem
	17	For an *I-hierarchy* constraint $a_m : r_j \underset{I}{\geq} r_k$, $a_m pu_{ir_j} - pu_{ir_k} \leq 0$	Any proxy user pu_i assuming role r_j also assumes r_k if the constraint $a_m : r_j \underset{I}{\geq} r_k$ is retained in the final policy.
	18	Let $PU_{Ik} = \left\{ pu \mid \neg uassign(pu, r) \wedge (r = r_k \vee r \underset{A}{\geq}^* r_k) \right\}$ a nd $R_{Ik} = \left\{ r_j \mid a_j : r_j \underset{I}{\geq} r_k \right\}$. $\forall pu \in PU_{IK}, \sum_{r_j \in R_{IK}} a_j pu_{r_j} - pu_{r_k} \geq 0$	Any proxy user *pu* not assigned to r_k or any of its senior roles in the A-hierarchy sense, can access r_k only if *pu* is able to access at least one role in the set R_{Ik}.
	19	Let $R_{Ak} = \left\{ r \mid (r = r_j) \vee (r \underset{A}{\geq}^* r_j) \right\}$ and $passign(p_k, r_j)$. $\sum_{r \in R_{Ak}} pu_{ir} - pu_{ip_k} \geq 0$	A proxy user pu_i can acquire a dynamic permission p_k assigned to role r_j by accessing at least one role in the set R_{Ak}
Dependency trigger	20a	For a generic dependency constraint, $a_m : \bigvee_{C_i} (\bigwedge_{u_x \in C_i} u_x) \Rightarrow u_y$, where $\Rightarrow \in \{\rightarrow\!\!\rightarrow, \rightarrow\}$, $\prod_{\substack{u_{x_1} \in C_{i_1} \subseteq C_i \\ C_{i_1} \neq \phi}} active\_proxy(u_{x_1}) \prod_{u_x \in C_i \setminus C_{i_1}} u_x \prod_{\substack{C_j \\ C_j \neq C_i}} \left(1 - \prod_{u_z \in C_j} u_z \right) - passive\_proxy(u_y) \leq 0$	This IP constraint implies that in case the body of the dependency trigger a_m is false, the passive proxy of *u* for role/permission *y* is able to access *y*. This ensures that all authorizations related to dependency constraints are checked in the final solution of IP.
	20b	For a strong dependency constraint, $a_m : \bigvee_{C_i} (\bigwedge_{u_x \in C_i} u_x) \rightarrow\!\!\rightarrow u_y$, $\sum_{u_x \in \bigcup C_i} active\_proxy(u_x) - passive\_proxy(u_y) \geq 0$	For strong dependency, 20a and 20b imply that u can access role/permission *y* if an only if the body of a_m is false and at least one active proxy for user-role or user-permission listed in the body of a_m accesses the corresponding role/permission.

Fig. 2(b) shows the policy instance of Fig. 2(a) with proxy user assignment according to Rules 11 - 13 of Table III. With this proxy assignment, Rules 9, 15, and 20 ensure that the dependency triggers a_4 and a_5 always get evaluated, implying that roles r_2 and r_3 are activated by either u_3 or its passive proxy user pu_{3b}. By applying Rule 21 on the policy graph of Fig. 2(b), an SoD constraint is defined between r_2 and r_3 for the proxy user pu_{3b}. This ensures that the conflict among the constraints a_3, a_4, and a_5 is detected and resolved in any feasible solution of the extended IP problem. However, in presence of dependency constraints, specification of role/permission-specific SoD constraints for proxy users becomes very tricky. For example, consider the RBAC policy graph of Fig. 2(c) which is similar to Fig. 2(b) except that it contains an additional user-specific SoD constraint a_6 defined between u_1 and u_2 for role r_1. Because of this additional constraint, users u_1 and u_2 cannot activate role r_1 simultaneously, and therefore the dependency triggers a_3 and a_4 cannot become active at the same time. This means that user u_3 is not able to assume the conflicting roles r_2 and r_3 concurrently, implying that the SoD constraint a_5 will never be violated. Therefore, the RBAC policy of Fig. 2(c) is consistent and conflict free. However, if a SoD constraint of the form of Rule 7 is defined between r_2 and r_3 for the proxy user pu_{3b} then one of the following constraints a_3, a_4, or a_5 will be removed from a consistent policy. Suppose that $a_6\left(pu_{3br_2}+pu_{3br_3}\right)\le 1$, is specified as a constraint in the integer program generated for the RBAC policy of Fig. 2(c). If in the solution none of the users (u_1 and u_2) activate role r_1, then by Rule 15 their proxies do. Because of the dependency constraint defined for the proxy users (Rule 20), the constraints a_3, a_4, and a_5 become conflicting and one of them is removed to get a feasible solution of the extended IP problem.

```
CSP($r_1$, $r_2$, $u_1$, $u_2$, G)
INPUT: $u_1$–$r_1$, $u_2$–$r_2$, and policy graph G
OUTPUT: Set A of conflicting constraints
OUTPUT: cyclic-inheritance if G contains cyclic hierarchy
OUTPUT:  infeasible if $u_1$ cannot access $r_1$ and $u_2$ cannot access $r_2$ concurrently in G
OUTPUT:  feasible otherwise
1.  G' $\leftarrow$ G
2.  Modify G' by removing all the role-specific or permission-specific SoD constraints
    except between $r_x$ and $r_y$ ($r_x \neq r_y$)that have following properties:
        a. There is an immediate or series of dependency constraints from $r_x$ to $r_1$ and
           from $r_y$ to $r_2$. That is, the graph G' contains event triggers of the form
           $u_{xr_x} \Rightarrow^* u_{1r_1}\ u_{yr_y} \Rightarrow^* u_{2r_2}$, where $\Rightarrow \in \{\rightarrow, \rightarrow\rightarrow\}$, $u_x = u_y$.
        b. There is no event dependency trigger that activates $r_x$ for $u_x$ and $r_y$ for $u_y$. Note
           that $u_x = u_y$. If such a trigger exists and condition (a) holds, then insert the
           corresponding dependency constraint variable in the set A.
3.  Modify G' by removing all the user-specific SoD constraint '$a_i$' between any two
    users $u_x$ and $u_y$ ($u_x \neq u_y$) over a role or permission $r$ such that the conditions (a), (b),
    and (c) listed below hold:
        a. Activation of $r$ by $u_x$ causes $u_1$ to access $r_1$, i.e., $u_{xr} \Rightarrow^* u_{1r_1}$
        b. Activation of $r$ by $u_y$ causes $u_2$ to access $r_1$, i.e., $u_{yr} \Rightarrow^* u_{2r_2}$
        c. There exists at least one dependency constraint that causes either $u_x$ or $u_y$ to
           access r.
        d. If all the three conditions (a), (b), and (c) hold for '$a_i$' and there does not exist
           any role or permission $r'$ with conflicting users $u_x$' and $u_y$' such that
           $u_{xr} \Rightarrow^* u_{x'r'} \Rightarrow^* u_{1r_1}$ or $u_{yr} \Rightarrow^* u_{y'r'} \Rightarrow^* u_{2r_2}$, then let $c_1 = \left(1 - a_i \prod a_{kx}\right)$ and
           $c_2 = \left(1 - a_i \prod a_{ky}\right)$, where, each $a_{kx}$ ($a_{ky}$) is a dependency constraint that
           appear in the sequence of dependency constraints $u_{xr} \rightarrow\rightarrow^* u_{1r_1}$
           ($u_{yr} \rightarrow\rightarrow^* u_{2r_2}$). Insert $c_1$ and $c_2$ in the set A.
4.  Modify G' by removing all user-specific SoD constraints between any two users $u_x$
    and $u_y$ ($u_x \neq u_y$) over a role or permission $r'$ such that there does not exist any
    dependence from $u_{xr'}$ to $u_{1r1}$ or from $u_{yr'}$ to $u_{2r2}$.
5.  Write the IP constraint equations for the modified graph G' using the IP
    transformation rules 1 – 9.
6.  For each constraint $a_i$ appearing in the modified graph G', add $a_i$ = 1 as an IP
    constraint.
7.  if no binary feasible solution to the constraints formulated in above steps exists then
    A = $\varnothing$ and return cyclic-inheritance and A.
8.  Add $u_{1r_1} = 1$ and $u_{2r_2} = 1$ as IP constraint.
9.  if no binary feasible solution to the constraints formulated in above steps exists then
    return infeasible.
10. return feasible and A.
```

Fig. 3. Constraint satisfiability algorithm that checks the possibility of SoD violations due to dependency constraints

Table IV. Transformation rules for SoD constraints involving proxy users

Role/permission Specific SoD, a_n		
Condition: Let a_n represents a role/permission-specific SoD between r_i and r_j and let u_k be a user such that there exists a passive dependency constraint for u_k over both r_i and r_j (*passive* dependency constraint means that the variables u_{kri} and u_{krj} appear in the head of their respective event triggers, e.g. $u_r \Rightarrow u_{krj}$, where $\Rightarrow \in \{\rightarrow\rightarrow, \rightarrow\}$. If *CSP*($r_i$, r_j, u_k, u_k, G) = cyclic-inheritance or feasible then the following IP constraints need to be added		
ID	Rule	Meaning
21a	$\forall a_i \in$ A (if A = ϕ, then $a_i = 1$) $a_i a_n(pu_{kbr_i} + pu_{kbr_j}) \le 1$, $a_i a_n(u_{kr_i} + pu_{kbr_j}) \le 1$, $a_i a_n(pu_{kbr_i} + u_{kr_j}) \le 1$	In any feasible solution of the underlying IP problem, the conflicting roles/permissions r_i and r_j cannot be accessed by u_k and/or its passive proxy pu_{kb} simultaneously.
21b	$\forall a_i \in$ A (if A = ϕ, then $a_i = 1$) If $uassign(pu_{ka}, r_i)$ then $a_i a_n(pu_{kar_i} + pu_{kbr_j}) \le 1$ If $uassign(pu_{ka}, r_j)$ then $a_i a_n(pu_{kar_i} + pu_{kar_j}) \le 1$	In any feasible solution of the underlying IP problem, the conflicting roles/permissions r_i and r_j cannot be simultaneously accessed by active and/or passive proxies of user u_k. Where, pu_{ka} and pu_{kb}, respectively, denote the active and passive proxies of u_k.
User Specific SoD		
Condition: Let a_m represents a user-specific SoD between users u_i and u_j for role r_k. if $\left(\left(uassign(u_i, r_n) \vee uassign(u_j, r_n)\right) \wedge r_n \underset{I}{\geq^*} r_k \wedge r_n \neq r_k\right)$ then the IP constraints defined in Rule 22 needs to be added.		
ID	Rule	Meaning
22	$a_m(pu_{iar_k} + pu_{iar_k}) \le 1$, if the active proxy pu_{ia} of user u_i and active proxy pu_{ja} of user u_j are authorized for r_k $a_m(pu_{iar_k} + u_{jr_k}) \le 1$, if the active proxy pu_{ia} of user u_i is authorized for r_k $a_m(u_{ir_k} + pu_{jar_k}) \le 1$, if the active proxy pu_{ja} of user u_j is authorized for r_k	This constraint prevents violation of user-specific SoD because of role inheritance. Two conflicting users u_i and u_j and/or their corresponding active proxies, pu_{ia} and pu_{jb}, cannot access role r_k (for which the users conflict), if one of the users u_i or u_j is assigned to a role senior to r_k in the *I-hierarchy* semantics.
Condition: Let a_n represents a role/permission-level-user-specific SoD between u_i and u_j for role/permission r_k. Also, there exist passive dependency constraints for u_i and u_j over r_k (*passive* dependency constraint means that the variables u_{irk} and u_{jrk} appear in the head of their respective event triggers, e.g. $u_r \Rightarrow u_{irk}$, where $\Rightarrow \in \{\rightarrow\rightarrow, \rightarrow\}$. If *CSP*($r_k$, r_k, u_i, u_j, G) = cyclic-inheritance or feasible then the following IP constraints need to be added		
ID	Rule	Meaning
23a	$\forall a_i \in$ A (if A = ϕ, then $a_i = 1$) $a_i a_n(pu_{ibr_k} + pu_{jbr_k}) \le 1$, $a_i a_n(u_{ir_k} + pu_{jbr_k}) \le 1$, $a_i a_n(pu_{ibr_k} + u_{jr_k}) \le 1$	In any feasible solution of the underlying IP problem, conflicting users u_i and u_j and/or their respective passive proxies pu_{ib} and pu_{jb} cannot simultaneously access r_k (for which u_i and u_j conflict).
23b	$\forall a_i \in$ A (if A = ϕ, then $a_i = 1$) $a_i a_n(pu_{iar_k} + pu_{jbr_k}) \le 1$, if the active proxy pu_{ia} of user u_i is authorized for r_k $a_i a_n(pu_{ibr_k} + pu_{jar_k}) \le 1$, if the active proxy pu_{ja} of user u_j is authorized for r_k	In any feasible solution of the underlying IP problem, role/permission r_k cannot be accessed simultaneously by the active and/or passive proxies of conflicting user u_i and u_j respectively. Where, pu_{ia} and pu_{ib} (pu_{ja} and pu_{jb}), respectively, denote the active and passive proxies of u_i (u_j).

In order to avoid the above discrepancy, an SoD constraint involving proxy users is only defined after ensuring that no other valid constraint prevents the violation of such SoD constraint. The *constraint satisfiability problem* (*CSP*) algorithm shown in Fig. 3 is used to determine the possibility of violation of role/permission-specific or user-specific SoD constraint due to the event dependency constraints. *CSP* takes the conflicting user-role/permission pairs (u_1-r_1, u_2-r_2) and the policy graph G as input and finds a configuration of the policy that allows u_1 to access r_1 and u_2 to access r_2 simultaneously. Note that for role/permission-specific SoD $u_1 = u_2$, and r_1 and r_2 are conflicting roles/permissions. For a user-specific SoD, u_1 and u_2 are conflicting users for $r_1 = r_2$. Lines 2 – 5 of the algorithm create a modified policy graph G' by removing all the SoD constraints that do not prevent violation of the queried SoD constraint. In case the modified graph G' is not conflict-free because of cyclic hierarchy (cf. Section 3.1.2) or other inconsistencies, *CSP* returns cyclic-inheritance (line 8) and therefore the SoD under consideration needs to be defined for the corresponding proxy variables to avoid any discrepancy in the final policy. *CSP* returns infeasible if it is not possible for u_1 to access r_1 and u_2 to access r_2 simultaneously in any valid configuration of the modified graph G'. This implies that the queried SoD constraint in the original policy graph G can never be violated, so this SoD constraint should not be defined for corresponding proxy users. If there exists a configuration in the modified graph G' in which u_1 accesses r_1 and u_2 accesses r_2 concurrently, then CSP returns feasible. However, before returning CSP finds a set of constraints A that conflicts with the given

SoD (line 3d). If any of the constraint $a_i \in$ A is retained in the final policy then the corresponding SoD needs to be defined for proxy users. The rules for defining the SoD constraints for proxy users are listed in Table IV.

3.1.2 Cyclic Hierarchy

Cyclic hierarchy is a form of inconsistency that arises because of the presence of one or more hierarchy paths consisting of *I*-hierarchy and/or *A*-hierarchy edges from a role to itself. In order to resolve this inconsistency, one of the hierarchical edges from all cyclic paths needs to be removed. Cyclic hierarchy conflicts can be specified in the IP problem using a special user variable. For any role r_j that has a cyclic inheritance path to itself, a special user u_{cj} is assigned to r_j and the following IP constraints are defined:

a. $u_{cjr_j} = 1$

b. If there is no inheritance path from role r_j to any role r_m then $u_{cjr_m} = 0$

c. For a hierarchy constraint $a_m : r_k \underset{f}{\geq} r_j$, where $f \in \{I, A\}$, the special user u_{cj}, assigned to r_j, cannot access the senior role r_k if the hierarchy constraint 'a_m' is retained in the final policy, *i.e.*, $a_m u_{cjr_k} = 0$.

d. For any hierarchy constraint $a_n : r_p \underset{f}{\geq} r_q$, if $r_j \underset{f}{\geq^*} r_p$ and $r_p \neq r_q$, then $a_n u_{cjr_p} - u_{cjr_q} \leq 0$

e. For a role r_p such that $r_j \underset{f}{\geq^*} r_p$ and $r_j \neq r_p$, let $R_f = \left\{ r : r \underset{f}{\geq} r_p \right\}$. For any role r_s belonging to the set R_f, let 'a_s' be the corresponding hierarchy constraint. For the special user u_{cj}, assigned to role r_j, the following constraint is added to the IP:

$$\sum_{r_s \in R_f} a_s u_{cjr_s} - u_{cjr_p} \geq 0$$

3.2 Optimal Resolution

The IP transformation rules described in the above section are used to represent the constraints embedded in the underlying access control policy. Once the policy constraints are transformed into IP constraints, an optimal resolution can be achieved by solving the IP problem described in the beginning of Section 3.1. The optimality measure is embedded in the objective function of the corresponding IP problem. Each decision variable in the objective function is assigned a weight and an optimal solution maximizes the over all weight of the objective function. These weights are assigned based on the priority of the underlying constraints and accesses. There are several other factors that need to be considered for weight assignment and a detailed discussion on this issue is beyond the scope of this paper.

```
ConfRes(G_i)
INPUT: Policy graph G_i.
OUTPUT: A consistent and conflict free policy graph.
1. G ← G_i
2. Add all the special and proxy users in the policy graph G according to the IP
   transformation rules.
3. Using the constraint transformation rules, write the constraints for policy graph G in
   algebraic form.
4. For each constraint variable $a_i$, add $a_i = 1$ as an IP constraint.
5. If a binary feasible solution to the constraints formulated in the above steps exists then
   G_i is consistent. In this case return G_i.
6. Reformulate the integer programming (IP) problem by removing all the assignment
   constraints added in step 4.
7. Define the objective function.
8. Find an optimal feasible solution for the IP problem.
9. Remove all the constraints from the policy graph G_i for which the corresponding
   constraint variable $a = 0$ in the optimal feasible solution.
10. Return the modified policy graph G_i.
```

Fig. 4. Conflict resolution algorithm

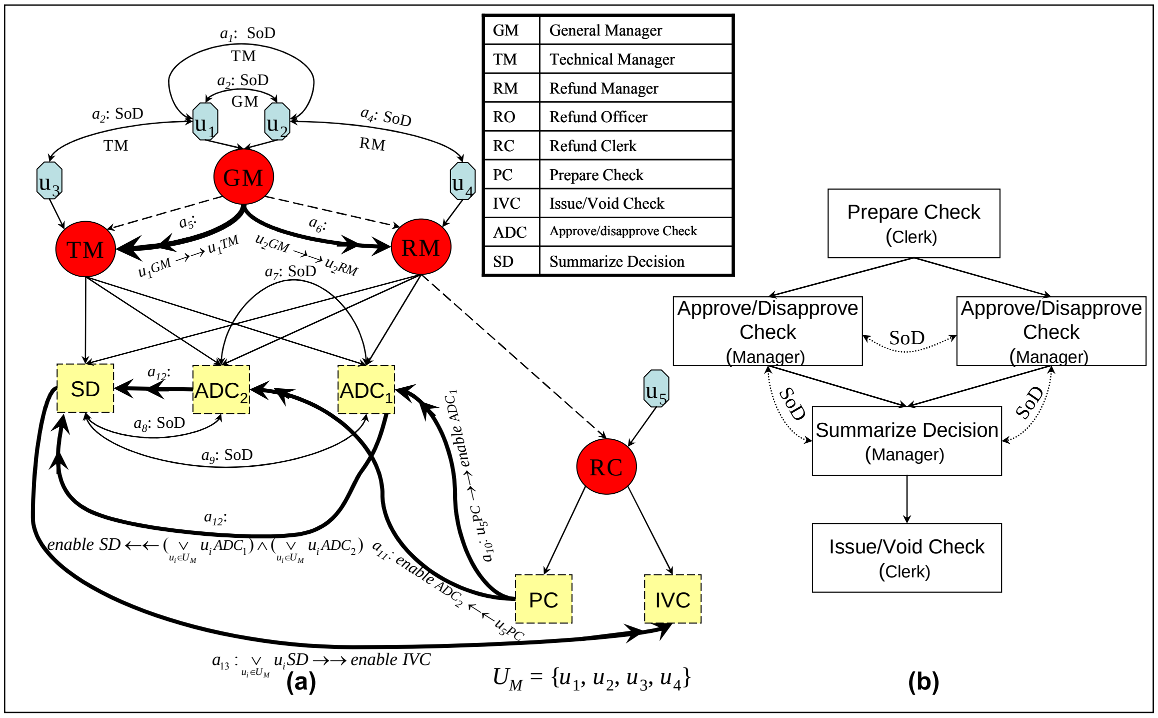
Fig. 4 shows the algorithm *ConfRes* for optimal resolution of conflicts in the event-based RBAC policy. The algorithm takes input an event-driven policy graph G_i and returns a consistent and conflict-free policy graph.

If the input graph policy G$_i$ is consistent then the same policy graph is returned. *ConfRes* first generates proxy user to role and special user to role assignment as discussed in the IP transformation rules. It then transforms the updated set of policy constraints into IP constraints using the transformation rules described in the above section. The consistency of the given policy graph is determined in steps 4 – 5. In step 4 all the variables a_is corresponding to original policy constraints are assigned a value of one implying that all constraints are assumed to be valid. If there exists an assignment that satisfies all the IP constraints formulated in steps 2 – 5, then the given policy graph is consistent. In this case, the algorithm returns with the original policy graph. Otherwise, the IP problem is reformulated by removing all the IP constraints added in step 4. Next the objective function is defined based on the desired optimality criterion and the IP problem is solved for an optimal solution. Finally, all the constraints for which the corresponding constraint variable a_i equals zero in the optimal solution are removed from given policy graph.

A single optimality measure may not be suitable for resolving all types of policy conflicts. For instance, resolution of conflicts pertaining to cycles in role hierarchy may require a different optimality criterion than the resolution of SoD and dependence conflicts. A resolution strategy that tends to maximize accessibility of roles and permissions may work well for resolution of SoD and dependence conflicts but may resolve cyclic hierarchy conflicts in an undesirable manner. Such a resolution strategy would produce an acyclic role hierarchy with more users assigned to senior roles than to junior roles. Generally, in any organization's hierarchy, the number of users authorized for senior roles is lesser than the number of users authorized for junior roles. A better strategy for resolving cyclic hierarchy conflicts would be to remove the hierarchy edges with least priority or weights, assuming that the priorities/weights of the hierarchical edges reflect the responsibilities and authority of corresponding roles. However, a different optimality measure might be considered for resolving policy conflicts other than cyclic hierarchy. In this case, policy conflicts need to be resolved in two steps, with cyclic hierarchy conflicts resolved first followed by resolution of conflicts of other types.

4 Illustrative Example

In this section, we illustrate the proposed conflict resolution technique by considering an event-based RBAC policy that models a workflow and the associated policy constraints. An important aspect highlighted in this example is the resolution of policy conflicts that may arise because of the interplay between workflow execution constraints and the organizational constraints restricting the accessibility of users over certain roles and permissions.



a_1: SoD TM
a_2: SoD GM
a_2: SoD TM
a_4: SoD RM
u1
u2
u3
u4
u5
GM
TM
RM
RC
a_5: $u_1GM \to\to u_1TM$
a_6: $u_2GM \to\to u_2RM$
a_7: SoD
SD
ADC2
ADC1
a_{12}:
a_8: SoD
a_9: SoD
a_{10}: $u_5PC \to\to$ enable ADC1
a_{11}: enable ADC2 $\leftarrow\leftarrow u_5PC$
a_{12}: enable SD $\leftarrow\leftarrow (\vee_{u_i \in U_M} u_iADC_1) \wedge (\vee_{u_i \in U_M} u_iADC_2)$
a_{13}: $\vee_{u_i \in U_M} u_iSD \to\to$ enable IVC
PC
IVC
(a)
$U_M = \{u_1, u_2, u_3, u_4\}$
GM General Manager
TM Technical Manager
RM Refund Manager
RO Refund Officer
RC Refund Clerk
PC Prepare Check
IVC Issue/Void Check
ADC Approve/disapprove Check
SD Summarize Decision
Prepare Check (Clerk)
Approve/Disapprove Check (Manager)
SoD
Approve/Disapprove Check (Manager)
SoD
SoD
Summarize Decision (Manager)
Issue/Void Check (Clerk)
(b)

Fig. 5. (a) Event-driven RBAC policy graph modeling access control policy related to tax refund process. (b) Tax refund process workflow

Fig. 5(a) shows a graphical representation of an event-driven RBAC policy modeling the workflow of the tax refund process of Fig. 5(b). The RBAC graph in Fig. 5(a) consists of four roles: General Manager (GM), Technical Manager (TM), Refund Manager (RM), and Refund Clerk (RC), and five users: u_1, u_2, u_3, u_4, and u_5. Users u_1 and u_2 are conflicting users for role GM (constraint a_2), u_1, u_2, and u_3 are conflicting for TM (constraint a_1), and u_2 and u_4 are conflicting for RM (constraint a_4). The strong dependency constraint a_5:$u_{1GM} \rightarrow\rightarrow u_{3TM}$ implies that use u_1 can activate role TM only if u_1 has activated the role of GM. Similarly the event trigger a_6:$u_{2GM} \rightarrow\rightarrow u_{2RM}$ prevents u_2 to activate the role RM without assuming the role GM. The tax refund workflow shown in Fig. 5(b) is represented in the policy graph using the dynamic permissions and event dependency triggers. The workflow includes the following tasks: prepare check, approve or disapprove check, summarize decision and issue or void check. The dynamic permissions PC, ADC_1 and ADC_2, SD, and IVC correspond to these workflow tasks. Since a check needs to be approved by two separate managers, this task is represented as two separate dynamic permissions ADC_1 and ADC_2. The event triggers defined on dynamic permissions define the execution semantics of the workflow. First a refund clerk (RC) prepares a check which needs to be approved by two separate managers for further processing. These constraints are represented in the graph by the triggers a_{10}:$u_{5PC} \rightarrow\rightarrow$ enable ADC_1 and a_{11}:$u_{5PC} \rightarrow\rightarrow$ enable ADC_2. Enabling of the permissions ADC_1 and ADC_2 imply that the users authorized for the manager roles can now approve or disapprove the refund check. The condition that the check must be approved by two different users assuming the manager role is enforced by defining a permission level SoD a_7 between ADC_1 and ADC_2. After the checks are approved or disapproved, a summary of the decision is prepared. The decision summary is prepared by accessing the permission SD, which can only be accessed by a user authorized for some Manager role who has not approved or disapproved the check. The event trigger a_{12}: $(\underset{u_i \in U_M}{\vee} u_{i ADC_1}) \wedge (\underset{u_i \in U_M}{\vee} u_{i ADC_2}) \rightarrow\rightarrow enable\ SD$, where $U_M = \{u_1, u_2, u_3, u_4\}$, implements this workflow dependency by enabling the permission SD. The permission-specific SoD constraints (a_7, a_8, a_9) among SD, ADC_1, and ADC_2 prevent a single user assuming manager role(s) to perform more than one operation on a given check. Once any user assuming Manager role prepares decision summary by accessing SD, the refund check needs to be issued or void. This workflow dependency is represented by the event trigger $\underset{u_i \in U_M}{\vee} u_i SD \rightarrow\rightarrow enable\ IVC$. Enabling of the permission IVC implies that any authorized user can issue or void the refund check depending on the decision summary.

In order to ensure successful completion of any workflow instance, it is required that in any configuration of the policy shown in Fig 5(a), there exist a set of authorized users who can take the workflow instance to completion by executing the corresponding dynamic permissions. The workflow completion constraints, although not discussed in this paper because of space limitations, are included in this example to illustrate the applicability of the proposed framework in capturing a wide variety of policy constraints. The workflow completion requirement, stated above, and the permission specific SoD constraints among ADC_1, ADC_2 and SD make the given policy inconsistent as there exist some policy instances of Fig. 5(a) in which the workflow completion requirement cannot be satisfied. For example, consider a scenario in which u_1 assumes the role GM. By activating GM, u_1 also acquire the junior role TM because of the dependency constraint a_5. In this case TM cannot be acquired by any other user because of the SoD constraint a_1. Similarly, the user specific SoD constraint a_6 for role RM allows either u_1 and u_2 or u_1 and u_4 to access RM. This means that only two users can assume the Manager roles; whereas, the completion of the underlying workflow requires that three different Managers must perform the tasks associated with ADC_1, ADC_2 and SD.

Apart from the workflow completion inconsistency, the policy graph shown in Fig. 5(a) may also lead to conflicting authorizations. For instance, the dependency constraint a_5 and user-specific SoD constraint a_1 becomes conflicting by activation of role GM by u_1 and assumption of TM by u_2 or u_3. Conflicts in the event-driven RBAC policy shown in Fig. 5(a) are resolved by applying the conflict resolution algorithm *ConfRes*. The IP constraint transformation process produces almost 750 constraints with 130 variables for the event-driven RBAC policy of Fig. 5(a). Fig. 6 shows the IP formulation of the event-driven RBAC graph of Fig. 5(a) generated by the conflict resolution tool that we have developed. Due to space limitations, only the IP constraints corresponding to user u_1 are shown in Fig. 6. The optimality criterion of the IP shown in Fig. 6 is to retain a maximum number of constraints specified in the original policy. This is evident from the objective function that consists of only constraint variables with uniform weight assignment. One of the optimal solution with this optimality criterion as found by our conflict resolution tool is to remove the user-specific SoD constraints a_1 and a_4 from the final policy graph. a_1 prevents users u_1, u_2, and u_3 from assuming the role TM in concurrent sessions; similarly, a_4 prevents users u_2 and u_4 from activating the role RM concurrently. This solution yields an objective function value of sixteen implying that sixteen out of a total of eighteen constraints are retained in this resolution.

5 Correctness of proposed approach

In order to show the correctness of the proposed conflict resolution approach, we need to prove that i) a consistent policy is not modified by the proposed conflict resolution algorithm, ii) the authorizations derived from the resulting policy are conflict free.

The following theorem states that a consistent event-driven RBAC policy remains unchanged during the process of conflict resolution.

Theorem 1: Let G_i be an input RBAC policy graph and G be the graph obtained after applying the proposed conflict resolution algorithm ConfRes. If G_i is consistent then $G = G_i$.
Proof: The proof of this theorem is given in the appendix

Conflicting authorizations in an event-driven RBAC policy occur because of cyclic hierarchy or SoD violations. The event driven RBAC policy obtained after applying the conflict resolution algorithm does not include any conflicts related to cyclic hierarchy and SoD constraints. This is formally stated in the following theorem.

Theorem 2: Let G be the final policy graph obtained after applying the conflict resolution algorithm *ConfRes*. The policy graph G satisfies the following properties:

a. There are no cyclic hierarchies in G (cf. Section 3.1.2).
b. No hierarchy or event-dependency constraint exists in G that violates any role-specific SoD constraint, or permission specific SoD constraint, or role-level user-specific SoD constraint, or permission level user-specific SoD constraint.

Proof: The proof of this theorem directly follows from the Propositions 1- 3 given in the appendix.

6 Related Work

Several research efforts have been devoted to the topic of conflict detection and resolution in access control policies [14, 7, 6, 1, 9]. The resolution techniques proposed in literature can be classified into three classes: resolution by priority [14, 7, 8], resolution by constraint/rule relaxation [7], and resolution by restriction [9]. In the priority-based techniques each authorization is assigned a priority and a high priority authorization prevails over a conflicting low priority authorization. Priorities can either be explicitly assigned to each individual authorization or can be derived based on the administrative scope of the grantor, the specificity and the modality of the authorization [14]. For the latter case, an administrative hierarchy is considered to determine the authorization privileges of the grantors. The authorization of a grantor higher in the privilege hierarchy overrides any conflicting authorization granted by a grantor lower in the hierarchy. In case the grantors are incomparable or conflicting authorizations are specified by same grantor, object and subject hierarchies are used to resolve policy conflicts and a more specific authorization is allowed to supersede a less specific authorization. In some cases, it may not be possible to compare conflicting authorizations based on their specificity. Then, conflicts may be resolved in favor of negative authorizations. The resolution strategy that relies on the authorization privileges of grantors is more suitable for systems that are managed by multiple administrators who may specify contradictory rules for access to a particular resource. Conflict resolution based on the specificity of authorization assumes the existence of an object oriented hierarchy relating the targeting objects and also the subjects. However, such a hierarchy may not exist for objects and/or subjects. For instance, in RBAC models there is no assumption as to how the underlying objects are related to each other? And for all practical purposes the target objects can be considered as atomic entities. Therefore, conflict resolution based on specificity is not applicable in RBAC. The rule that negative authorization takes precedence although resolve policy conflicts but decreases flexibility [14] and may produce deadlocks in case there are other authorizations dependent on the denied authorization For instance in RBAC model, if a user assuming role ‘r’ is denied access to an object ‘o’ assigned to a role junior to ‘r’, then none of the users assigned to role ‘r’ or its senior roles can access the object ‘o’ via the role ‘r’. Other priority-based resolution techniques [8, 13] resolve policy constraints at runtime and do not consider a global optimality measure for conflict resolution.

Restriction-based resolution strategy prevents any conflicting access at the expense of restricting accessibility. Some of the conflict resolution techniques that belong to this group include the event-action constraint cancellation technique proposed by Chomicki et. al. [9], the well-formed model authorization set, and the stable model authorization set with pessimistic reasoning [7]. The restricted access semantics in these approaches may significantly reduce accessibility which may drive the system to a deadlock.

A constraint or rule relaxation strategy avoid deadlock at the expense of dropping some constraints or policy rules. Bertino et. al. in [7] have proposed a conflict resolution technique based on optimistic stable model authorization. This technique uses constraint relaxation and yields maximum accessibility. The authorizations in this technique are derived by evaluating the current system state and all the stable models which may correspond to different relaxation of constraints. The multiplicity of stable models and the differences in the relaxation rules deriving these stable models make this approach non-deterministic, i.e., same access request with same system state evaluated at different times may result in different authorization. This kind of non-determinism cannot be accepted in systems that require well-defined and consistent authorization semantics at all times.

Maximize $\sum_{i=1}^{18} a_i$

Subject to:

Constraints derived from Rule 4

$c_1 : u_{1GM} + u_{1TM} + u_{1RM} - u_{1ADC_1} \geq 0,\ c_2 : u_{1GM} + u_{1TM} + u_{1RM} - u_{1ADC2} \geq 0,\ c_3 : u_{1GM} + u_{1TM} + u_{1RM} - u_{1SD} \geq 0$

Constraints derived from Rule 5

$c_4 : u_{1ADC_1} - u_{eADC_1} \leq 0,\ c_5 : u_{1ADC_2} - u_{eADC2} \leq 0,\ \ c_6 : u_{1SD} - u_{eSD} \leq 0,$

Constraints derived from Rule 6

$c_7 : u_{1GM} + u_{2GM} \geq 1$

Constraints derived from Rule 7

$c_8 : a_7\left(u_{1ADC_1} + u_{1ADC_2}\right) \leq 1,\ \ c_9 : a_8\left(u_{1ADC_2} + u_{1SD}\right) \leq 1,\ c_{10} : a_9\left(u_{1ADC_1} + u_{1SD}\right) \leq 1$

Constraints derived from Rule 8

$c_{11} : a_2\left(u_{1GM} + u_{2GM}\right) \leq 1,\ \ c_{12} : a_1\left(u_{1TM_1} + u_{2TM} + u_{3TM}\right) \leq 1$

Constraints derived from Rule 9

$c_{13} : a_5\left(u_{1GM} - u_{1TM}\right) = 0$

$\forall u_x \in U = \{u_1, u_2, u_3, u_4\},$

$c_{14} : a_{12}\left(u_{1ADC_1} u_{xADC_2}\right) - u_{eSD} \leq 0$ and $a_{12}\left(u_{xADC_1} u_{1ADC_2}\right) - u_{eSD} \leq 0$

$c_{15} : \sum_{u_x \in U} \prod_{u_y \in U} u_{xADC_1} u_{yADC_2} - u_{eSD} \geq 0$

Constraints derived from Rule 14

$c_{16} : pu_{1TM_b GM} = 0,\ c_{17} : pu_{1TM_b RM} = 0,\ \ c_{18} : pu_{1TM_b RC} = 0,\ c_{19} : pu_{1RM_a GM} = 0,\ \ c_{20} : pu_{1RM_a TM} = 0$

Constraints derived from Rule 15

$c_{21} : u_{1GM} + pu_{1GM_a GM} = 1,\ \ c_{22} : u_{1RM} + pu_{1RM_a RM} = 1$

Constraints derived from Rule 16

$c_{23} : u_{1TM} + pu_{1TM_b TM} \leq 1$

Constraints derived from Rule 20(a)

$c_{24} : a_5\left(pu_{1GM_a GM} - pu_{1TM_b TM}\right) = 0$

Constraints derived from Rule 18

$\forall pu \in PU_1 = \{pu_{1GM_a}, pu_{1TM_b}, pu_{1RM_a}\}$ and $\forall p_x \in \{ADC_1, ADC_2, SD\}$

$c_{25} : pu_{GM} - pu_{px} \geq 0,\ c_{26} : pu_{TM} - pu_{px} \geq 0,\ \ c_{27} : pu_{RM} - pu_{px} \geq 0$

Constraints derived from Rule 7 and 16

$c_{34} : u_{1ADC_1} + pu_{1GM_a ADC_1} + pu_{1RM_a ADC_1} + pu_{1TM_b ADC_1} + u_{1ADC_2} + pu_{1GM_a ADC_2} + pu_{1RM_a ADC_2} + pu_{1TM_b ADC_2}$
$+ u_{1SD} + pu_{1GM_a SD} + pu_{1RM_a SD} + pu_{1TM_b SD} \leq 1$

Fig. 6. Integer program corresponding to event-driven RBAC policy of Fig. 5(a)

7 Conclusion

In this paper, we have presented methodology for detection and resolution of inconsistencies and conflicts in event-driven RBAC policies. This methodology uses a binary integer programming (IP) based technique for optimal resolution of policy conflicts. The proposed approach is generic and can be tuned to a variety of optimality measures such as maximizing accessibility, minimizing set of relaxed constraints and maximizing prioritized accesses. We have developed a conflict resolution tool that first transforms an RBAC policy specification into IP problem and then solves the corresponding IP problem for optimal resolution of policy conflicts. In addition, we have proposed two important extensions to current RBAC models for supporting specification and enforcement of access control policies at a very fine granularity. First, we introduce the concept of dynamic permissions which would allow specification of permission centric constraints. We also define two new types of event triggers for modeling stricter form of dependencies that often occur in many workflow applications.

As a future work, we plan to extend the proposed framework to resolve inconsistencies in dynamic workflow-based applications. An important aspect of these applications which is not considered in this paper is the strict temporal inter-dependency between the workflow tasks. These temporal dependencies make the problem of conflict resolution extremely challenging.

8 References

[1] T. Ahmed and A.R. Tripathi, "Static Verification of Security Requirements in Role Based CSCW Systems," in *Proceedings of ACM SACMAT*, June, 2003, pages 196-203.

[2] G. Ahn and R. Sandhu, "Role-based Authorization Constraint Specification," *ACM TISSEC*, Vol. 3(4), Nov. 2000.

[3] V. Atluri and W-K. Huang, "A Petri Net Based Safety Analysis of Workflow Authorization Models," *Journal of Computer Security*, Volume 8, Issue 2/3, 2000.

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

[5] J. Bacon, K. Moody, W. Yao, "A Model of OASIS Role-Based Access Control and its Support for Active Security," *ACM TISSEC*, Vol. 5(4), Nov. 2002, pages 492- 540.

[6] E. Bertino, E. Ferrari, V. Atluri, "The Specification and Enforcement of Authorization Constraints in Workflow Management Systems," *ACM TISSEC*, 2(1), February 1999, pages 65-104.

[7] E. Bertino, F. Buccafurri, E. Ferrari, and P. Rullo, "A Logical Framework for Reasoning on Data Access Control Policies," in *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, 1999, pages 175-189

[8] E. Bertino, P. A. Bonatti, E. Ferrari, "TRBAC: A Temporal Role-based Access Control Model," *ACM TISSEC*, 4(3), August 2001, pages 191-233.

[9] J. Chomicki, J. Lobo, and S. Naqvi, "Conflict Resolution Using Logic Programming," *IEEE TKDE*, Vol. 15(1), Jan. 2003.

[10] D. F. Ferraiolo, D. M. Gilbert, N. Lynch, "An Examination of Federal and Commercial Access Control Policy Needs," In *Proceedings of NISTNCSC National Computer Security Conference*, Baltimore, MD, September 20-23, 1993, pages 107-116.

[11] D, F. Ferraiolo , R. Sandhu , S. Gavrila, D. Richard Kuhn, R. Chandramouli, "Proposed NIST Standard for Role-Based Access Control," *ACM TISSEC*, 4(3), August 2001, pages 224 - 274.

[12] V. C. Hu and D. A. Frincke and D. F. Ferraiolo, "The Policy Machine for Security Policy Management," *LCNS*, Vol. 2074, pages 494-506.

[13] J. Joshi, E. Bertino, U. Latif, and A Ghafoor, "Generalized Temporal Role Based Access Control Model (GTRBAC) (Part I) - Specification and Modeling," Accepted for publication *IEEE TKDE*.

[14] E. Lupu and M. Sloman, "Conflicts in Policy-based Distributed Systems Management," *IEEE TSE*, Vol 25(6) Nov. 1999, pages. 852-869.

[15] D. L. Nazerath, "Investigating the Applicability of Petri Nets for Rule-Based Systems," *IEEE TKDE*, Vol. 4(3), June 1993, pages pp. 402 – 415.

[16] T. A. Nguyen, W. A. Perkins, T. J. Laffey, and D. pecora, "Knowledge Base Verification," *AI Magazine*, No. 49, 1987, pages 69 – 75.

[17] S. Osborn editor. Proc. of the Fifth ACM Workshop on Role-Based Access Control, Berlin, Germany, July 2000.

[18] J. park and R. Sandhu, "The UCONABC Usage Control Model," *ACM TISSEC*, Vol. 7(1), Feb. 2004, pages 128-174.

[19] R. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, "Role-Based Access Control Models," *IEEE Computer* 29(2), IEEE Press, 1996, pages 38-47.

[20] *R. Sandhu, "Role Activation Hierarchies," in Proceedings of the third ACM workshop on Role-based Access Control*, pp.33-40, October 22-23, 1998.

[21] Z. Tari, S. Chan, "A Role-Based Access Control for Intranet Security," *IEEE Internet Computing*, Sept-Oct, 1997, pages 24-34.

[22] A. Tripathi, T. Ahmed, R. Kumar, and S. Jaman, "Design of a Policy-Driven Middleware for Secure Distributed Collaboration," in *Proceedings ICDCS-2002*, pages. 393 – 400.

[23] XACML 1.0 Specification http://xml.coverpages.org/ni2003-02-11-a.html.

9 Appendix

9.1 Proof of Theorems 1 and 2

Proof of Theorem 1: The conflict resolution algorithm, *ConfRes*, before relaxing any constraint checks the consistency of the input policy graph. If the input policy graph is inconsistent then no constraint relaxation takes place. Steps 2 – 4 of *ConfRes* generate IP problem constraints for consistency checking. In this IP problem all the constraint variables a_i are assigned a value of one implying that all constraints are assumed to be valid. If a binary feasible solution to this IP problem exists, then the corresponding input policy graph G_i is consistent. Now we need to show that if the input policy is consistent then there exists a binary feasible solution to the IP problem formulated in steps 2 – 4 of the algorithm *ConfRes*.

It can be proved that the only source of infeasibility that might occur in a IP problem derived from a consistent policy graph is because of the addition of SoD constraints that involve proxy users. A proof of this statement is omitted because of space limitation. SoD constraints involving proxy users are added in the IP problem only if there is a non-zero element in the set A returned by the algorithm CSP (Rules 21 and 23). Since all the constraint variables a_i are assigned a value of one, therefore all the elements of set A have a value of zero (see line 3d of the CSP algorithm). This implies that in the IP problem generated in steps 2 -4, no SoD constraint is defined for proxy users if the corresponding input policy graph is consistent. Therefore, if the policy graph is consistent a binary feasible solution would exist to the IP problem generated in step 2-4 of the algorithm *ConfRes*. □

Proposition 1: Let G be the final policy graph obtained after applying the conflict resolution algorithm *ConfRes*. The graph G does not contain any *cyclic hierarchy*.

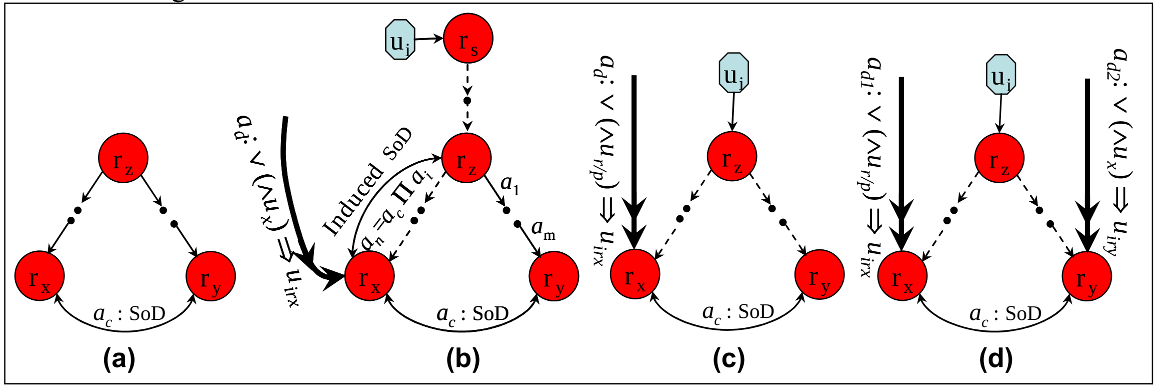
Proof: Suppose the graph G contains cyclic hierarchy. In particular, consider two roles r_i and r_j with the hierarchy constraint, $a_m : r_i \underset{f}{\geq} r_j$ and $r_j \underset{f}{\overset{*}{\geq}} r_i$, where $f \in \{I, A\}$. Let P_{ij} be the set of all hierarchy paths from r_i to r_j. Since r_i and r_j are included in the cyclic hierarchy path, therefore $a_m \neq 0$ and the set P_{ij} is not empty. Because of the hierarchy constraint $a_m : r_i \underset{f}{\geq} r_j$, $a_m u_{cjr_i} = 0$ is added as a constraint to the IP problem, where, u_{cj} is a special user assigned to role r_j to detect cycle in role hierarchy (Section 3.1.2). Since $a_m \neq 0$ therefore $u_{cjr_i} = 0$, i.e., u_{cj} cannot access role r_i.

According to condition *a* of Section 3.1.2, $u_{cjr_j} = 1$. If P_{ij} is not empty then condition *d* leads to the deduction that $u_{cjr_i} = 1$, which contradicts the assumption that $a_m \neq 0$. Therefore, either $a_m = 0$ in any feasible solution or P_{ij} is an empty set, implying that there is no cyclic hierarchy path to r_i via r_j. □

In the event-driven RBAC semantics, users' access to roles/permissions can be classified as *direct access* or *indirect access*. Access to a role is considered as a *direct access* if a user gains access to the role by activating that role. An indirect access to a role *r* can be made by either activating a senior role *r'* that is related to *r* in the *I-hierarchy* sense, or by the occurrence of an event that triggers activation of *r* by some user. In the event-driven RBAC policies, all the constraints related to direct accesses are explicitly stated and all the direct accesses by users to roles/permissions for which some SoD is defined can be easily checked based on the explicit policy statements. In presence of hierarchy and dependency constraints, indirect accesses may lead to conflicting authorizations that result in SoD violations. The proposed conflict resolution strategy ensures that no indirect or derived access will cause violation of any SoD constraint retained in the final policy. We prove this for role-specific and user-specific (role-level) SoD constraints only. The case for permission specific and user-specific (permission-level) SoD constraints can be proved in a similar manner.

Proposition 2: Let G be the final policy graph obtained after applying the conflict resolution algorithm *ConfRes*. For a role-specific SoD constraint a_c in G, No hierarchy or event-dependency constraint exists in G that violates a_c.

Proof: Any indirect access that may lead to the violation of role-specific SoD constraint will fall into one of the four cases shown in Fig. 7.



r_z
r_x
r_y
a_c : SoD
(a)
u_j
r_s
r_z
a_1
a_m
Induced SoD
$a_n = a_c \prod a_i$
a_d: $\wedge (\wedge u_x) \Rightarrow u_{irx}$
r_x
r_y
a_c : SoD
(b)
a_d: $\wedge (\wedge u_{rp}) \Rightarrow u_{irx}$
u_j
r_z
r_x
r_y
a_c : SoD
(c)
a_{d1}: $\wedge (\wedge u_{rp}) \Rightarrow u_{irx}$
u_j
r_z
a_{d2}: $\wedge (\wedge u_x) \Rightarrow u_{iry}$
r_x
r_y
a_c : SoD
(d)

Fig. 7. Case of role-specific SoD violations

Case 1: In this case a user accesses two conflicting roles by activating a senior role that is related to both conflicting roles through an *I-hierarchy*. This is depicted in Fig. 7(a) in which roles r_x and r_y have a SoD constraint a_c and role r_z is a senior role linked to r_y through an *I-hierarchy*. Assume that the policy graph G after conflict resolution contains a sub-graph isomorphic to the graph shown in Fig. 7(a). This implies that in the final solution to the IP $a_c = 1$ and all the variables a_i, corresponding to *I-hierarchy* edges that link r_z to r_x and r_y, are assigned a value of one. As a result of the inheritance path from r_z to r_x and r_y, any authorized user by activating r_z accesses r_x and r_y (Rules 2 and 3). Rules 6 and 15 ensure that at least one regular user or proxy user accesses role r_z. Without Loss of generality, assume that u_i is such user. Therefore, in any feasible solution $u_{ir_x} = 1$ and $u_{ir_y} = 1$. However, the IP constraint $a_c(u_{ir_x} + u_{ir_y}) \le 1$, derived from Rules 7 and 21 , imply that either $a_c = 0$ or there is no inheritance from r_z to r_x or from r_z to r_y. Hence G does not contain a role that is linked to two conflicting roles through an *I-hierarchy*.

Case 2: This case captures the role-specific SoD violations because of the existence of inheritance path and dependency constraint. The inheritance path from r_z to r_y enables a user assuming the role r_z to access r_y (Rules 2 and 3). An induced SoD constraint a_n (see Section 9.2) is defined between r_z and r_x because of the role specific constraint a_c and the inheritance path from r_z to r_y. $a_n = a_c \prod a_i$, where a_i corresponds to an *I-edge* in the path from r_z to r_y. Similar to case 1, Rules 6 and 15 ensure that at least one regular user or proxy user accesses role r_z in any feasible solution to the underlying IP problem Let u_i be such user. The dependency constraint a_d causes u_i to access role r_x, whenever the body of the dependency trigger becomes true. Rules 9, 10 and 20 imply that in any feasible solution either u_i or its passive proxy accesses r_x.

In the underlying IP problem, SoD constraint involving proxy variables is defined between r_z and r_x if there is a possibility that such constraint may be violated (Rule 21). If such a possibility does not exist ($CSP(r_x,r_y,u_i,u_i,$G) = infeasible) then in the graph G, the inheritance path from r_z to r_y and the role specific SoD constraint a_c are non-conflicting. Proposition 4 implies that if the procedure CSP returns infeasible for the given role-specific SoD then the corresponding SOD will never be violated in the policy graph G.

In case there is no other valid constraint in the policy graph G that prevents the violation of induced SoD between r_z and r_x, then SoD constraint with the corresponding proxy variables are added in the underlying IP problem (Rule 21). In addition the IP constraints derived from Rule 7 for the induced SoD constraint a_n, imply that in any feasible solution, if $a_n = 1$, then the conflicting roles r_z and r_x cannot be accessed by u_i and/or its proxies concurrently. This contradicts with the authorizations for u_i derived from Rules 2 and 3 in conjunction with Rules 6 and 15 as explained above. Therefore, in any feasible solution either $a_d = 0$ or $\prod a_i = 0$ or $a_c = 0$. Hence, G does not contain a role-specific SoD constraint that conflicts with a dependency constraint because of *I-hierarchy*, as shown in Fig. 7(b).

Case 3: Fig. 7(c) depicts a generic scenario in which a role-specific SoD conflicts with a dependency constraint. The nature of conflict between the role-specific SoD constraint a_c and the dependency constraint a_d in this case is similar to the conflict between the induced SoD a_n and the dependency constraint a_d in Case 2. Using a reasoning similar to Case 2, we can show that either the SoD constraint a_c never gets violated because of some preventive constraint in G, or one of the constraint a_c or a_d is not present in the policy graph G.

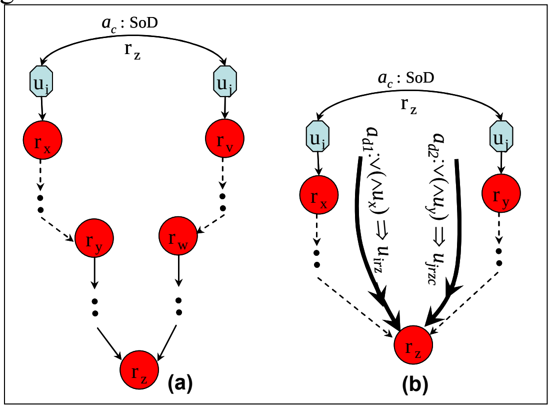
Case 4: In this case two dependency constraints jointly conflict with a role-specific SoD constraint as shown in Fig. 7(d). If we assume that the graph G contains the graph shown in Fig. 7(d) as a sub-graph then in any feasible solution of the underlying IP problem, either u_i or its passive proxy activate the conflicting roles r_x and r_y simultaneously (Rules 9, 10, 15 and 16).

The dependency constraints of Fig. 7(d) do not conflict with the SoD constraint a_c if some other constraint prevents simultaneous activation of dependency triggers a_{d1} and a_{d2}. In this case, no indirect access through the dependency constraints a_{d1} and a_{d2} violates the roles specific SoD a_c.

If $CSP(r_x,r_y,u_i,u_i,\text{G}) \neq$ infeasible, implying that no other constraint prevents the violation of the SoD constraint a_c, then SoD constraints involving u_i and its proxies are added in the underlying IP problem (Rule 21). In addition, the IP constraints derived from Rule 7 for the SoD constraint a_c, imply that in any feasible solution, if $a_c = 1$, then the conflicting roles r_x and r_y cannot be accessed by u_i and/or its proxies concurrently. This contradicts with the authorizations for u_i derived from Rules 9, 10, 15, and 16 as explained above. Therefore, in any feasible solution either $a_c = 0$ or $a_{d1} = a_{d2} = 0$. Hence, G does not contain a role-specific SoD constraint that conflicts with dependency constraints, as shown in Fig. 7(d). □

Proposition 3: Let G be the final policy graph obtained after applying the conflict resolution algorithm *ConfRes*. For a role-level user-specific SoD constraint a_c in G, no hierarchy or event-dependency constraint exists in G that violates a_c.

Proof: Any indirect access that may lead to the violation of a role-level-user-specific SoD constraint can be classified into one of the following two cases.



a_c : SoD
r_z
u_i
u_j
r_x
r_v
r_y
r_w
r_z
(a)
a_c : SoD
r_z
u_i
u_j
r_x
r_y
$a_{d1}: \vee(\wedge u_x) \Rightarrow u_{irz}$
$a_{d2}: \vee(\wedge u_y) \Rightarrow u_{jrzc}$
r_z
(b)

Fig. 8. Role-level user-specific SoD violation cases

Case1: This case covers all the scenarios in which the existence of inheritance path(s) leads to the violation of role-level-user-specific SoD as shown in Fig. 8(a). In Fig. 8(a), users u_i and u_j have a user-specific SoD a_c for role r_z. For a generic case, the relationship between the roles shown in Fig 8(a) is given by:

$$(r_x \underset{A}{\geq^*} r_y \vee r_x = r_y) \wedge (r_y \underset{I}{\geq^*} r_z) \wedge (r_v \underset{A}{\geq^*} r_w \vee r_v = r_w) \wedge (r_w \underset{I}{\geq^*} r_z \vee r_w = r_z)$$

In any feasible solution of the IP problem formulated for the policy graph of Fig. 8(a), either u_i or its proxy (u_j or its proxy) activates role r_y (r_w), which causes the corresponding users to access role r_z because of the inheritance paths from r_y to r_z and from r_w to r_z. The constraint a_c implies that the role r_z cannot be simultaneously accessed by u_i and u_j. In case some valid constraint in G prevents simultaneous activation of r_y by u_i and r_w by u_j, the inheritance paths and the SoD constraint a_c becomes non-conflicting and no violation of SoD constraint a_c can occur through the inheritance paths: $r_y \underset{I}{\geq^*} r_z$ and $r_w \underset{I}{\geq^*} r_z$.

If $CSP(r_x,r_x,u_i,u_j,\text{G}) \neq$ infeasible, implying that no other constraint prevents the simultaneous activation of r_y by u_i and r_w by u_j, then in the underlying IP problem, SoD constraints involving the corresponding proxy variables are defined for a_c (Rule 23). In addition, the IP constraints derived from Rule 8 for the user specific SoD constraint a_c, imply that in any feasible solution, if $a_c = 1$, then both u_i and u_j and/or their respective proxies cannot access role r_z concurrently. This contradicts with the authorizations for u_i and u_j derived from Rules 2 and

3 in conjunction with Rules 6 and 15. Therefore in any feasible solution of the underlying IP problem either a_c = 0, or there is no inheritance from r_y to r_z and from r_w to r_z. Hence, G does not contain a user specific SoD constraint that conflicts with any inheritance path.

Case2: In this case two dependency constraints jointly conflict with a role-level-user-specific SoD as shown in Fig. 8(b). In this figure, users u_i and u_j have a user-specific SoD a_c for role r_z. For a generic case, the relationship between the roles shown in Fig. 8(b) is given by: $(r_x \underset{A}{\geq^*} r_z \vee r_x = r_z) \wedge (r_y \underset{A}{\geq^*} r_z \vee r_y = r_z)$.

Because of the dependency constraint a_{d1} (a_{d2}) depicted in Fig. 8(b), either u_i or its proxy (u_j or its proxy) activates role r_z. The constraint a_c implies that the role r_z cannot be simultaneously accessed by u_i and u_j. In case some valid constraint in G prevents simultaneous activation of r_z by u_i and u_j, the dependency constraints and the SoD constraint a_c becomes non-conflicting and no violation of SoD constraint a_c can occur through the dependency constraints a_{d1} and a_{d2}.

If *CSP*(r_x,r_x,u_i,u_j,G) ≠ infeasible, implying that no other constraint prevents the simultaneous activation of r_z by u_i and u_j, then SoD constraints involving the corresponding proxy variables are defined for a_c (Rule 23). In addition, the IP constraints derived from Rule 8 for the user specific SoD constraint a_c, imply that in any feasible solution, if a_c = 1, then both u_i and u_j and or their respective proxies cannot access role r_z concurrently. This contradicts with the authorizations for u_i and u_j derived from transformation rules for dependency constraints (Rules 9, 10, and 20) in conjunction with Rule 15. Therefore in any feasible solution of the underlying IP problem either a_c = 0, or a_{d1} = a_{d2} = 0. Hence, G does not contain a user specific SoD constraint that conflicts with any dependency constraint. □

Proposition 4: If *CSP*(r_1,r_2,u_1,u_2,G) returns infeasible then no state in which u_1 accesses r_1 and u_2 accesses r_2 simultaneously, can be derived from the event-driven policy graph G.

Proof: The algorithm CSP constructs a modified graph G' from the original policy graph G by removing some of the edges corresponding to SoD constraints. All Other edges associated with hierarchy and dependence constraints in G are included in G'. *CSP*(r_1,r_2,u_1,u_2,G) returns infeasible if it is not possible in the modified graph G' to reach a state in which u_1 accesses r_1 and u_2 accesses r_2 simultaneously. Since G' is a less restricted version of G (because of the removal of SoD constraints) and if such a state cannot be derived from G', it cannot be derived from G. □

9.2 Induced Constraints

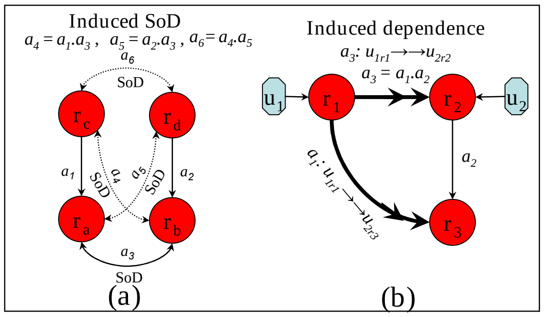
Induced Constraints are added in the event-driven RBAC policy because of incomplete specification and without their addition the policy becomes inconsistent. There are two types of induced constraints: i) *induced SoD* and ii) *induced dependence*.

i) *Induced SoD*: The *I-hierarchy* semantics of RBAC requires that conflicting role-set of a senior role includes the conflicting role-set of all its junior roles that are related to the senior role by *I-hierarchy*. Conflicting role-set of a role *r* is the set of all roles that have a role-specific SoD constraint with *r*. *Induced SoD* constraints are recursively defined from junior roles to senior roles in the following manner.

Consider a role-specific SoD constraint 'a_1' between two roles r_a and r_b as shown in Fig. 9(a). Let roles r_c and r_d be related to r_a and r_b with *I-hierarchy* constraints $a_2 : r_c \underset{I}{\geq} r_a$ and $a_3 : r_d \underset{I}{\geq} r_b$. Induced SoD constraints a_4 = a_1a_2 is added between r_c and r_b, a_5 = a_1a_3 is added between r_d and r_a, and a_6 = a_4a_5 is added between r_c and r_d. Similarly, the induced SoD constraint a_5 propagates upward in the hierarchy. Note that an induced SoD constraint is defined between two roles only if such roles do not have a previous role-specific SoD constraint. The product of constraints in the definition of induced SoD constraint implies that if the original SoD constraint between junior roles is removed, the induced SoD constraint becomes invalid.

ii) *Induced dependence*: The strong dependence semantics in an event trigger requires that the triggered event cannot occur without the occurrence of triggering event. An incomplete constraint specification may violate this dependence. For instance, in the event-driven RBAC policy shown in Fig. 9(b), it is possible that user u_2 accesses role r_3 by activating the senior role r_2 without the activation of role r_1 by u_1. This is a violation of the dependence constraint $a_1 : u_{1r_1} \rightarrow\rightarrow u_{2r_3}$. In order to preserve the strong dependency implied by a_1, a strong dependency constraint of the form $u_{1r_1} \rightarrow\rightarrow u_{2r_2}$ need to be defined. Like *Induced SoDs*, *induced dependence* constraints are recursively defined from junior roles to senior roles. The recursive definition of *induced dependence* is given below.

For a strong dependency constraint $a_1 : \vee_{C_i} \left(\wedge_{u_x \in C_i} u_x \right) \rightarrow\rightarrow u_{ir_j}$, if there exists a role r_k such that $a_2 : r_k \underset{I}{\geq} r_j$ and user u_i is authorized for r_k, then a dependency $a_3 : \vee_{C_i} \left(\wedge_{u_x \in C_i} u_x \right) \rightarrow\rightarrow u_{ir_k}$ is induced by a_1 and a_2. The induced dependency a_3 is related to a_1 and a_2 by the equation $a_3 = a_1 a_2$, implying that the a_3 becomes ineffective if any of the constraints a_1 or a_2 is dropped.


Induced SoD
$a_4 = a_1.a_3$, $a_5 = a_2.a_3$, $a_6 = a_4.a_5$
a_6
SoD
r_c
r_d
a_1
SoD
a_4
a_5
SoD
a_2
r_a
r_b
a_3
SoD
(a)
Induced dependence
a_3: $u_{1r1} \rightarrow\rightarrow u_{2r2}$
$a_3 = a_1.a_2$
u_1
r_1
r_2
u_2
a_1: $u_{1r1} \rightarrow\rightarrow u_{2r3}$
a_2
r_3
(b)
Fig. 9. (a) Induced SoD constraint. (b)) Induced dependence constraint