Analyzing Temporal Role Based Access Control Models

Emre Uzun
Rutgers University
emreu@rutgers.edu

Vijayalakshmi Atluri
Rutgers University
atluri@rutgers.edu

Shamik Sural
Indian Institute of Technology
shamik@cse.iitkgp.ernet.in

Jaideep Vaidya
Rutgers University
jsvaidya@business.rutgers.edu

Gennaro Parlato
University of Southampton
gennaro@ecs.soton.ac.uk

Anna Lisa Ferrara
University of Bristol
anna.lisa.ferrara@bristol.ac.uk

P. Madhusudan
University of Illinois at Urbana-Champaign
madhu@illinois.edu

ABSTRACT

Today, Role Based Access Control (RBAC) is the de facto model used for advanced access control, and is widely deployed in diverse enterprises of all sizes. Several extensions to the authorization as well as the administrative models for RBAC have been adopted in recent years. In this paper, we consider the temporal extension of RBAC (TRBAC), and develop safety analysis techniques for it. Safety analysis is essential for understanding the implications of security policies both at the stage of specification and modification. Towards this end, in this paper, we first define an administrative model for TRBAC. Our strategy for performing safety analysis is to appropriately decompose the TRBAC analysis problem into multiple subproblems similar to RBAC. Along with making the analysis simpler, this enables us to leverage and adapt existing analysis techniques developed for traditional RBAC. We have adapted and experimented with employing two state of the art analysis approaches developed for RBAC as well as tools developed for software testing. Our results show that our approach is both feasible and flexible.

Categories and Subject Descriptors

H.1.0 [**Information Systems Models and Principles**]: General

General Terms

Design, Security, Verification

Keywords

Access Control, Temporal RBAC, Safety Analysis

1. INTRODUCTION

Providing restrictive and secure access to resources is a challenging and socially important problem. Role-based access control (RBAC), which allocates permissions to users via roles, is one of the primary access control models. It has been successfully incorporated in a variety of commercial systems, and has become the norm in many of today's organizations for enforcing security.

One major advantage with RBAC is that, unlike in discretionary access control (DAC) where users can grant access privileges at their own discretion, organizations have central control over its resources. For large organizations, it is normal to have roles in the order of thousands and users in the order of tens of thousands. Typically, security administration is performed by a system security officer (SSO) and is decentralized by delegating administrative activities as it is overwhelming for a single SSO to administer all roles. Administrative RBAC (ARBAC) [14], is a comprehensive model of using RBAC to administer RBAC, by introducing administrative roles and associated privileges. Administrative activities of ARBAC include user-role assignment (URA), permission-role assignment (PRA) and role-to-role assignment (to specify the role hierarchies).

While decentralized RBAC administration enhances the flexibility and scalability, an obvious side effect of it is reduced organizational control over its resources. Therefore, certain security guarantees are essential to ensure controlled delegation and to retain the desired level of control. Such guarantees can only be ensured through a formal analysis of the security properties of the RBAC system. A study of the formal behavior of RBAC models helps organizations gain confidence on the level of control they have on the resources they own. Moreover, security analysis helps them set policies so that owners do not unknowingly lose their control on resources, and aids them make to changes to the policies only if the analysis yields no security policy violations.

Several extensions to RBAC have been proposed for implementation in different domains such as geospatial, mobile and temporal. Noted examples include: temporal RBAC (TRBAC) [2], generalized temporal RBAC (GTRBAC) [10], and Geo-RBAC [3].

In this paper, we limit our focus to the temporal extensions of RBAC. In temporal RBAC, it is possible to specify that a role can be enabled only during a certain time interval (fixed or periodic), or a role can be enabled as long as another role is enabled. In advanced temporal RBAC models there are additional constraints that introduce the notion of role activation, temporal validity on the user-to-role and permission-to-role assignments. Bertino et al. [2] propose a model for TRBAC which has temporal constraints on role activation and deactivation, periodic role enabling and disabling. Later, Joshi et al. [10] propose a generalized version of TRBAC (GTRBAC) that has temporal constraints on user and permission assignments, in addition to the existing temporal constraints on TRBAC.

The main goal of this paper is to offer formal analysis of TRBAC. In order to accomplish this, we first define an administrative model for TRBAC (called ATRBAC). In conventional RBAC systems, where administration is done by users with specific administrative privileges (separation of duty), a separate administrative model in which, the policies to control role assignments, role revocations and other possible operations are declared. These policies are often represented by *can-assign* and *can-revoke* rules, which allow administrative users to assign or revoke roles for users that satisfy certain preconditions, or assign or revoke permissions from roles. Therefore, the assignment / revocation operations are limited to the extent covered by the policies.

The first comprehensive administrative model for RBAC is called ARBAC97 [14], which introduces the notion of an administrator role (AR) with administrative permissions (AP). ARBAC97 has three components: URA97 (user-role assignment), PRA97 (permission-role assignment) and RRA97 (role-role assignment) dealing with the different aspects of RBAC administration. URA97 and PRA97 are exact duals of each other, and are based on the notions of prerequisite conditions and role ranges. They use *"can_assign"* and *"can_assignp"* rules for role and permission assignments, respectively. These rules use the notions of (1) the role range that an administrator role has authority over, and (2) the prerequisite role (or prerequisite condition) needed to exercise that authority. Later, several other administrative models are proposed by Crampton and Loizou [4], Kern et al. [11], Li and Mao [12], Dekker et al. [5] and Bertino et al. [1] to cover more advanced administrative structures.

The temporal constraints can be considered as an additional layer of preconditions to the access control policies, but they are not limited to them. There are two different ways to embed temporal constraints in RBAC. The role assignments can have a temporal dimension, denoting the time intervals (we denote them as *role schedules*) in which the role assignment is valid, or the administrative rules can have a temporal dimension denoting the time intervals (we denote them as *rule schedules*) in which a particular rule is executable. The former restricts the time intervals that the users assume a given role, whereas the latter restricts the time intervals that the administrators execute an administrative rule in the system.

The safety problem, first identified by Harrison, Ruzzo and Ullman [8] can be formulated as testing the following: "Whether there exists a reachable authorization state in which a particular subject possesses a particular privilege for a specific object". Stoller et al. [15] have developed a fixed parameter tractable algorithm to perform security analysis for a simplified version of RBAC. Another security analysis on RBAC using reduction based algorithms is proposed by Li and Tripunitara [12]. Jha et al. [9] compared the use of model checking and first order logic programming for the security analysis of RBAC, and concluded that model checking is a promising approach for security analysis. Ferrara et al. [7] propose an RBAC reachability analysis which exploits abstraction techniques as used in program verification. Their approach reduces the RBAC model into a program and makes a sound analysis to prove whether a particular state is unreachable. While none of the above work address the TRBAC analysis, recently, a timed automata based analysis of GTRBAC is proposed by Mondal et al. [13]. Since it assumes continuous time model, the cost of analysis has been shown to be very expensive; in this paper, we tackle the problem by considering discrete time intervals.

Specifically, we make the following contributions in this paper. (1) We propose an administrative model for the TRBAC that governs the access control rights of the users. (2) We propose a novel approach for security analysis of TRBAC. The main strategy we use while performing the security analysis is to decompose the TRBAC analysis problem into multiple subproblems similar to RBAC. Essentially, we split the problem into simpler RBAC subproblems so that deciding whether a particular target state is reachable or not can be potentially simpler. Additionally, it lends itself to employ the analysis techniques developed for traditional RBAC.

We present two different decomposition strategies – (1) Decomposition using rule schedules and (2) Decomposition using role schedules. Though we can employ any RBAC reachability analysis method, in this paper, we have used the approach developed by Stoller et al. [15] and Ferrara et al. [7], where the latter method can perform multi-user analysis.

The rest of the paper is organized as follows. In Section 2, we review the preliminaries. In Section 3, we introduce the temporal constraints and propose our administrative model for TRBAC. In Section 4, we discuss our two different decomposition strategies that map the TRBAC into a series of subproblems. In Sections 5 and 6, we present the details of the methodologies along with their experimental results. In Section 7, we provide a discussion to compare these two strategies and introduce an approach to analyze the multi-user case. In Section 8, we give our concluding remarks and future work.

2. PRELIMINARIES

In this section, we review the definition of our administrative model for TRBAC. Since we reduce the problem of analyzing administrative TRBAC systems to that of administrative RBAC, we also recall the definition of RBAC and administrative RBAC.

2.1 RBAC and Administrative RBAC

A RBAC policy [6] is a tuple $\langle U, R, PRMS, UA, PA\rangle$ where U, R and $PRMS$ are finite sets of *users*, *roles*, and *permissions*, respectively, $UA \subseteq U \times R$ is the *user-role assignment* relation, $PA \subseteq PRMS \times R$ is the *permission-role assignment* relation. A tuple $(u, r) \in UA$ represents that user u belongs to role r. Similarly, $(p, r) \in PA$ represents that members of role r are granted permission p.

The *Administrative RBAC* (ARBAC) [14] model specifies rules to modify an RBAC system. It is composed of three modules `URA` user-role administration, `PRA` permission-role administration, and `RRA` role-role administration. In this paper we focus on administrative permissions to assign users to roles, therefore, we will only describe the `URA` component.

The `URA` policy allows to make changes to the user-role assignment relation UA by using assignment/revocation rules performed by administrators. Administrators are those users that belong to administrative roles. We denote the set of administrative roles as AR. Some policies consider the set AR to be disjoint from the set of roles R. Those policies are said to meet the *separate administration* constraint [15]. We assume that the set of administrative roles AR is included in the set of roles R, unless differently specified. A user can be assigned to a role if she satisfies the precondition associated to that role. A *precondition* is a conjunction of literals, where each literal is either in positive form r or in negative form $\neg r$, for some role r in R. Following [7], we represent preconditions by two sets of roles Pos and Neg. A user u satisfies a precondition (Pos, Neg) if u is member of all roles in Pos and does not belong to any role of Neg.

Rules to assign users to roles are specified by the set:

$$can\_assign \subseteq AR \times 2^R \times 2^R \times R.$$

A can-assign tuple $(admin, Pos, Neg, r) \in can\_assign$ allows a member of the administrative role $admin$ to assign a user u to roles r provided u's current role memberships satisfies the precondition (Pos, Neg).

Rules to revoke users from roles are specified as follows:

$$can\_revoke \subseteq AR \times R.$$

If $(admin, r) \in can\_revoke$, a member of the administrative role $admin \in AR$, can revoke the membership of any user from role $r \in R$.

A `URA` system can be seen as a state-transition system defined by the tuple $\mathcal{S} = \langle U, R, UA, can\_assign,\ can\_revoke\rangle$. A *configuration* of $\mathcal{S}$ is any user-role assignment relation $UR \subseteq U \times R$. A configuration UR is *initial* if $UR = UA$. Given two $\mathcal{S}$ configurations $c = UR$ and $c' = UR'$, there is a *transition* (or *move*) from c to c' with rule $m \in (can\_assign \cup can\_revoke)$, denoted $c \xrightarrow{\tau_m} c'$, if there exists an *administrative* user ad and administrative role $admin$ with $(ad, admin) \in UR$ and a user $u \in U$, and one of the following holds:

[**can-assign move**] $m = (admin, P, N, r)$, $P \subseteq \{r' \mid (u, r') \in UR\}$, $N \subseteq R \setminus \{r' \mid (u, r') \in UR\}$, and $UR' = UR \cup \{(u, r)\}$;

[**can-revoke move**] $m = (admin, r)$, $(u, r) \in UR$, and $UR' = UR \setminus \{(u, r)\}$.

A *run* (or *computation*) of $\mathcal{S}$ is any finite sequence of $\mathcal{S}$ transitions $\pi = c_1 \xrightarrow{\tau_{m_1}} c_2 \xrightarrow{\tau_{m_2}} \dots c_n \xrightarrow{\tau_{m_n}} c_{n+1}$ for some $n \geq 0$, where c_1 is the *initial* configuration of $\mathcal{S}$. An $\mathcal{S}$ configuration c is *reachable* if c is the last configuration of a run of $\mathcal{S}$.

Reachability Problem: Given an `URA` system $\mathcal{S}$ over the set of roles R and a role $\mathtt{goal} \in R$ and a user u, the *role-reachability problem* asks whether a configuration c with $(\mathtt{u}, \mathtt{goal}) \in \mathtt{c}$ is reachable in $\mathcal{S}$.

2.2 Temporal RBAC

We consider a TRBAC model which is a simplified version of prior work [2, 10]. Temporal RBAC enriches the definition of RBAC by associating to each component a schedule that enables users to assume roles only in certain time intervals. Let T_{MAX} be a positive integer. A *time slot* of *Times* is a pair $(a, a+1)$, where a is an integer, and $0 \leq a < a+1 \leq T_{MAX}$. A time slot $(a, a+1)$ represents the set of all times in the set $[a, b)$, i.e., $\{t \mid a \leq t < b\}$. We use a *time interval*, consisting of a pair (a, b) where a, b are two integers and $0 \leq a < a+1 \leq T_{MAX}$, to represent the set of corresponding time slots $\{(a, a+1), (a+1, a+2), \dots (b-1, b)\}$ succinctly. A *schedule* over T_{MAX} is a set of time slots.

For instance, consider a hospital that works for 24 hours with three shifts (between 8 am and 4 pm, between 4 pm and 12 am, and between 12 am and 8 am). If we want to have the precision of hours, we choose $T_{MAX} = 24$, and a schedule s that covers shifts 8 am–4 pm and 4 pm–12 am is represented as $s = \{(8, 9), (9, 10), \dots, (23, 24)\}$. The schedule definition is a simplified version of the Calendar definition in Bertino et al. [2], where we have simpler periodic constraints and do not have duration constraints. We assume that the system is periodic, thus the schedules repeat themselves after any T_{MAX}; in the hospital example above, time intervals are repeated each 24 hours. Given a schedule s over T_{MAX} and an real number t, we say that t belongs to s, denoted $t \in s$, if there is a time interval $(a, b) \in s$ such that $t' \in [a, b)$, where $t' = t \mod T_{MAX}$.

Let S be the set of all possible schedules over T_{MAX}. A TRBAC policy over T_{MAX} is a tuple $M = \langle U, R, PRMS, TUA, PA, RS\rangle$ where U, R and $PRMS$ are finite sets of *users*, *roles*, and *permissions*, respectively, $TUA \subseteq (U \times R \times S)$ is the *temporal user-role assignment* relation, $PA \subseteq (PRMS \times R)$ is the *permission-role assignment* relation, and $RS \subseteq (R \times S)$ is the *role-status* relation. A tuple $(u, r, s) \in TUA$ represents that user u is a member of the role r only during the time intervals of schedule s. During the life time of the system, a role can be either enabled or disabled. A tuple $(r, s) \in RS$ imposes that role r is *enabled* only during the time intervals of s (and therefore it can be assumed to be a member of r only at these times), and *disabled* otherwise. As in classical RBAC, a tuple $(p, r) \in PA$ means that permission p is associated to role r. Thus, a user u is granted permission p at time $t \in [0, T_{MAX}]$ provided that there exists a role $r \in R$ such that $(u, r, s_1) \in TUA$, $(r, s_2) \in RS$, $(p, r) \in PA$, and $t \in (s_1 \cap s_2)$, for some time intervals s_1 and s_2.

Throughout the paper, we assume that relation RS for each role $r \in R$ contains always exactly one pair with first component r. Similarly, the relation TUA contains exactly one tuple for each pair in $U \times R$. Thus, if a role r is disabled in any time interval, we require that RS relates r with the empty schedule. Similarly, if a user u does not belong to a role r in any time interval, the pair (u, r) is associated to the empty schedule by the relation TUA.

In the next section we introduce our administrative model.

3. ADMINISTRATIVE TRBAC

In temporal RBAC, it is important to decide on the extent to which the notion of time is embedded into the system. Theoretically, if one wants to capture complete behavior of the system at every piece of time instance, then a continuous time model should be built and for the analysis, continuous time models like timed automata should be used. However, continuous time analysis adds extra complexity to the model, since in real life systems, temporal decisions on access control are usually handled in discrete time intervals. For instance, the access control schemes may change depending on predefined intervals like day-time hours and night-time hours, weekdays and weekends (in both cases we have two discrete intervals), or even hourly (24 discrete intervals). Thus, it is sufficient to analyze the behavior of the system in only these intervals rather than having a continuous time analysis.

In the following, we propose an administrative model that allows administrators to make changes to the role-status relation RS and the temporal user-role assignment relation TUA by using respectively enable / disable and assignment / revocation rules. More specifically, the goal of an enable/disable (respectively, assignment/revocation) rule for a role r (for a user u and a role r, resp.) is that of updating the time intervals of the current schedule s associated to each pair $(r, s) \in RS$ (respectively, each triple $(u, r, s) \in TUA$).

In the following, we describe a Temporal `URA` system (`TURA`) as a state-transition system. A `TURA` system is a tuple $\mathcal{S}_T = \langle M, \mathit{can\_enable}, \mathit{can\_disable}, \mathit{t\_can\_assign}, \mathit{t\_can\_revoke}\rangle$ where $M = \langle U, R, PRMS, TUA, PA, RS\rangle$ is a TRBAC policy over T_{MAX}, and $\mathit{can\_enable}$, $\mathit{can\_disable}$, $\mathit{t\_can\_assign}$, $\mathit{t\_can\_revoke} \subseteq (R \times S \times 2^R \times 2^R \times S \times R)$.

A *configuration* of $\mathcal{S}_T$ is a pair (ER, TUR) where $ER \subseteq (R \times S)$ is an enabled-role assignment relation and $TUR \subseteq (U \times R \times S)$ is a user-role assignment relation. A configuration (ER, TUR) is *initial* if $ER = RS$ and $TUR = TUA$. Given two $\mathcal{S}_T$ configurations $c = (ER, TUR)$ and $c' = (ER', TUR')$, we describe below the conditions under which there is a *transition* (or *move*) from c to c' at time $t \in \mathbb{N}$ with rule $m \in \mathcal{M}_{ALL} = (\mathit{can\_enable} \cup \mathit{can\_disable} \cup \mathit{t\_can\_assign} \cup \mathit{t\_can\_revoke})$, denoted $c \xrightarrow{(\tau_m, t)} c'$.

Before defining the transition relation, we first describe the components of move $m = (\mathit{admin}, s_{rule}, Pos, Neg, s_{role}, r)$. Move m can be executed only by a user, say ad, belonging to the *administrative role admin* $\in R$. The times t in which ad can execute m are all those in which ad is assumed to be a member of role admin, and furthermore, t must also belong to the schedule s_{rule} which denotes the time intervals when m can be fired (or we say *valid*): $t \in (s_{ad} \cap s_{admin} \cap s_{rule})$ where $(ad, \mathit{admin}, s_{ad}) \in TUR$ and $(\mathit{admin}, s_{admin}) \in ER$. In the rest of the section we say that m can be *executed* at time t whenever t fulfills the above condition. The component s_{role} is used to update the schedule of a role, or the membership of a user to a role, depending on the kind of rule of m. The pair of disjoint role sets (Pos, Neg) is called the *precondition* of m whose fulfillment depends by the kind of the rule m.

The fulfillment of the precondition of a can-enable and can-disable rule depends on the current status of the other roles. Let $\hat{s} \subseteq s_{role}$. A can-enable or can-disable rule $m = (\mathit{admin}, s_{rule}, Pos, Neg, s_{role}, r)$ satisfies its precondition (Pos, Neg) w.r.t. candidate schedule $\hat{s}$, if for every time slot $\alpha \in \hat{s}$, if 1) for every role $pos \in Pos$, $\alpha \subseteq s_{pos}$ where $(pos, s_{pos}) \in ER$, 2) for every role $neg \in Neg$, $\alpha \cap s_{neg} = \emptyset$, where $(neg, s_{neg}) \in ER$, and 3) α satisfies all preconditions. In other words, a candidate schedule $\hat{s} \subseteq s_{role}$ satisfies a precondition only if each time slot $\alpha \in \hat{s}$ satisfies the precondition individually. Let $(r, s) \in ER$.

[Enabling Rules] A can-enable rule adds a new schedule to a specific role. A tuple $(\mathit{admin}, s_{rule}, Pos, Neg, s_{role}, r) \in \mathit{can\_enable}$ allows to update the tuple $(r, s) \in RS$ to $(r, s \cup \hat{s})$ for some schedule $\hat{s}$, provided that m can be executed at time t and also satisfies its precondition. Formally, rule m is executable at time t, m satisfies its precondition (Pos, Neg) w.r.t. schedule $\hat{s}$, $ER' = (ER \setminus \{(r, s)\}) \cup \{(r, s \cup \hat{s})\}$, and $TUR' = TUR$.
[Disabling Rules] A can-disable rule removes a schedule from a designed role. A tuple $m = (\mathit{admin}, s_{rule}, Pos, Neg, s_{role}, r) \in \mathit{can\_disable}$ allows to update the tuple $(r, s) \in RS$ to $(r, s \setminus \hat{s})$, for some schedule $\hat{s}$, provided that m can be executed at time t, and satisfies its precondition. Formally, m is executable at time t, m satisfies its precondition (Pos, Neg) w.r.t. schedule $\hat{s}$, $ER' = (ER \setminus \{(r, s)\}) \cup \{(r, s \setminus \hat{s})\}$, and $TUR' = TUR$.

The next two rules are similar to those given above with the difference that we now update the schedules associated to each element of the user-role relation. Another difference is that can-assign and can-remove rules have a different semantics to fulfill their preconditions. A user $u \in U$ satisfies a precondition (Pos, Neg) w.r.t. a schedule $\hat{s}$ if for every time slot $\alpha \in \hat{s}$, 1) for every $(u, pos, s_{pos}) \in TUR$ with $pos \in Pos$, $\alpha \subseteq s_{pos}$, 2) for every $(u, neg, s_{neg}) \in TUR$ with $neg \in Neg$, $\alpha \cap s_{neg} = \emptyset$, and 3) α satisfies all preconditions. Let $(u, r, s) \in TUR$.
[Assignment Rules] A tuple $(\mathit{admin}, s_{rule}, Pos, Neg, s_{role}, r) \in \mathit{t\_can\_assign}$ allows to update the user-role assignment relation for the pair (u, r) as follows. Let $\hat{s}$ be a schedule over T_{MAX} with $\hat{s} \subseteq s_{role}$. Then, if m can be executed at time t, and user u satisfies the precondition (Pos, Neg) w.r.t. schedule $\hat{s}$, then the tuple (u, r, s) is updated to $(u, r, s \cup \hat{s})$, i.e. $TUR' = (TUR \setminus \{(u, r, s)\}) \cup \{(u, r, s \cup \hat{s})\}$, and $ER' = ER$.
[Revocation Rules] A tuple $(\mathit{admin}, s_{rule}, Pos, Neg, s_{role}, r) \in \mathit{t\_can\_revoke}$ allows to update the user-role assignment relation for the pair (u, r) as follows. Let $\hat{s}$ be a schedule over T_{MAX} with $\hat{s} \subseteq s_{role}$. Then, if m can be executed at time t, and user u satisfies the precondition (Pos, Neg) w.r.t. schedule $\hat{s}$, then the tuple (u, r, s) is updated to $(u, r, s \setminus \hat{s})$, i.e. $TUR' = (TUR \setminus \{(u, r, s)\}) \cup \{(u, r, s \setminus \hat{s})\}$, and $ER' = ER$.

Reachability problems: A *run* (or *computation*) of $\mathcal{S}_T$ is any finite sequence of $\mathcal{S}_T$ transitions $\pi = c_1 \xrightarrow{(\tau_{m_1}, t_1)} c_2 \xrightarrow{(\tau_{m_2}, t_2)} \ldots c_n \xrightarrow{(\tau_{m_n}, t_n)} c_{n+1}$ for some $n \geq 0$, where c_1 is an *initial* configuration of $\mathcal{S}_T$, $t_1 = 0$, and $t_i \leq t_{i+1}$ for every $i \in [n-1]$. An $\mathcal{S}_T$ configuration c is *reachable within time* t, if there exists a run π in which $c_{n+1} = c$ and $t_n \leq t$. Furthermore, c is simply *reachable* if c is reachable within time t, for some $t \geq 0$.

Let $\mathcal{S}_T$ be a `TURA` system over T_{MAX}, u and r be a user and a role of $\mathcal{S}_T$, respectively, and s be a schedule over T_{MAX}. Given a time t, the *timed reachability problem* for $(\mathcal{S}_T, u, r, s, t)$ asks whether there is a reachable configuration within time t of $\mathcal{S}_T$ in which user u is a member of role r in the schedule s. Similarly, the *reachability problem* for $(\mathcal{S}_T, u, r, s)$ is defined as above where there is no constraint on time t.

In our analyses, we assume Separate Administration, in which there is an administrative user who is assigned to the required administrative roles which are enabled all the time. Hence, the times to fire a rule is only restricted by s_{rule}.

Example 1. Let us now consider an example of a TRBAC system deployed in a hospital. Assume that there are 7 different roles, namely, Employee (*EMP*), Day Doctor (*DDR*), Night Doctor (*NDR*), Practitioner (*PRC*), Nurse (*NRS*), Secretary (*SEC*) and Chairman (*CHR*). Hospital works for 24 hours and there are three different shifts (time slots) from 8 am to 4 pm (Time Slot 1), 4 pm to 12 am (Time Slot 2) and 12 am to 8 am (Time Slot 3). Only the Chairman role (*CHR*) has administrative privileges.

1. (*CHR*, $\{(0,2)\}$, $\{$*DDR*$\}$, $\emptyset$, $\{(0,1)\}$,*PRC*)$\in$ *can_enable*: At time slots 1 and 2, a chairman can enable the role *Practitioner* for the first time slot if the role *Day Doctor* is also enabled during this time slot.
2. (*CHR*, $\{(0,3)\}$, $\{$*EMP*, *NDR*$\}$, $\{(2,3)\}$, *NRS*) $\in$ *can_disable*: At time slots 1, 2 and 3, a chairman can disable the role *Nurse* for the third time slot if the roles *Employee* and *Night Doctor* are enabled at this time slot.
3. (*CHR*, (0,2), $\{$*EMP*$\}$, $\{$*NRS*$\}$, (0,2), *DDR*,) $\in$ *t_can_assign*: At time slots 1 and 2, a chairman can assign the role *Day Doctor* for the first and the second time slots to any user that has *Employee* role and does not have *Nurse* role during these time slots.
4. (*CHR*, $\{(2,3)\}$, $\{$*EMP*$\}$, $\{$*NRS*$\}$, $\{(2,3)\}$, *NDR*) $\in$ *t_can_assign*: At time slot 3, a chairman can assign the role *Night Doctor* for the third time slot to any user that has *Employee* role and does not have *Nurse* role during this time slot.
5. (*CHR*, $\{(0,2)\}$, $\{$*EMP*$\}$, $\{$*DDR*, *NDR*$\}$, $\{(0,3)\}$,*NRS*) $\in$ *t_can_assign*: At time slots 1 and 2, a chairman can assign the role *Nurse* for all time slots of any user that has *Employee* role and does not have *Day Doctor* and *Night Doctor* role during these time slots.
6. (*CHR*, $\{(0,2)\}$, $\{$*DDR*$\}$, $\emptyset$, $\{(0,1)\}$,*PRC*) $\in$ *t_can_assign*: At time slots 1 and 2, a chairman can assign the role *Practitioner* for the first time slot of any user that has *Day Doctor* role during this time slot.
7. (*CHR*, $\{(0,3)\}$,$\{$*NDR*$\}$, $\emptyset$, $\{(2,3)\}$, *PRC*) $\in$ *t_can_assign*: At time slots 1, 2 and 3, a chairman can assign the role *Practitioner* for the third time slot to any user that has *Night Doctor* role during this time slot.
8. (*CHR*, $\{(0,3)\}$, $\emptyset$,$\emptyset$, $\{(0,3)\}$, *SEC*) $\in$ *t_can_revoke*: At time slots 1 and 2, a chairman can revoke the role *Secretary* for all time slots of any user that has *Secretary* role assigned in these slots.

In short, the TRBAC system has *t_can_assign* and *t_can_revoke* rules for users and *can_enable* and *can_disable* rules for roles. Using this distinction, we can analyze the system by two separate subsystems, since *t_can_assign* - *t_can_revoke* rules and *can_enable* and *can_disable* rules are mutually exclusive. Suppose that a rule is used to assign a role to a user. This assignment process and the schedule update operation can be done even though the role is not enabled at that moment. This implies that the user has the rights to assume the role with respect to the given schedule in the future only if the role is enabled at that time. The preconditions for the *t_can_assign* rules only check the presence of role assignments, not the enabled / disabled state of the roles. Enabling/disabling operation on roles is a completely separate procedure handled by *can_enable* / *can_disable* rules. Hence, for a complete reachability analysis, first we need to trace the roles assigned to a particular target user and then we need to trace the time periods that a particular role (a set of roles) can be enabled. Since the latter is similar to the former, in the analysis, we only consider *t_can_assign*/*t_can_revoke* rules.

Our TRBAC administrative model and analysis assume a static set of users, permissions, roles and administrative rules and only cover the user to role assignments without any role hierarchies.

In the next two sections, we propose two alternative strategies to perform security analysis on TRBAC systems.

4. PROPOSED TRBAC ANALYSIS STRATEGIES

As outlined in Section 1, security analysis is essential to understand the implications of the policies and changes to them. The typical analysis questions in a TRBAC related to safety, liveness and mutual exclusion include the following:

1. Safety:
   (a) Will there be no reachable state in which a user u is assigned to a role r at time t?
   (b) Will an enabled role r eventually be disabled?
   (c) Will a user u ever assigned to a role r?
2. Liveness:
   (a) Will an enabled role remain enabled at time t?
   (b) Will a user u eventually be assigned to a role r?
3. Mutual Exclusion: Will a user u be assigned to roles r_1 and r_2 at the same time (i.e., do the time intervals during which u is assigned to roles r_1 and r_2 overlap?

The reachability analysis of TRBAC that we do in this paper can be easily modified to answer all the above questions. Our main idea for performing the reachability analysis is to use a divide and conquer strategy. Since the time dimension is discrete, we decompose the TRBAC analysis problem into multiple subproblems, so that each instance can be treated similar to an RBAC system. We employ two different alternative decomposition strategies – the *rule schedule strategy* and the *the role schedule strategy*. In the following, we explain these two strategies in detail, and we provide computational complexity analysis and the results of the computational experiments in Sections 5 and 6.

4.1 Rule Schedule Strategy

Under this strategy, we split the problem into smaller subproblems with respect to the schedules associated with the rules (s_{rule}) and analyze them serially with respect to time.

Let $m \in \mathcal{M} \subseteq \mathcal{M}_{ALL}$ be a subset of all rules in the system. A *constant region* $\mathcal{C}(a, b, \mathcal{M})$ is a bounded time interval between $t = a$ and $t = b$, $a \leq b$ such that $\forall m \in \mathcal{M}$, $(a, b) \subseteq s^m_{rule}$ and $\nexists m' \notin \mathcal{M}$ such that $s^{m'}_{rule} \subseteq (a, b)$. Informally, if a rule m is included in a constant region $\mathcal{C}$ then it should be valid in all time slots $\alpha \in (a, b)$, and there should not be any other rule m' that is valid in some but not all of the time slots of (a, b). In the rule schedule approach, we split the timeline from 0 to T_{MAX} into non overlapping constant regions $\mathcal{C}_i$ w.r.t the s_{rule} of the roles. Because the rules are static in each $\mathcal{C}_i$, it can be treated similar to an RBAC system. Then, we trace the state space in each constant region serially w.r.t time. In order to control the expansion of the state space, we provide a Sub-schedule assumption to restrict the number of states generated. This assumption and the other details of this decomposition is explained in Section 5.

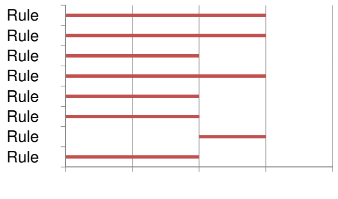


Figure 1: Rule Schedules

Example 2. Now, let us consider the hospital example given in Section 3. There are eight different administrative rules with different valid periods as depicted in Figure 1, where the bars indicate their respective rule schedules. As can be seen from the figure, the set of valid rules does not change in interval (0,2) $\mathcal{C}_1$ and (2,3) ($\mathcal{C}_2$). More specifically, the valid rules for $\mathcal{C}_1$ are 1, 3, 4, 5, 6, 7, 8 and the valid rules for $\mathcal{C}_2$ are 2, 5, 7, 8. Essentially, we decompose the analysis problem of TRBAC into two subproblems which are similar to RBAC problems pertaining to these *constant regions*.

4.2 Role Schedule Strategy

Under this strategy, we decompose the TRBAC analysis problem into multiple subproblems using schedules associated with the roles (s_{role}).

Let $\mathcal{T}(\alpha, \mathcal{M})$ be a subproblem for time slot $\alpha \in (0, T_{MAX})$, where a rule $m \in \mathcal{M}$ if $\alpha \subseteq s^m_{role}$. Informally, a subproblem for a time slot contains all of the rules that is valid w.r.t its role schedule (i.e.: the rules that is authorized to change ER and TUR relations for that particular time slot). The details of this decomposition is explained in Section 6.

Example 3. Consider Figure 2, which shows the role schedules of the rules in the hospital example given in Section 3. Here, we have three distinct time slots (Time Slot 1: (0,1), Time Slot 2: (1,2), Time Slot 3: (2,3)) with different rules. The rules for Time Slot 1 are Rule 1, 3, 4, 6, and 8; for Time Slot 2 are Rule 1, 3, and 8; for Time Slot 3 are Rule 2, 3, 5, 7, and 8.

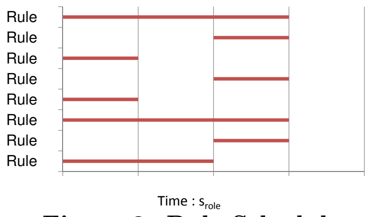


Figure 2: Role Schedules

5. REACHABILITY ANALYSIS USING RULE SCHEDULE STRATEGY

As noted earlier, in this strategy, we split the TRBAC analysis problem into multiple RBAC analysis problems using the rule schedules. In order to handle the RBAC problems, we have adapted the ideas from Stoller et al. [15] and modified them to suit to the temporal case.

In the analysis, we keep track of different configurations c that can be reachable from an initial state c_0. Since we only consider *t_can_assign* and *t_can_revoke* rules and one target user, c is composed of $(\widehat{TUR})$ where $\widehat{TUR} \subseteq R \times S$. Hence in each configuration, we track $(role, s_{role})$ pairs. In the algorithm, we trace *constant regions* $\mathcal{C}_1, \mathcal{C}_2, ...$ serially with respect to time. These regions can be seen as separate RBAC systems. However, $\mathcal{C}_{i+1}$ depends on $\mathcal{C}_i, \forall i$, which implies the output of an RBAC reachability analysis at $\mathcal{C}_i$ is an input (or initial configuration) to $\mathcal{C}_{i+1}$. Since an RBAC analysis could result in multiple configurations, then, in each *constant region*, a separate RBAC analysis should be performed for each configuration generated by the analysis done in the previous *constant region*. Moreover, there are other issues related to role schedules that are assigned by the rules. Recall that all of the rules have a role schedule which denotes the time intervals that the role can be assigned. But, according to the rule definitions, the administrators are free to choose a sub schedule of the role schedule and assign/revoke the role only for some of the designated time intervals. This further complicates the reachability analysis, since in a serial fashion, one should keep all of the possible schedule combinations for the subsequent time intervals. Therefore we make the following assumption to simplify the algorithm:

Sub-schedule Assumption: For each *t_can_assign* and *t_can_revoke* rule, the assignment and revocation operations are performed using the entire schedule s_{role}. In other words, assume that a schedule s_{role} covers all time slots. This means that an administrator may use this rule to assign the associated role r to a user u, all of the subsets of the schedule s_{role} (as long as the preconditions are satisfied). In our analysis, we assume that s_{role} is assigned or revoked completely - no sub schedule assignments are allowed. Hence, this assumption ensures that a rule can generate only one (new) configuration, which is actually similar to the non temporal analysis.

Here we provide a sketch of the algorithm. The TRBAC reachability analysis starts with an initial configuration c_0 and *constant region* $\mathcal{C}_1$. The state space is expanded using Stoller's algorithm and the rules that are valid at time $t = 0$. At the end of this step, a set of reachable configurations, $\mathcal{S}_1 = \{c_1, c_2, ..., c_M\}$ are obtained. Afterwards, the analysis moves to $\mathcal{C}_2$. For each distinct configuration obtained so far, Stoller's algorithm is used to expand these configurations using the valid rules in this constant region. At the end of this step, we obtain an updated set of reachable configurations $\mathcal{S}_2 \supseteq \mathcal{S}_1$. The algorithm then moves to $\mathcal{C}_3$ and the trace goes in this fashion for a specified number of cycles P of length T_{MAX} (The algorithm returns to $\mathcal{C}_1$ whenever T_{MAX} is reached). Since `TURA` tuple S_T is finite and since the iterations are bounded by the number of cycles, the algorithm is guaranteed to terminate. However since this approach is a greedy heuristic, we are not guaranteed to get an optimal solution.

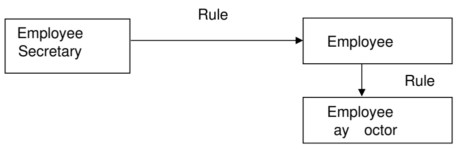


Figure 3: State diagram for the first constant region.

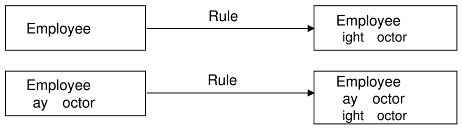


Figure 4: State diagram for the second constant region.

Example 4. Now, suppose we are interested in checking whether a particular part time employee, Alice, with TUA records (Alice, EMP, (1,3)) and (Alice, SEC, (1,2)) would eventually get "Day Doctor" and "Practitioner" roles at the same time. In the configuration c_0, only the roles Employee and Secretary exist with the schedules (1,3) and (1,2), respectively (The other roles have an empty schedule). Then implementing the Stoller's algorithm in the first *constant region* results two new configurations, c_1 and c_2 (Figure 3). In the next *constant region*, the search starts with two configurations, and implementing Stoller's algorithm on these configurations separately results (any duplicate configuration is discarded) two new configurations c_3 and c_4. 4. Revisiting the first and the second *constant regions* does not generate new configurations. Hence the possible reachable configurations of this TRBAC system is c_1, c_2, c_3, c_4. Afterwards, one should trace the *can_enable* and *can_disable* rules in the same fashion to conclude on the enabled roles on each constant region. Since there are no configuration that "Day Doctor" and "Practitioner" appear together, we can conclude that the goal is unreachable.

The complexity of the algorithm depends not only on the number of roles and rules but also depends on the number of time slots, and the schedules (rule-role) that are assigned to the roles. The state space that is generated by this algorithm tends to be exponential in the worst case since it is a brute force state space exploration algorithm. We have implemented our algorithm with C programming language and have run it on a computer with 3 GB RAM and Intel Core2Duo 3.0 GHz processor running Debian Linux operating system. In the experiments, the initial state is set to be an empty state (meaning that none of the roles are assigned), and the rules and the goal are created randomly by the code with respect to the corresponding parameter values for the number of rules, number of roles, number of time slots and the number of cycles. The parameter settings are shown on Table 1. 10 replications are done for each parameter setting and their average is reported. The results are in Figure 5(a),5(b) and 5(c).

According to the results obtained, the run time performances of the algorithms do not tend to be exponential, especially for the number of roles. A possible explanation to this situation is that the datasets are generated randomly.

Table 1: Parameter Settings

Number of Roles $\lvert R\rvert$	100, 500, 900
Number of Rules $\lvert\mathcal{M}_{ALL}\rvert$	100, 500, 900
Number of Time Slots T_{MAX}	100, 500, 900
Number of Cycles P	30 for all cases

Hence there does not exist any "pattern" among the rules. We mean pattern in the sense that, the components that determine the usability of the rules, i.e., all of the precondition relations, rule and role schedules of the moves are generated randomly – so it might become probabilistically harder to satisfy all of these conditions. Nevertheless, the results give some insight about how the algorithm is likely to behave under different parameter settings.

The effect of number of rules while all other parameters are constant is more significant and tends to be an increasing relationship as number of rules increases (See Figure 5(b)). Moreover, the increasing tendency becomes more significant as the number of roles and number of time slots increase. Furthermore, there is a noticeable group formation between the fixed parameters (number of roles and number of time slots). The groups are formed by different number of time slots values indicating that the effect of number of roles is comparably smaller. Finally, Figure 5(c) denotes the relationship between different values of number of time slots parameter when the other two parameters are kept constant. The results show that for the majority of the cases, there is a linearly increasing relationship with the increasing number of rules.

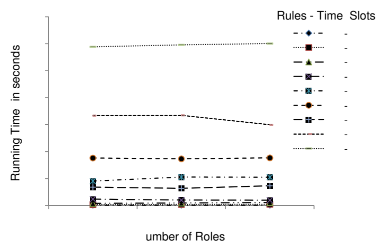
6. REACHABILITY ANALYSIS USING ROLE SCHEDULE STRATEGY

In this approach, we split the TRBAC problem into smaller RBAC subproblems using the role schedules of the rules. The main idea is to generate subproblems $\mathcal{T}(\alpha, \mathcal{M})$ for each time slot $\alpha \in (0, T_{MAX})$ with nontemporal administrative rules, so that the system can be treated like an RBAC. In order to achieve nontemporal administrative rules, (and hence an RBAC system for each time slot), we need to remove two components: Rule schedules and Role Schedules and we need show the inter-time slot independency.

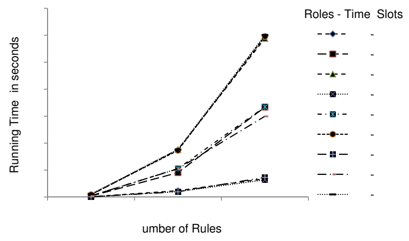
The removal of the role schedules follows the definition of subproblems $\mathcal{T}(\alpha, \mathcal{M})$. For the rule schedules, we observe the Long Run Behavior property of the administrative model that we propose.

Long Run Behavior: In the long run, rule schedules of the rules can be neglected, if the system is periodic.

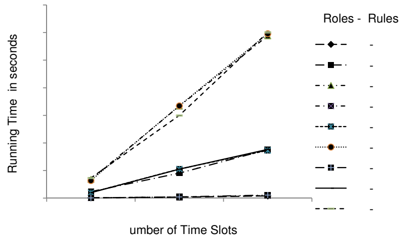
Here we give the intuition of this result. Rule schedules restrict the times that a particular rule can be fired. This means that if a rule m is valid in at least one time slot and if the assignment/revocation (or enabling/disabling) operation that is going to be performed m is necessary for the other rules m', one can wait until m becomes valid, and perform the necessary operation. The other rules m' can be fired next time when the system periodically repeats itself. For example, suppose that we have two roles, r_1 and r_2 and two *t_can_assign* rules $(..., (4, 10), r_1, ...)$ and $(..., (1, 3), \{r_1\}, r_2, ...)$. The first rule states that we can use it only within $(4, 10)$; the second rule states that we can only use it within $(1, 3)$. Notice that if the rules are serially applied with respect to time, then since the second rule has



(a) Effect of Number of Roles

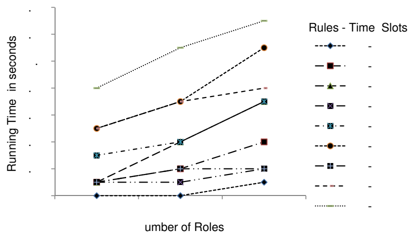


(b) Effect of Number of Rules

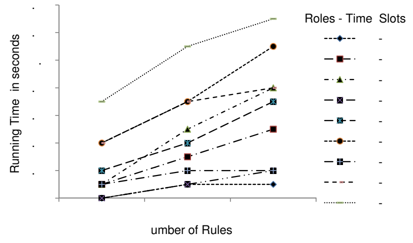


(c) Effect of Number of Time Slots

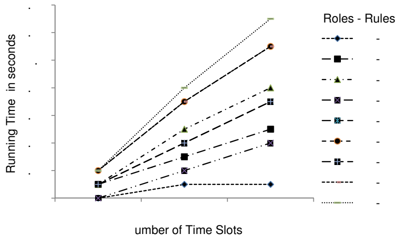
Figure 5: Rule Schedule Approach



(a) Effect of Number of Roles



(b) Effect of Number of Rules



(c) Effect of Number of Time Slots

Figure 6: Role Schedule Approach

a precondition of r_1, we cannot fire second rule if we do not have r_1 already assigned. It means that first we need to wait until first rule becomes valid (until $t = 4$) and assign r_1. Then we should wait until the system restarts from $t = 0$ (since it is periodic) to fire second rule. Then the Long Run Behavior property ensures that for the reachability analysis purposes, if one waits sufficient amount of time then the effects of these kind of rule conflicts can safely be neglected. This property allows us to treat all of the rules valid on the entire time line. Hence, the s_{rule} restrictions can be relaxed from the rules.

In order to handle the independency issues among different time slots, we need to consider preconditions. Recall that in Section 3, we define the preconditions as (Pos, Neg) relationships to be satisfied in order to fire a rule. Now consider a rule $m \in \mathcal{M}$ which belongs to $\mathcal{T}(\alpha, \mathcal{M})$, and $\hat{s} = \alpha$. In order to fire m, the precondition relations declared by (Pos, Neg) of m must be satisfied for $\hat{s}$. For each role $pos \in Pos$ ($neg \in Neg$, resp.) $\hat{s} \subseteq s_{pos}$ ($\hat{s} \cap s_{pos} = \emptyset$, resp.) must be satisfied, which simply depends on the corresponding (single) time slot in s_{pos} (s_{neg}, resp.). Then it is sufficient to check the schedule only for time slot α for each rule. This implies that the preconditions do not depend on other time slots, hence the time slots are independent.

So, using the Long Run Behavior property and the independency of time slots, one can perform an RBAC reachability analysis using the rules $m \in \mathcal{M}$ for time slot α. Then, the whole TRBAC system can be analyzed by a series of independent RBAC systems $\mathcal{T}_i$ traced separately. This reduction provides usability of any RBAC reachability analysis procedure proposed in the literature.

The computational complexity of the algorithm depends on the RBAC analyzer. Suppose that the RBAC analyzer has the complexity $O(\cdot)$ then our approach yields a complexity of $O(T_{MAX}\cdot)$ since we utilize the RBAC analyzer for each time slot (Totally we have T_{MAX} of them). Since the algorithm runs for T_{MAX} iterations and given that the RBAC analyzer terminates, our algorithm is guaranteed to terminate.

We perform computational experiments of this approach under the same parameter and hardware setup as the rule schedule approach. We again report the average run time of 10 replications for each parameter setting. We use the RBAC analyzer by [15]. According to the results obtained, there is a linear and increasing relationship with 100, 500 and 900 roles in the system while all other parameters are constant (See Figure 6(a)). The effect of number of rules while all other parameters are constant is very similar to the effect of roles. There is an increasing relationship in the running time as the number of rules increases (See Figure 6(b)).

Finally, Figure 6(c) denotes the relationship between different values of number of time slots parameter when the other two parameters are kept constant. The results show that there is a linearly increasing behavior as the number of time slots increase. This result is expected since the complexity of the algorithm linearly depends on this parameter.

7. DISCUSSION

The procedures that we propose have certain advantages and disadvantages to be considered. The advantage of using the rule schedule approach is that it simulates the system in a serial fashion until a given time in the future. This is crucial, since it can determine exactly when the goal state is reachable, so it can answer many of the security questions like, will the system be safe at the end of a specified amount of time? On the other hand, the algorithm has an exponential complexity, so running this algorithm for large number of time slots can explode the state space, which may lead to scalability issues. This is depicted in Figure

7, where the number of states that should be considered in subsequent constant regions simply explodes. Lastly, this algorithm tracks the assignments on only one target user.

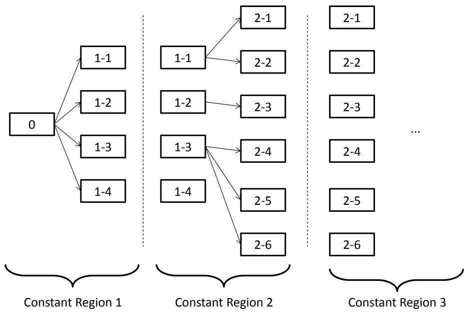


Figure 7: Depicting the Complexity of the Rule Schedule approach

The advantage of the role schedule approach is that it determines the worst case scenarios in the long term by simply running the algorithm for the number of time slots T_{MAX}. Moreover, it provides a direct mapping from TRBAC to a series of independent RBAC problems so that the user is flexible to choose the RBAC analyzer that she wants to use. The performance of this approach will completely depend on the underlying RBAC analyzer. Since the overall complexity depends linearly on the number of time slots, this approach is more scalable than the prior one. The disadvantage of this approach is that, the algorithm only outputs whether the goal state is reachable or not, but it does not output the exact moment when it becomes reachable. In this case, certain security queries, that can be answered by the rule schedule approach, cannot be answered by this approach. An example query can be: "Will there be no reachable state in which a user u is assigned to a role r at time t?". Furthermore, the approach does not work if the time slots are not independent, meaning that the precondition relations require tracking the system status in the other time slots.

From the experimental results, it is clear that the running time of the rule schedule approach grows faster than that of the role schedule approach. Hence, if the security question of interest only covers a short amount of time, then rule schedule approach is useful, but for large number of slots, there might be computational problems related to state space. Role schedule approach is better for these problems but it fails to answer certain time-specific security questions as noted above.

Multi-user Reachability Analysis: In real life systems, where there is no separate administration, there might be certain relations between users in such a way that they might obtain administrative privileges and could use these privileges to assign roles to other users. Our reachability analyses (presented in Sections 5 and 6), that track one target user, cannot capture these specific instances. Hence, the analyses should also support multiple target user tracing to be fully applicable in more realistic scenarios.

Example 5. Consider once again the hospital example given in Section 3. Suppose that we have two additional *t_can_assign* rules as follows: (*SEC*, (1,3), {*EMP*}, {*CHR*, *SEC*} ,(0,3), *ASST*) and (*CHR*, (0,3),{}, {*ASST*}, (1,3), *SEC*) where the role "ASST" stands for "Assistant Chair". Now suppose that we have a target user "John" who has a TUA record ("John", EMP,(0,3)) and one initial administrative user, "CHR". Given these rules, in order for John to get "Assistant Chair" role, there must be a separate user who is a secretary. Otherwise, the goal is always unreachable. Hence, multiple user reachability analysis is crucial to capture a possible security breach in this scenario.

The approach of Ferrara et al. [7] can perform RBAC reachability analysis with multiple target users. Since our role schedule approach has the flexibility to utilize any of the existing RBAC analyzing techniques without altering the algorithm, we extend our analysis using the RBAC analyzer of Ferrara et al. [7]. We perform reachability analysis under the same hardware configuration with different number of users having random initial role assignments. The parameters used are on Table 2. According to the experiments, the results in Figure 8 are obtained. It can be seen that the running time is significantly longer when the analysis involves multiple users.

Number of Roles, Rules, Time Slots	Number of Users
100-100-100	50,100,200
500-500-500	50,100,200
900-900-900	50,100,200

Table 2: Parameters used in the experiments

Finally, in Figure 9, we compare the runtime performance of all three cases, (i) the rule schedule approach, (ii) the role schedule approach with single target user, and (iii) the role schedule approach with multiple target users. As can be seen, the runtime performance of single target user cases are significantly faster than multiple user cases. Furthermore, the role schedule approach is faster than the rule schedule strategy especially when the number of time slots is higher.

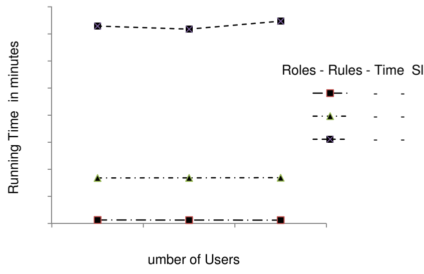


Figure 8: The results of the Role Schedule approach via the RBAC algorithm by Ferrara et al.

8. CONCLUSIONS

Several different extensions of RBAC systems are proposed to cover different requirements or constraints including temporal and geospatial. Although RBAC and its extensions provide a more manageable environment, security analysis is essential to capture any unforeseen security breaches. In this paper, we propose an administrative model for a TRBAC system, and present two different strategies for performing the security analysis of TRBAC that enable one to employ the already existing traditional RBAC analysis techniques. The results of our computational studies indicate that the algorithms are scalable and flexible.

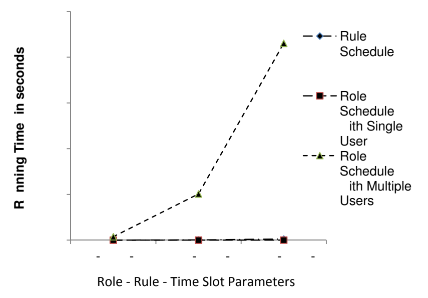


Figure 9: Comparison of all three cases

Although the administrative model that we propose is capable of handling many temporal access control scenarios, it does not cover some of the characteristics such as role hierarchies. We also assume a static environment so that no new rules, roles, users and permissions are introduced to the system. Hence the administrative model does not have rules such as *can create*. As a future work, the model can be extended by including such components to fully represent real life access control systems. The reachability analysis strategies, especially role schedule strategy is flexible enough to handle these additional components without any significant change in the algorithm as long as the RBAC analyzer that is being used supports them. Our computational analysis can be extended to observe the behavior of the strategies by implementing rules derived from real life access control data. In addition, we plan to explore the analysis problem in spatio-temporal RBAC models.

9. ACKNOWLEDGMENTS

The work of Uzun and Sural is supported in part by NSF under grant number 1018414 and the work of Madhusudan is supported in part by NSF grant number 1018182. The work of Atluri is supported through the IR/D by the National Sceince Foundation.

10. REFERENCES

[1] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. Decentralized administration for a temporal access control model. *Information Systems*, 22(4):223–248, 1997.

[2] E. Bertino, P. Bonatti, and E. Ferrari. TRBAC: A temporal role based access control model. *ACM Transactions on Information and System Security*, 4(3):191–233, 2001.

[3] E. Bertino, B. Catania, M. Damiani, and P. Perlasca. GEO-RBAC: A spatially aware RBAC. *ACM Symposium on Access Control Models and Tecnologies*, pages 29–37, 2005.

[4] J. Crampton and G. Loizou. Administrative scope: A foundation for role-based administrative models. *ACM Transactions on Information and System Security*, 6(2):201–231, 2003.

[5] M. Dekker, J. Crampton, and S. Etalle. RBAC administration in distributed systems. *ACM Symposium on Access Control Models and Technologies*, pages 93–102, 2008.

[6] D. Ferraiolo and R. Kuhn. Role-based access control. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[7] A. L. Ferrara, P. Madhusudan, and G. Parlato. Security analysis of access control policies through program verification. In *25th IEEE Computer Security Foundations Symposium*, 2012.

[8] M. Harrison, W. Russo, and J. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8), 1976.

[9] S. Jha, N. Li, M. Tripunitara, Q. Wang, and W. Winsborough. Towards formal verification of role based access control policies. *IEEE Transactions on Dependable and Secure Computing*, 5(4):242–255, October 2008.

[10] J. Joshi, E. Bertino, U. Latif, and A. Ghafoor. A generalized temporal role based access control model. *IEEE Transactions on Knowledge and Data Engineering*, 17(1):4–23, 2005.

[11] A. Kern, A. Schaad, and J. Moffett. An administration concept for the enterprise role-based access control model. *ACM Symposium on Access Control Models and Technologies*, pages 3–11, 2003.

[12] N. Li and Z. Mao. Administration in role-bsed access control. *ACM Symposium on Informationi Computer and Communications Security*, pages 127–138, 2007.

[13] S. Mondal, S. Sural, and V. Atluri. Towards formal security analysis of GTRBAC using timed automata. In *ACM Symposium on Access Control Models and Technologies*, pages 33–42, 2009.

[14] R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for administration of roles. *Proceedings of the 14th ACM conference on Computer and communications security*, 2(1):105–135, 1999.

[15] S. Stoller, P. Yang, C. Ramakrishnan, and M. Gofman. Efficient policy analysis for administrative role based access control. *ACM*, pages 445–455, 2007.