Security Analysis of Administrative Role-Based Access Control Policies with Contextual Information

Khai Kim Quoc Dinh, Tuan Duc Tran, and Anh Truong(✉)

Ho Chi Minh City University of Technology, Ho Chi Minh, Vietnam
{714039,anhtt}@hcmut.edu.vn

Abstract. In many ubiquitous systems, Role-based Access Control (RBAC) is often used to restrict system access to authorized users. Spatial-Temporal Role-Based Access Control (STRBAC) is an extension of RBAC with contextual information (such as time and space) and has been adopted in real world applications. In a large organization, the RBAC policy may be complex and managed by multiple collaborative administrators to satisfy the evolving needs of the organization. Collaborative administrative actions may interact in unintended ways with each other's that may result in undesired effects to the security requirement of the organization. Analysis of these RBAC security concerns have been studied, especially with the Administrative Role-Based Access Control (ARBAC97). However, the analysis of its extension with contextual information, e.g., STRBAC, has not been considered in the literature. In this paper, we introduce a security analysis technique for the safety of Administrative STRBAC (ASTRBAC) Policies. We leverage First-Order Logic and Symbolic Model Checking (SMT) by translating ASTRBAC policy to decidable reachability problems. An extensive experimental evaluation confirms the correctness of our proposed solution, which supports finite ASTRBAC policies analysis without prior knowledge about the number of users.

Keywords: Computer security · Security analysis · Access control · Role-based access control · Spatial-temporal role-based access control

1 Introduction

Enterprise data is a valuable target for malicious users and perpetrators. As the threats become higher nowadays, Access Control [1] is crucial to protect these sensitive resource and information in large scale information management system against unauthorized accesses by mediating every requests to resources and determining whether to grant or deny each individual access requests. Access Control Policy defines the high-level rules applying to Access Control process to regulate and control who has what kind of permissions to access which resources. Three main Access Control models are Discretionary Access Control (DAC) [2], Mandatory Access Control (MAC) [3] and Role-Based Access Control (RBAC) [4]. These model have been continuously developed, from DAC to MAC and RBAC in order to approach higher and higher level of Access Control. The RBAC model, which separate responsibilities in a system where multiple roles are fulfilled, is more suitable for nowadays organizations. The main power of RBAC comes from the Principle of Least Privilege and Separation of Duties, under which no one can take advantage of their granted accesses to perform malicious activities since no standalone individual has all permissions needed for an important operation.

Research works [5, 6] have been devoted to expand RBAC model to support higher level of security where the number of users and administrators keep increasing. Administrative Role-Based Access Control (ARBAC) [7] is proposed which specifies how RBAC policies may be changed by administrators and supports decentralized policy administration. In reality, RBAC policies may be changed occasionally when user changes their job or gets promoted. Thus, it needs to be modified, e.g., add or remove some tuples in the policies, by administrators. Managing changes is a complex task, which requires many administrators. Each administrator can make small changes to parts of the policies that, at first look, seems harmless. However, when in effect, the combination of all of these changes may lead to unsafe state which violates security properties of the policies. Therefore, it is paramount to have a solid change management solution which checks for vulnerabilities and violations in security before applying those changes to the policies. Analysis of these vulnerabilities in Administrative Role-Based Access Control (ARBAC97) [10] have been thoroughly done [17]. Over the last few year, many research have been focused on STRBAC [8] and the Administrative model of STRBAC [9]; however, there is no work focusing on the security analysis of the Administrative model of STRBAC. In order to overcome these shortcomings, in this paper, we propose a security analysis technique for STRBAC based on First Order Logic and Symbolic Model Checking [18]. The main idea is to adapt First Order Logic and Symbolic Model Checking to translate the security analysis problems of ASTRBAC policy to decidable reachability problems where total users and roles are finite but their exact number is not known in order to mechanize the analysis. This paper is the first research on security analysis of the ASTRBAC model. Based on the model checking proposed in [16], our research creates a framework to help security officer aware of the existence of vulnerabilities in the policies before applying those polices to production systems. This model can also return the group of actions which cause the vulnerability to help security officers in detecting and modifying security polices easier according to their needs and keep compliance with security requirements of the organization.

The rest of this paper is organized as follow. In Sect. 1, we introduce the RBAC, ARBAC and related security analysis problems. Section 2 briefly reviews the STRBAC, ASTRBAC models and demonstrates security issues related to these models. Section 3, we describe how we design a technique to analyze the problems. Section 4 summarizes our experiments and results, Sect. 5 is our conclusions.

2 Background

RBAC has been considered as an alternative to the well-known tradition access control such as DAC and MAC. In general, RBAC policy is a tuple (U, R, P, UA, PA) which consist of a set U of users, a set R of Roles, a set P of Permissions, a User-Role

Assignment relation UA ⊆ U × R, a Role-Permission Assignment relation PA ⊆ R P, and for simplicity, we ignore the role hierarchy (see [30] for more details). As stated in RBAC, a user u is a member of a role r if (u, r) ∈ UA; a role r is assigned permission p if (p, r) ∈ PA. Thus, a user is granted to permission p iff there exists a role r ∈ R such that (p, r) ∈ PA and u is the member of r. The UA relations in RBAC keep changing according to the growth and reduction of human resources in an organization while the PA will be less likely to change because of the fact that the change of this part means there is a change in organization structure and this may impact the entire system.

As RBAC expands to support hundreds of users and thousands of permissions in a large organization, several administrators are required to maintain the security policies, and thus, there is a high demand of both a consistent RBAC policy and an assurance that the policy can only be modified by administrators who are authorized to do so. The ARBAC is the most accepted administrative framework to control how RBAC policies might change through administrative actions by assigning or revoking user memberships into roles (URA model, a sub-model of ARBAC [7]). In URA model, administrators can only update the relation UA using the defined administrative actions while the relation PA keeps constant. The first administrative action is to assign users to roles and is defined using ternary relation can_assign(A,C,r) where A (called Administrative) and C (called Simple) are pre-conditions and r is a role (called target role). The second administrative action is to revoke users from roles and is defined using binary relation can_revoke(A,r) where A (called Administrative) is a pre-condition and r is a role (called target role). A pre-condition C is defined as a finite set of signed role, which expressed using +r or –r for r ∈ R. We can say that a user u ∈ U satisfied a pre-condition C (written as u ⊨ C) if for each c ∈ C, u is the member of r ∈ R when c is +r and u is not a member of r ∈ R when c is –r.

While this restriction can limit the administrative actions, research [11] has found that the change to RBAC policy by one administrator may interact in unintended or malicious ways with other administrator's actions. This problem is well-known as the safety problem (also called the reachability problem), which the effects of these interactions may lead to an unintended role assignment to an untrusted user, and let that user have the ability to view or stole sensitive information or resources. In large organizations with thousands of roles, it is hard for policy designers to understand the ARBAC's implications for those interactions.

In general, the safety problem is undecidable [19]. Several techniques have been introduced in [10, 12–14, 17] to solve the safety problem in administrative RBAC. The first research [10] uses state-transition system and logic programming but this approach faces the state space explosion problem and thus, it can only support a small and simple policies. The second [12] approaches the problem using model checking but its runtime is unacceptable with large policies. The third research [13] uses bounded model checking to check large policy but in some cases, it cannot decide whether a policy is safe or not. The forth research [14] simplifies the state space by reducing a policy to a smaller one that preserves the reachability of the target role, which requires limited users, and support only one role in the administrative pre-condition of administrative actions. The fifth research [17] has put forward to analysis and performs better than [14] but it still cannot specify "the reason" the policy is unsafe. Recently, the research [16] has improved from [17] with temporal support. To the best of our knowledge, ours is the first tool to solve the safety problem in STRBAC model.

3 Administrative Spatial Temporal RBAC

TRBAC. In many scenarios, authorization depends on addition contextual information such as the location of user and the time of the day. In this case, an intern of an organization should only be authorized to access the information system of a company only in the branch he is working and during working hours such as between 7 am to 11 am. In order to understand the authorization conditions that depend on spatial temporal constrains, we need to introduce the model of location and time.
In [20, 21], the TRBAC model of time is usually specified by means of intervals periodically repeating time intervals, such as day and night-time (two intervals repeating daily), each hour per day (twenty-four intervals repeating daily), or each day per week (seven intervals repeating weekly). Let T_{Max} be a positive integer and a is a non-negative integer such that $a + 1 \leq T_{MAX}$, A time slot is a pair (a; a + 1); to ease the readability, we will use (8am; 4pm), (4pm; 12am), and (12am; 8am) to denote time slots (0; 1), (1; 2), and (2; 3), respectively. The set of all time slots is $TS_{TMAX} = \{(a; a + 1) \mid 0 \leq a < T_{MAX}\}$. We will usually write TS in place of TS_{TMAX} when T_{MAX} is clear from context and in this paper, we assume T_{MAX} to be given so that the set TS is fixed. A time instant is a non-negative real number. A time instant t belongs to a time slot (a; a + 1), written as $t \in (a; a + 1)$, iff $a \leq (t \bmod T_{MAX}) < a + 1$ where mod is the usual modulo operator, i.e., $t_0 = t \bmod T_{MAX}$ iff there exists a non-negative integer k such that $t = t_0 + k \cdot T_{MAX}$.

The location of a user proposed in [22] should be updated automatically using position determination system (PDS). GPS is one of the most well-known methods to get locations using satellite. Another method requires infrared sensors base station, infrared transponders and active infrared badges that can respond to the sensors to detect and inform user location to base station in small organizations. Other methods use wireless signal strength information from multiple stations to estimate the locations more accurately, which is usually found on mobile devices. In order to make RBAC spatially capable, the authors want to express location in a convenient way that can be interpreted by humans easily and to have a standard way of representing location in raw format, as stored by the system. They define two levels of locations, namely Primitive Location and Logical Location. A Primitive Location L_p is either the volume associated with basic unit of position that is returned by the PDS, or an artificially created volume defined by the administrator for PDSs that have high resolution. These may be created using Constructive Solid Geometry from basic geometric shapes defined by their coordinates. A logical location L_l is a combination of one or more logical or raw locations joined by a $\cup$, $\cap$ and / or \ operator combined with other primitive locations to form a logical location. In this paper, for the sake of simplicity, we will focus on logical location and assume that the location L_l to be updated by the PDS.

An enhanced version of STRBAC is ESTRBAC [23], this model proposes new concepts of role extent and permission extent to define the spatiotemporal access control policies. ESTARBAC still consists components of RBAC, namely, users, roles and permissions but they are associated with either spatial extents or spatial temporal extents. In ESTRBAC model, a set I of intervals is a set of all time slots that participate in at least one spatial temporal access control policy specification (e.g., I is a subset of TS. For simplicity, we consider I = TS in this paper, i.e., every time slots participates in policy specifiacation). Roles and permissions can be available only at specific locations and during specific time intervals, namely, Role Extents (RE) and Permission Extents (PE). In this paper, we will use these RE and PE to support our analysis.

We are now ready to formalize a simplified version of the Enhanced Spatial Temporal RBAC model based on ESTARBAC. The idea is to make RBAC policies depend on constraints based on the notion of both location and time introduced above.

Spatial Temporal RBAC. From now on, we assume that both L_l and T_{MAX} is given so that the set TS of all time slots is fixed and the set UL of all user logical locations $UL \subseteq U \times L$ is updated from the PDS. STRBAC extends RBAC by adding the Role Extents relation $RE \subseteq R \times L \times TS$, the Permission Extends $PE \subseteq P \times L \times TS$ and replacing the user-role assignment UA with its spatial temporal user-role assignment relation $UA \subseteq U \times R \times L \times TS$. For the sake of simplicity, following [24], we exclude role hierarchies.

An extent is a pair (l, ts) which associates spatial- temporal extent to roles or permissions. A role r is enabled at logical location l and time instant t iff there exists a time interval ts such that t belongs to ts and ts $\in$ TS and (r, l, ts) $\in$ RE. A user u is a member of role r at location l and interval ts iff r is enabled at location l and interval ts and (u; r; l, ts) $\in$ UA. A user u can activate role r at location l and interval ts iff u is a member of role r within extent (l, ts) and u is at location l: (u, l) $\in$ UL and the current time-slot is ts. Similarly, a permission p is enabled at location l and interval ts iff (p; l, ts) $\in$ PE. A user u has permission p at location l and interval ts iff there exists role r such that (p; r) $\in$ PA and p is enabled within extent (l, ts) and u is a member of r within extent (l, ts). A user u can access permission p at location l and interval ts iff u has permission p within extent (l, ts) and u is at location l: (u, l) $\in$ UL and the current time-slot is ts. Our STRBAC policy is a tuple (U; R; P; UA; PA; L; TS; UL; RE; PE).

ASTRBAC. One of the Administrative model designed to manage the change of STRBAC policy named ADMINESTAR [9], which allows multiple administrators to modify the STRBAC policy while ensures they cannot abuse the system using their powers. An administrative action consists two components, administrative policies and administrative operation, to define which administrators are allowed to modify ESTRBAC policy. Administrative Policies governs a set of administrative rules to specific which administrative role is authorized to modify ESTARBAC entities of which regular role range. From now on, we focus on the set of administrative rules. All ESTARBAC entities together define the system state, which changes when one or more of the entities change. Administrative Operations are the change of the system state upon their completion only if the administrative policies allow. Administrative Policies and Administrative Operations become more complex if their access control has more attributes.

Our promoted ASTRBAC model. Our model based on ADMINESTAR [9], has more constraints in administrative rule. In ADMINESTAR, administrator condition only have one role so it cannot express actions that require administrator to have more than one role. In our ASTRBAC model, administrator rule is a set of roles so that administrative actions can describe more administrative scenario. ASTRBAC focuses on managing data of these entities: UA, RE, PE, PA by providing actions on them. These actions are divided into four groups depend on their target, can_assign_UA and can_revoke_UA are designed to manage entity UA; can_assign_PA and can_revoke_PA are designed to manage entity PA; can_add_RE and can_delete_RE are designed to manage entity RE; and can_add_PE and can_delete_PE are designed to manage entity PE.

We assume that entity UL is automatically managed by PDS so there is no actions to manage UL in this paper; that the basic entities R, P, L, and TS are finite and constant; and that entity U is infinite. Thus, a STRBAC policy depends on the entities UA, RE, PE, PA. If one of those entities is modified, the STRBAC status will be changed. Hence, the administrative actions need to be examined carefully as these actions can lead the STRBAC policies to a state in which the security requirement of the system is violated. Such problem is well-known as the reachability problem [10, 11].

In the following, let α = (U; R; P; UA; PA; L; TS; UL; RE; PE) be a STRBAC policy. A signed role is an expression of the form +r or –r. A role condition is a finite set of signed roles. A signed role σ in a condition C is positive when there exists a role r such that $\sigma = +$ r, a condition C is negative when there exists a role r such that $\sigma = -$r. An administrative action is a tuple ($\{A_{rule}, l_a, ts_a\}$, $\{R_{rule}, l_u, ts_u\}$, Ud) where tuple $\{A_{rule}, l_a, ts_a\}$ is called admin pre-conditon, A_{rule} is a role condition, (l_a, ts_a) are location and time-slot that together describe spatial temporal constraint on A_{rule}; Tuple $\{R_{rule}, l_u, ts_u\}$ is called user pre-condition where R_{rule} is a role condition; (l_u, ts_u) are location and time-slot that express spatial temporal constraint on R_{rule} that are used together to limit the users whose extents can be modified by the administrator; the Ud element can be an element or many elements depend on each actions. The user pre-condition is optional while admin pre-condition is compulsory for all actions.

Pre-condition. We will discuss the way our system checks these pre-condition.

The admin pre-condition is passed if at least, one administrator satisfies tuple $\{A_{rule}, l_a, ts_a\}$, the positive roles +r in A_{rule} specify the roles administrators must activate at location l_a during time slot ts_a, the negative roles –r in A_{rule} describe the roles administrators cannot activate within the extent (l_a, ts_a). Administrator a can activate role r within the extent (l, ts) iff the formula $\exists\, ts_{cur}$: $[(ad,l) \in UL \wedge (ad, r, l, ts) \in UA \wedge (r, l, ts) \in RE \wedge (ts_{cur} = ts)]$ returns true, where ts_{cur} is the current time-slot determined by system. The following check_role_admin formula ensures the admin pre-condition is checked, if it returns true, there exist an administrator can perform the corresponding action, otherwise, the action is rejected since there is no administrator who can satisfy admin pre-condition.

$$\begin{aligned}&\text{check\_role\_admin}\,(A_{rule}, l_a, ts_a)\ :\ \exists\, ad,\ ts\ [((ad,\ l_a) \in UL) \wedge (ts_{cur} =\ ts_a) \wedge\\ &\wedge_{(role \in Arule)}\ ((ad,\ r,\ l_a,\ ts_a) \in UA \wedge (r,\ l_a,\ ts_a) \in RE)\ \textit{if role} = +r\\ &\wedge_{(role \in Arule)}\ ((ad,\ r,\ l_a,\ ts_a) \notin UA \vee (r,\ l_a,\ ts_a) \notin RE)\ \textit{if role} = -r]\end{aligned}$$

The user pre-condition $\{R_{rule}, l_u, ts_u\}$ limits which users whose extents can be modified by administrators. The positive roles +r in R_{rule} specify the roles user must be assigned within extent (l_u, ts_u), negative roles –r specify the roles users must not be assigned within extent (l_u,ts_u). The User Role Assignment relation UA needs to be checked if user u satisfies (R_{rule}, l_u, ts_u) using check_role_user formula

$$\text{check\_role\_user}(u, R_{rule}, l_u, ts_u) : \bigwedge_{(role\in Rrule)}((u, r, l_u, ts_u) \in UA \textit{ if role} = +r)$$

$$\wedge_{(role\in Rrule)}((u, r, l_u, ts_u) \notin UA \textit{ if role} = -r)$$

If check_role_user returns true, extents (role or permission extents) of user u can be updated, otherwise, it cannot be updated with this action. In conclusion for these pre-condition, an action can be performed iff admin pre-condition is passed and there exist an user can be updated (in some actions).

Administrative Actions. An STRBAC policy has 4 main sets, set UA, RE, PE and PA. The corresponding action for these sets are listed below.

- Can_Assign_UA (A_{rule}, l_a,ts_a, R_{rule}, l_u, ts_u, r)
- Can_Revoke_UA (A_{rule}, l_a, ts_a, R_{rule}, l_u, ts_u, r)
- Can_Add_RE (A_{rule}, l_a, ts_a, r, l, ts)
- Can_Delete_RE (A_{rule}, l_a, ts_a, r, l, ts)
- Can_Assign_PA (A_{rule}, l_a, ts_a, r_t, p_t)
- Can_Revoke_PA (A_{rule}, l_a, ts_a, r_t, p_t)
- Can_Add_PE (A_{rule}, l_a, ts_a, p, l, ts)
- Can_Delete_PE (A_{rule}, l_a, ts_a, p, l, ts)

User-Assignment actions: such actions are designed to manage the actions add or delete tuples in relation UA using role assignment and role revocation actions of users within certain spatial-temporal extents.

- Can_Assign_UA (A_{rule}, l_a, ts_a, R_{rule}, l_u, ts_u, r), where $\{A_{rule}, l_a, ts_a\}$ is admin pre-condition, [R_{rule}, l_u, ts_u] is user pre-condition, (l_u, ts_u,r) is the update tuple. A_{rule} and R_{rule} can contain positive roles +r in companion within extent (l_u, ts_u), negative roles is −r conflict within extent (l_u, ts_u). Notice that, this action can assign (l_u, ts_u, r) to any users that satisfy user pre-condtion. Can_Assign_UA is enabled if admin pre-condtion is satisfied and there exist a user u satisfy user pre-condition $R_{rule.}$

$$\exists u\left(\text{check\_role\_admin}(A_{rule}, l_a, ts_a) \bigwedge \text{check\_role\_user}\left(u, R_{rule,} l_u, ts_u\right)\right)$$

  Once this rule is passed, the tuple (u, r, l_u, ts_u) is added to UA using $UA' = UA \bigcup (u, r, l_u, ts_u)$ where u is the user need to assign new roles.
- Can_Revoke_UA (A_{rule}, l_a, ts_a, R_{rule}, l_u, ts_u, r) where (A_{rule}, l_a, ts_a) is admin pre-condition, (l_u, ts_u,r) is update tuple. R_{rule} can contain positive roles +r in companion within extent (l_u, ts_u), negative roles is −r conflict within extent (l_u, ts_u). Can_Revoke_UA revoke (l_u, ts_u, r) from any users. Administrator need to satisfy the admin pre-condition to enable this rule and there exist a user u satisfy user pre-condition R_{rule}, this check uses formula check_role_admin(A_{rule}, l_a, ts_a). Once

this rule is passed, the tuple (u, l_u, ts_u, r)) is removed from $UA : \exists u\ UA' = UA \backslash (u,\ r,\ l_t,\ ts_t)$ where u is the user need to revoke user roles.

RoleExtent actions: such actions are designed to manage the actions add or delete tuples in relation RE using adding and deleting actions of role within certain spatial-temporal extents. Role extents that are registered in RE can be assigned to users in entity UA. In order to activate a role within a spatial-temporal extent, that role must be assigned to user in UA and enable within that extent in RE.

- Can_Add_RE (A_{rule}, l_a, ts_a, r, l, ts) where (A_{rule}, l_a, ts_a) is admin pre-condition, (r,l, ts) is the update tuple. The admin pre-condition must be checked using formula: check_role_admin (A_{rule}, l_a, ts_a). Once this rule is passed, the spatial-temporal extent of role r is added with tuple (l, ts): $RE' = RE \bigcup (r,\ l,\ ts)$
- Can_Delete_RE (A_{rule}, l_a, ts_a, r, l, ts) where (A_{rule}, l_a, ts_a) is admin pre-condition, (r, l, ts) is the update tuple. The admin pre-condition must be checked to enable this action using formula: check_role_admin (A_{rule}, l_a, ts_a). Once this rule is passed, the spatial-temporal extent of role r is deleted with tuple (l, ts): $RE' = RE \backslash (r,\ l,\ ts)$

Permission-Assignment actions: such actions are designed to manage the actions add or delete tuples in relation PA using permission assignment and permission revocation actions of roles.

- Can_Assign_PA (A_{rule}, l_a, ts_a, r_t, p_t) where {A_{rule}, l_a, ts_a} is admin pre-condition, the (r_t, p_t) is the update tuple. The admin pre-condition must be checked using formula: check_role_admin (A_{rule}, l_a, ts_a). Once this rule is passed, the role r_t and permission p_t is added $PA' = PA \bigcup (r_t,\ p_t)$
- Can_Revoke_PA (A_{rule}, l_a, ts_a, r_t, p_t) where (A_{rule}, l_a, ts_a) is admin pre-condition, (r_t, p_t) is the update tuple. The admin pre-condition must be checked to enable this action using formula: check_role_admin (A_{rule}, l_a, ts_a). Once this rule is passed, the role r_t and permission p_t is r is deleted $PA' = PA \backslash (r_t,\ p_t)$

PermissionExtent actions: such actions are designed to manage the actions add or delete tuples in realtion PE using Can Add and Can Delete actions of permissions within certain spatial-temporal extents.

- Can_Add_PE (A_{rule}, l_a, ts_a, p, l, ts) where {A_{rule}, l_a, ts_a} is admin pre-condition, (p, l, ts) is the update tuple. The admin pre-condition must be checked using formula: check_role_admin (A_{rule}, l_a, ts_a). Once this rule is passed, the permission p and its spatial temporal is added $PE' = PE \bigcup (p,\ l,\ ts)$
- Can_Delete_PE (A_{rule}, l_a, ts_a, p, l, ts) where (A_{rule}, l_a, ts_a) is admin pre-condition, (p, l, ts) is the update tuple. The admin pre-condition must be checked to enable this action using formula: check_role_admin (A_{rule}, l_a, ts_a). Once this rule is passed, the permission p and its spatial temporal is deleted $PE' = PE \backslash (r_t,\ p_t)$

A run of an ASTRBAC system (α_0, φ) is a (possibly infinite) sequence $(\alpha_0; 0)\ldots(\alpha_i; t_i); (\alpha_{i+1}; ti + 1), \ldots$ of states such that $(\alpha_i; t_i) \rightarrow (\alpha_{i+1}; t_{i+1})$ and $t_i \leq t_{i+1}$ for i = 1… n − 1 with n > 1. If the run is finite, i.e. it is of the form $(\alpha_0; 0) \ldots (\alpha_n; t_n)$ for some $n \geq 0$, we say that $(\alpha_n; t_n)$ is the final state of the run.

Even if administrators can only execute a given set of administrative actions mentioned above, it is still very difficult to foresee all possible interleaving of actions when many administrators perform their administrative actions together with their effect on an initial STRBAC policy. Therefore, in some cases, an untrusted user can gain, in some spatial temporal, a permission which that person should not gain. In order to identify this situation, we need to solve the next analysis problem.

A reachability problem for an ASTRBAC system (α_0; φ) is identified by a tuple (u; C_f; $l_{f,}$ ts_f) and amounts to checking if there exists a finite run of the ASTRBAC system whose final state (α_f; l_f, ts_f) is such that user u, location l_s and timeslot ts_f satisfy condition C_f under UA_f and $l_{s,}$ ts_f satisfies C_f under RE_f, and $l_{s,}$ ts_f satisfies C_f under PE_f where α_f = (UA_f; RE_f; PE_f).

Example 1. Consider an STRBAC policy below.

- Let U = {Alice; Bob; Peter; Shan; Mary}
- R = {Manager; Engineer; Technician; Tester; Developer}
- L = {A1 building; A2 building; A3 building}
- TS = {Morning: [8:00am – 12:00pm]; Afternoon: (12:00pm – 18:00pm); Night: [18:00pm – 8:00am]}
- P = {Write_O1; Write_O2; Read_O2; Write_O2}
- UA = {(Alice, Manager, A1 building, Morning); (Bob, Engineer, A2 building, Morning); (Peter, Technician, A3 Building, Afternoon); (Shan, Engineer, A1 building, Afternoon); (Mary, Engineer, A1 Building, Afternoon); (Shan, Tester, A1 building, Afternoon); (Bob, Developer, A2 building, Morning)}
- PA = {(Manger, Write_O1); (Engineer, Write_O2); (Technician, Read_O2); (Manager; Read_O2); (Manager, Write_O2)}
- RE = {(Manager, A2 building, Morning); (Engineer, A1 building, Morning); (Engineer, A2 building, Morning); (Manager, A1 building, Morning); (Technician, A3 building, Afternoon); (Tester, A1 building, Afternoon); (Engineer, A1 building, Afternoon)}
- PE = {(Read_O2, A3 building, Afternoon); (Write_O2, A2 building, Morning); (Write_O1, A1 building, Afternoon)}
- UL = {(Alice, A1 building); (Bob, A2 building); (Shan, A1 building); (Peter, A3 building), (Mary, A1 building)}

The ASTRBAC rule contains these rule.

1. Can_Assign_PA ({Manager}, A1 building, Morning, Engineer, Write_O1)
2. Can_Assign_UA({Manager}, A1 building, Morning, {Engineer, -Technician, -Manager}, A2 building, Afternoon, Tester)
3. Can_Add_RE ({Manager}, A1 building, Morning, Engineer, A1, Afternoon)
4. Can_Add_RE({Manager, A1 building, Morning, Tester, A2, Afternoon)
5. Can_Add_PA({Engineer}, A1 building, Morning, Tester, Read_O2)
6. Can_Assign_UA({Engineer, -Tester}, A1, Afternoon, {Engineer, -Tester}, A2 building, Morning, Tester)

In check_role_admin, consider actions (6), the admin pre-condition is tuple ({Engineer, -Tester}, A1 building, Afternoon) and pretends that our current time slots

ts_{cur} is Afternoon. The check_role_admin ({Engineer, -Tester}, A1 building, Afternoon) returns true because there exists admin "Mary" satisfied [((Mary, A1 building) $\in$ UL) $\wedge$ (ts_{cur} = Afternoon) $\wedge$ ((Mary, Engineer, A1 building, Afternoon) $\in$ UA) $\wedge$ ((Engineer, A1 building, Afternoon) $\in$ RE) $\wedge$ [((Mary, Tester, A1 building, Afternoon) $\notin$ UA $\vee$ ((Tester, A2 building, Afternoon) $\notin$ RE)]. Since check_role_admin returns true, user Mary can perform corresponding administrative action (6). In this example, user Shan can activate both role Engineer and Tester within extents (A1 building, Afternoon) but Tester is a negative role in (6) so Shan is not allowed to perform administrative action (6). Similarly, the formula check_role_user (Bob; {Engineer, -Tester}, A2 building, Morning) returns true since it satisfied ((Bob, Engineer, A2 building, Morning) $\in$ UA $\wedge$ ((Bob, Tester, A2 building, Morning) $\notin$ UA). User Mary can assign a new role Tester for Bob in location "A2 building" at the interval "Morning".

Now, we consider the reachability problems in ASTRBAC and a safety attribute (Write_O1, A1 building, Afternoon) and our current timeslot ts_{cur} is Morning. We assume that the initial state is α_0 = (UA_0, RE_0, PE_0, PA_0). It is easy to check that user Alice can use actions (1) and (3), because UL_{Alice} = {Alice, A1 building} $\in$ UA can satisfy administrative condition $A_{rule(1)}$ = {Manager, A1 building, Morning} under RE = {Manager, A1 building, Morning}. After actions (1), our STRBAC state will change from α_0 to α_1 which adds new information to PA where $PA_1 = PA_0 \cup$ (Engineer, Write_O1) since Alice. Similarly, Alice can use action (3) to change our STRBAC state from α_1 to α_2 to adds new information to RE where $RE_2 = RE_1 \cup$ (Engineer, A1 building, Afternoon). In state α_2, when our current timeslot change ts_{cur} = Afternoon, user Shan can satisfy conditions [((Shan, A1 building) $\in$ UL) $\wedge$ (ts_{cur} = Afternoon) $\wedge$ ((Shan, Engineer, A1 building, Afternoon) $\in$ UA) $\wedge$ ((Engineer, A1 building, Afternoon) $\in$ RE) $\wedge$ ((Write_O1, A1 building, Afternoon) $\in$ PE) $\wedge$ ((Engineer, Write_O1) $\in$ PA)]. In the end, user Shan can Write_O1 in location A1 building and timeslot Afternoon, which violate our security attributes.

4 Implementation and Evaluation

The reachability problem analysis. In [30], the reachability problem analysis can be separated into two main parts: User-Role Reachability Analysis (URRA) and Permission-Role Reachability Analysis (PRRA). As seen in Example 1, in order to check the reachability problem of ASTRBAC, we need to check the both of them in the tuple (UA, RE) and (PE, PA). However, just like ASTRBAC, the tuple (PE, PA) of STRBAC are less likely to change as mentioned in Sect. 2. In our techniques, we will focus on implementing a tool to check the User-Role Reachability Analysis since the Permission-Role Reachability Analysis can be implemented similarly.

Implementing the Translator. We implement our technique which will be discussed below in a tool called $_{\text{ASASP}}$SPACETIME. As in Fig. 1 this tool has two main parts, the Translator, implemented in Python, will get the input of our ASTRBAC reachability problem (u; C; l; ts) as reachability problem for STRBAC policies (α_0; φ), answer our problem with statement "reachable" or "unreachable" and show the sequence of actions which changed our STRBAC policies from α_o to α_n where α_n can satisfy (u; C; l; ts). The second part for analysis of ASTRBAC policies uses SMT-based model checker named MCMT [25] to solve our problem. According to [28], we try to reduce our reachability problems for ASTRBAC model to a (finite) sequence of constraint satisfaction problems.

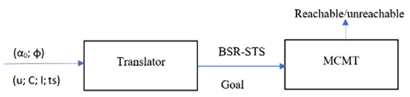

Fig. 1. Our technique to solve the reachability problem for ASTRBAC

At first, we translate ASTRBAC policies to First Order Logic formula which belongs to Bernays-Schonfinkel-Ramsey (BSR) [28] class to determine the satisfy-ability of formula, which has the form $\exists \underline{x}.\forall \underline{y}.\varphi(\underline{x}; \underline{y})$, where φ is a quantifier-free formula, $\underline{x}$ and $\underline{y}$ are (disjoint and possible empty) tuples of variable. Then, we use Model Checking Modulo Theories (MCMT) [25], which is a framework to solve reachability problems for infinite state systems that can be represented by transition systems whose set of states and transitions are encoded as constraints in First-Order Logic. MCMT framework uses a backward reachability procedure to solve a particular class of constraint satisfy-ability problems, called Satisfy-ability Modulo Theories (SMT) problems. According to [26], MCMT framework is a scalable and efficient SMT solver currently available.

Here is how we translate the ASTRBAC to First Order Logic in BSR class. We need to translate an initial state, the administrative actions, time passing, and the goal.

Our state variable in ASTRBAC contains re, ua, loc, and at where re is a variable of for the current role extent RE, similarly, ua for UA, loc for UL and at is current system time.

Our initial state contains the tuple α_0 = (RE, UA, UL, ts_0) where ts is timeslot, which can be translated as below.

$$\begin{aligned} \forall x, y, z, t.\ ua(x, y, z, t) &\Leftrightarrow \vee_{(u,r,l,ts) \in UA}(x = u \wedge y = r \wedge z = l \wedge t = ts) \wedge \\ re(y, z, t) &\Leftrightarrow \vee_{(r,l,ts) \in RE}(y = r, z = l, t = ts) \wedge \\ at(z) &\Leftrightarrow z = ts_o \end{aligned}$$

Example 2. We analyze a simple example of ASTRBAC

- Let U = {Alice; Bob; Peter}
- R = {Manager; Engineer; Technician; Tester; Developer}
- L = {A1 building; A2 building}
- I = {Morning: [8:00am – 12:00 pm]; Afternoon: (12:00 pm – 18:00 pm)}
- UA = {(Alice, Manager, A1 building, Morning); (Bob, Engineer, A2 building, Morning);}

- RE = {(Engineer, A1 building, Morning: [8:00am – 12:00pm]), (Technician, A2 building, Afternoon: [12:00pm – 18:00pm])}
- UL = {(Alice, A1 building); (Bob, A2 building)}
- Current time = 8am;

The ASTRBAC rule contains these rule.

1. Can_Assign_UA({Manager}, A1 building, Morning, {Engineer}, A2 building, Afternoon, Tester)
2. Can_Revoke_UA({Manager}, A1 building, Morning, {Engineer}, A2 building, Afternoon, Tester)
3. Can_Add_RE ({Manager}, A1 building, Morning, Engineer, A1 building, Afternoon)
4. Can_Delete_RE ({Manager}, A1 building, Morning, Engineer, A1 building, Afternoon)

According the example above, the current time belongs to time slots Morning, so our initial state will be

$$\begin{aligned}&\forall x, y, z, t.\ \mathrm{ua}(x, y, z, t) \Leftrightarrow \\ &((x = \text{Alice} \wedge y = \text{Manager} \wedge z = \text{A1 building} \wedge t = \text{Morning}) \vee (x = \text{Bob} \wedge y = \text{Engineer} \\ &\wedge z = \text{A2 building} \wedge t = \text{Afternoon}))\wedge \\ &\quad \mathrm{re}(y,\ z,\ t) \Leftrightarrow ((y\ = \text{Engineer},\ z = \text{A1 building},\ t = \text{Morning}) \vee (y\ = \text{Technician}, \\ &z = \text{A2 building},\ t = \text{Afternoon}))\wedge \\ &\quad \mathrm{at}(z) \Leftrightarrow z = \text{Morning}\end{aligned}$$

Our ASTRBAC now contains 4 actions (Can_Assign_UA, Can_Revoke_UA, Can_Add_RE, Can_Delete_RE) and a goal. We translate each of them as follow.

- Can_Assign_UA (A, l_a, $ts_{a,}$ R_u, l_u, ts_u, r_u)

$$\begin{aligned}&\Leftrightarrow \exists u_a,\ u,\ ts.\ \mathrm{at}(ts) \wedge ts\ =\ ts_a\wedge \\ &\quad \mathrm{Loc}(u_a,\ l_a)\wedge \\ &\qquad \textstyle\bigwedge_{+r\in A} \mathrm{re}(r,\ l_{a,}t_a)\wedge\wedge_{-r\in A}\neg \mathrm{re}(r,\ l_{a,}t_a)\wedge \\ &\qquad \textstyle\bigwedge_{+r\in A} \mathrm{ua}(u_a,\ r,\ l_{a,}t_a)\wedge\wedge_{-r\in A}\neg \mathrm{ua}(u_a,\ r,\ l_{a,}t_a)\wedge \\ &\qquad \textstyle\bigwedge_{+r\in R} \mathrm{ua}(u,\ r,\ l_{u,}t_u)\wedge\wedge_{-r\in R}\neg \mathrm{ua}(u,\ r,\ l_{u,}t_u)\wedge \\ &\qquad (\forall_{x,y,z,t}\, \mathrm{ua}'(x,\ y,\ z,\ t) \Leftrightarrow \mathrm{ua}(x,\ y,\ z,\ t) \vee (x\ =\ u \wedge y\ =\ r_u \wedge z\ =\ l_u \wedge t\ =\ ts_u))\wedge \\ &\qquad (\forall_{x,y,z,t}\, \mathrm{re}'(y,\ z,\ t) \Leftrightarrow \mathrm{re}(y,\ z,\ t))\wedge \\ &\qquad (\forall_{x,y,z,t}\, \mathrm{loc}'(x,\ z) \Leftrightarrow \mathrm{loc}(x,\ z))\end{aligned}$$

Example 3. According to Example 2, the Can_Assign_UA({Manager}, A1 building, Morning, {Engineer}, A2 building, Afternoon, Tester) can be translated as

$$\begin{aligned}
&\exists u_a, u, ts.\ at(ts)\\
&\quad at(ts) \wedge ts = Morning \wedge\\
&\quad Loc(Manager,\ A1\ building) \wedge\\
&\qquad re(Manager,\ A1\ building, Morning) \wedge\\
&\qquad ua(u_a,\ Manager,\ A1\ building, Morning) \wedge\\
&\qquad ua(u, Engineer,\ A2\ building, Afternoon) \wedge\\
&\qquad (\forall_{x,y,z,t} ua'(x,\ y,\ z,\ t) \Leftrightarrow ua(x,\ y,\ z,\ t) \vee (x = u \wedge y = Tester \wedge z = A2\ building \wedge t = Afternoon)) \wedge\\
&\qquad (\forall_{x,y,z,t} re'(y,\ z,\ t) \Leftrightarrow re(y,\ z,\ t)) \wedge\\
&\qquad (\forall_{x,y,z,t} loc'(x,\ z) \Leftrightarrow loc(x,\ z))
\end{aligned}$$

- Can_Revoke_UA (A, l_a, ts_a, R, l_u, ts_u, r_u)

$$\begin{aligned}
&\Leftrightarrow \exists u_a, u, ts.\ at(ts) \wedge ts = ts_a \wedge\\
&\quad Loc(u_a,\ l_a) \wedge\\
&\qquad \bigwedge\nolimits_{+r \in A} re(r,\ l_a, t_a) \wedge \wedge_{-r \in A} \neg re(r,\ l_a, t_a) \wedge\\
&\qquad \bigwedge\nolimits_{+r \in A} ua(u_a,\ r,\ l_a, t_a) \wedge \wedge_{-r \in A} \neg ua(u_a,\ r,\ l_a, t_a) \wedge\\
&\qquad \bigwedge\nolimits_{+r \in R} ua(u,\ r,\ l_u, t_u) \wedge \wedge_{-r \in R} \neg ua(u,\ r,\ l_u, t_u) \wedge\\
&\qquad (\forall_{x,y,z,t}\, ua'(x,\ y,\ z,\ t) \Leftrightarrow ua(x,\ y,\ z,\ t) \wedge \neg(x = u \wedge y = r_u \wedge z = l_u \wedge t = ts_u)) \wedge\\
&\qquad (\forall_{x,y,z,t}\, re'(y,\ z,\ t) \Leftrightarrow re(y,\ z,\ t)) \wedge\\
&\qquad (\forall_{x,y,z,t}\, loc'(x,\ z) \Leftrightarrow loc(x,\ z))
\end{aligned}$$

Example 4. According to Example 2, the Can_Revoke_UA({Manager}, A1 building, Morning, {Engineer}, A2 building, Afternoon, Tester) can be translated as

$$\begin{aligned}
&\exists u_a, u, ts.\ at(ts)\\
&\quad at(ts) \wedge ts = Morning \wedge\\
&\quad Loc(Manager,\ A1\ building) \wedge\\
&\qquad re(Manager, A1\ building,\ Morning) \wedge\\
&\qquad ua(u_a, Manager, A1\ building,\ Morning) \wedge\\
&\qquad ua(u, Engineer,\ A2\ building, Afternoon) \wedge\\
&\qquad (\forall_{x,y,z,t}\, ua'(x,\ y,\ z,\ t) \Leftrightarrow ua(x,\ y,\ z,\ t) \wedge \neg(x = u \wedge y = Tester \wedge z = A2\ building \wedge t = Afternoon)) \wedge\\
&\qquad (\forall_{x,y,z,t}\, re'(y,\ z,\ t) \Leftrightarrow re(y,\ z,\ t)) \wedge\\
&\qquad (\forall_{x,y,z,t}\, loc'(x,\ z) \Leftrightarrow loc(x,\ z))
\end{aligned}$$

- Can_Add_RE (A, l_a, ts_a, r_u, l_u, ts_u)

$$\begin{aligned}
&\Leftrightarrow \exists u_a,\ u,\ ts.\ at(ts) \wedge ts = ts_a \wedge\\
&\quad Loc(u_a,\ l_a) \wedge\\
&\qquad \bigwedge\nolimits_{+r \in A} re(r,\ l_a, t_a) \wedge \wedge_{-r \in A} \neg re(r,\ l_a, t_a) \wedge\\
&\qquad \bigwedge\nolimits_{+r \in A} ua(u_a,\ r,\ l_a, t_a) \wedge \wedge_{-r \in A} \neg ua(u_a,\ r,\ l_a, t_a) \wedge\\
&\qquad (\forall_{x,y,z,t}\, ua'(x,\ y,\ z,\ t) \Leftrightarrow ua(x,\ y,\ z,\ t) \wedge\\
&\qquad (\forall_{x,y,z,t}\, re'(y,\ z,\ t) \Leftrightarrow re(y,\ z,\ t)) \vee (y = r_u \wedge z = l_u \wedge t = ts_u)) \wedge\\
&\qquad (\forall_{x,y,z,t}\, loc'(x,\ z) \Leftrightarrow loc(x,\ z))
\end{aligned}$$

Example 5. According to Example 2, the Can_Add_RE ({Manager}, A1 building, Morning, Engineer, A1 building, Afternoon) can be translated as

$$\begin{aligned}&\exists u_a,\ u,\ ts.\ at(ts) \wedge ts = \text{Manager} \wedge\\ &\quad Loc(\text{A1 building},\ \text{Morning}) \wedge\\ &\quad\quad re(\text{Manager},\ \text{A1 building}, \text{Morning}) \wedge\\ &\quad\quad ua(u_a,\ \text{Manager},\ \text{A1 building}, \text{Morning}) \wedge\\ &\quad\quad (\forall_{x,yz,t}\, ua'(x,\ y,\ z,\ t) \Leftrightarrow ua(x,\ y,\ z,\ t) \wedge\\ &\quad\quad (\forall_{x,y,z,t}\, re'(y,\ z,\ t) \Leftrightarrow re(y,\ z,\ t)) \vee (y = \text{Engineer} \wedge z = \text{A1 building} \wedge t = \text{Afternoon})) \wedge\\ &\quad\quad (\forall_{x,y,z,t}\, loc'(x,\ z) \Leftrightarrow loc(x,\ z))\end{aligned}$$

- Can_Delete_RE (A, l_a, ts_a, r_u, l_u, ts_u)

$$\begin{aligned}&\Leftrightarrow \exists u_a,\ u,\ ts.\ at(ts) \wedge ts = ts_a \wedge\\ &\quad Loc(u_a,\ l_a) \wedge\\ &\quad\quad \bigwedge\nolimits_{+r\in A} re(r,\ l_a,\ t_a) \wedge \wedge_{-r\in A} \neg re(r,\ la,\ ta) \wedge\\ &\quad\quad \bigwedge\nolimits_{+r\in A} ua(ua,\ r,\ la,\ ta) \wedge \wedge_{-r\in A} \neg ua(ua,\ r,\ la,\ ta) \wedge\\ &\quad\quad (\forall_{x,y,z,t}\, ua'(x,\ y,\ z,\ t) \Leftrightarrow ua(x,\ y,\ z,\ t) \wedge\\ &\quad\quad (\forall_{x,y,z,t}\, re'(y,\ z,\ t) \Leftrightarrow re(y,\ z,\ t)) \wedge \neg(y = ru \wedge z = lu \wedge t = tsu)) \wedge\\ &\quad\quad (\forall_{x,y,z,t}\, loc'(x,\ z) \Leftrightarrow loc(x,\ z))\end{aligned}$$

Example 6. According to Example 2, the Can_Delete_RE ({Manager}, A1 building, Morning, Engineer, A1 building, Afternoon) can be translated as

$$\begin{aligned}&\exists u_a,\ u,\ ts.\ at(ts) \wedge ts = \text{Manager} \wedge\\ &\quad Loc(\text{A1 building},\ \text{Morning}) \wedge\\ &\quad\quad re(\text{Manager}, \text{A1 building}, \text{Morning}) \wedge\\ &\quad\quad ua(u_a,\ \text{Manager}, \text{A1 building}, \text{Morning}) \wedge\\ &\quad\quad (\forall_{x,y,z,t} ua'(x,\ y,\ z,\ t) \Leftrightarrow ua(x,\ y,\ z,\ t) \wedge\\ &\quad\quad (\forall_{x,y,z,t} re'(y,\ z,\ t) \Leftrightarrow re(y,\ z,\ t)) \wedge \neg(y = \text{Engineer} \wedge z = \text{A1 building} \wedge t = \text{Afternoon})) \wedge\\ &\quad\quad (\forall_{x,y,z,t} loc'(x,\ z) \Leftrightarrow loc(x,\ z))\end{aligned}$$

- Time passing: as mentioned in Sect. 3, the following formula means that every time the state of time change, we will move to the next time slots.

  If $j + 1 < T_{max}$, then

$$\left[\begin{array}{c} at\,(ts) \wedge ts = (j, j+1) \wedge \\ \forall y, z, t.re'(y,z,t)\,(re(y,z,t) \wedge \forall x,y,z,t.ua'(x,y,z,t)(ua(x,y,z,t) \wedge \\ \forall z.L'(z)\,(L(z) \wedge \forall t.at'(t)\,((t = (j+1, j+2)) \end{array}\right]$$

  Otherwise:

$$\left[\begin{array}{c} at\,(ts) \wedge ts = (j, j+1) \wedge \\ \forall y, z, t.re'(y,z,t)\,(re(y,z,t) \wedge \forall x,y,z,t.ua'(x,y,z,t)(ua(x,y,z,t) \wedge \\ \forall z.L'(z)\,(L(z) \wedge \forall t.at'(t)((t = (0,1)) \end{array}\right]$$

- Goal state:

$$\left(u_{g,} r_{g,} l_g,\ ts_g\right) \Leftrightarrow$$
$$\exists u,\ r,\ l,\ ts.\ re(r,\ l,\ ts) \wedge ua(u,\ r,\ l,\ ts) \wedge u = u_g \wedge r\ = r_g \wedge l = l_g \wedge ts = ts_g$$

Example 7. Our goal (Alice, Write_O1, A1 building, Afternoon) can be translated as

$$\exists u,\ r,\ l,\ ts.\ re(r,\ l,\ ts) \wedge ua(u,\ r,\ l,\ ts) \wedge u\ =\ \text{Alice} \wedge r\ =\ \text{Write\_O1} \wedge l\ =\ \text{A1 building} \wedge ts\ =\ \text{Afternoon}$$

After translating all the ASTRBAC policy to BSR, we need to do an AND operator on all the translated formula. If the result return true, then our goal is reachable, otherwise, it is unreachable. If the state is reachable, we know that our system is unsafe, otherwise, it is safe.

Evaluations. We use real scenario test cases synthesized from [27], which contains university and hospital benchmark, and are widely used in security analysis community. For our testing, we randomly adding temporal and spatial to the test cases. Since this is the first analysis tool of ASTRBAC model, we cannot compare it with any previous tool. All our experiments performs on an Intel Core i7 CPU with 4 GB Ram running Ubuntu 12.04 LTS 32 bit. We run 2 experiments with our test cases. Our first experiments run 15 times with different goals and calculate the average time for each test cases, the results is in Table 1. The first column shows our test case names; test cases 1 to 6 is our simple test cases created to test this program, test cases from 7 to 12 is taken from hospitals sets, test cases from 13 to 16 were taken from university sets. Each config contains 3 number representing max roles, max locations and max time slots in our STRBAC. The number of actions contain the number of administrative actions in ASTRBAC policies. We configure our experiment with maximum number of roles, locations and time slots.

Table 1. Our first experimental results

#	Testcase	Config	Number actions	Average runtime (sec)
1	Test1	3 3 3	4	7.58
2	Test2	3 3 3	5	1.53
3	Test3	3 3 3	7	318.40
4	Test4	3 3 3	4	2.48
5	Test5	3 3 3	5	4.05
6	Test6	3 3 3	6	13.14
7	AGTHos1_test	16 4 5	125	16.34
8	AGTHos2_test	16 4 5	131	87.67
9	AGTHos3_test	35 6 10	165	370.27
10	AGTHos4_test	35 6 10	283	106.29

(*continued*)

Table 1. (*continued*)

#	Testcase	Config	Number actions	Average runtime (sec)
11	AGTHos5_test	16 8 20	355	794.58
12	AGTHos6_test	16 8 20	398	913.67
13	AGTUniv1_test	35 4 5	146	251.47
14	AGTUniv2_test	35 4 5	188	317.48
15	AGTUniv3_test	35 6 10	209	367.51
16	AGTUniv4_test	35 6 10	246	480.78

In our second experiments, we use a simple test cases without spatial temporal constrains, set 16 for the maximum number of roles and 10 for the maximum number of time slots. Then, we run these test with randomly added locations each time to the test cases to get their average run time in Table 2 and Fig. 2.

After that, we create others test cases, set 16 for the maximum number of roles, set 5 for maximum number of locations. We run these test cases 5 times and add random time slots each time to the test cases to get their average run time shows in Table 3 and Fig. 3.

Table 2. Our second experimental results with different locations

#	Testcase	Config	Number of action	Average runtime (sec)
1	AGTHos1_l3	16 3 10	129	6.93
2	AGTHos1_l6	16 6 10	132	7.35
3	AGTHos1_l9	16 9 10	135	7.45
4	AGTHos1_l12	16 12 10	138	7.7
5	AGTHos1_l15	16 15 10	141	7.83
6	AGTHos1_l18	16 18 10	144	7.83

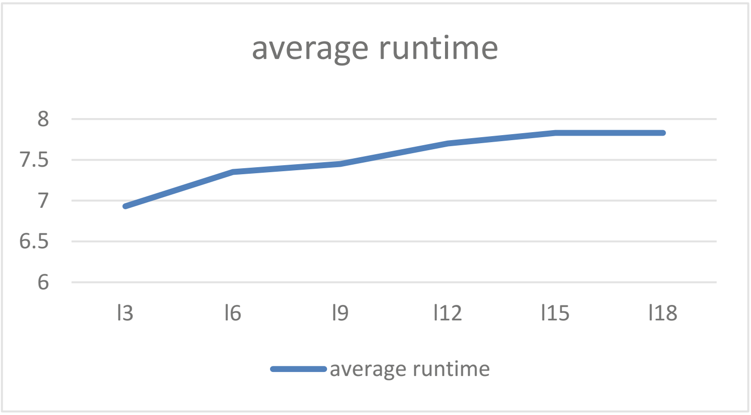

Fig. 2. Our average run time with different locations

Table 3. Our second experimental results with different time slots

#	Testcase	Config	Number of action	Average runtime (sec)
1	AGTHos1_t5	16 5 5	126	10.67
2	AGTHos1_t10	16 5 10	131	7.23
3	AGTHos1_t15	16 5 15	136	7.78
4	AGTHos1_t20	16 5 20	141	7.87
5	AGTHos1_t25	16 5 25	146	8.21
6	AGTHos1_t30	16 5 30	151	8.58

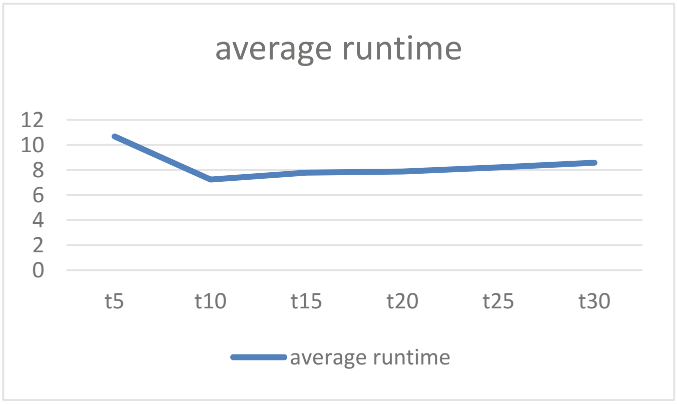

Fig. 3. Our average run time with different time slots

In our first experiments, we can assume that as the number of actions in ASTRBAC keep increasing, the runtime is affected and increased too. In our second experiments, we can assume that the increase in the number of time slots and the location does not affect the run time of our system. According to this result, we can conclude that more works need to be done for our solutions to get the answer faster when the number of actions keep increasing. We plan to improve this version in future works using some heuristics and functions supported in MCMT.

5 Conclusion

In this paper, we propose techniques using SMT-based model checker approach to solve the reachability problem in ASTRBAC. We also design a translation technique for ASTRBAC policies using First Order Logic.

As future works, we plan to design and apply heuristics to reduce the state exploration problem and speed up our approach in STRBAC by focusing only on the actions that may lead STRBAC to an unsafe state and using some functions provided in our SMT-based model checker such as the capability of tracking the visited states for later use.

Acknowledgements. This research is funded by Vietnam National University HoChiMinh City (VNU-HCM) under grant number C2017-20-17.

References

1. Samarati, P., Vimercati, S.: Access control policies, models, and mechanisms. In: FOSAD: International School on Foundations of Security Analysis and Design, pp. 137–196 (2000)
2. National Computer Security Center (NCSC): A Guide to Understanding Discretionary Access Control in Trusted System, Report NSCD-TG-003 Version1, 30 September 1987
3. Osborn, S.: Mandatory access control and role-based access control revisited. In: Proceedings of the 2nd ACM Workshop on Role-Based Access Control, RBAC 1997, pp. 31–40. ACM (1997)
4. Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role-based access control models. IEEE Comput. **29**(7), 38–47 (1996)
5. Ferraiolo, K.: Role-based access control. In: 15th National Computer Security Conference, pp. 554–563, October 1992
6. Sandhu, R., Ferraiolo, D., Kuhn, R.: The NIST model for role based access control: toward a unified standard. In: 5th ACM Workshop Role-Based Access Control, pp. 47–63, July 2000
7. Sandhu, R., Bhamidipati, V., Munawer, Q.: The ARBAC97 model for role-based administration of roles. ACM Trans. Inform. Syst. Secur. (TISSEC) **2**(1), 105–135 (1999)
8. Kumar, M., Newman, R.: STRBAC - an approach towards spatiotemporal role-based access control. In: Proceedings of the Third IASTED International Conference on Communication Network and Information Security CNIS, pp. 150–155 (2006)
9. Sharma, M., Sural, S., Atluri, V., Vaidya, J.: An administrative model for spatio-temporal role based access control. In: Bagchi, A., Ray, I. (eds.) ICISS 2013. LNCS, vol. 8303, pp. 375–389. Springer, Heidelberg (2013). doi:10.1007/978-3-642-45204-8_28
10. Li, N., Tripunitara, M.: Security analysis in role-based access control. In: The Proceedings of ACM Symposium on Access Control Models and Technologies, pp. 126–135. ACM Press (2004)
11. Jha, S., Li, N., Tripunitara, M., Wang, Q., Winsborough, H.: Towards formal verification of role-based access control policies. IEEE TDSC **5**(4), 242–255 (2008)
12. Gofman, M.I., Luo, R., Solomon, Ayla C., Zhang, Y., Yang, P., Stoller, Scott D.: RBAC-PAT: a policy analysis tool for role based access control. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 46–49. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00768-2_4
13. Jayaraman, K., Tripunitara, M., Ganesh, V., Rinard, M., Chapin, S.: Mohawk: abstraction-refinement and bound-estimation for verifying access control policies. ACM TISSEC **15**(4), 18 (2013)
14. Ferrara, A.L., Madhusudan, P., Nguyen, T.L., Parlato, G.: Vac - verifier of administrative role-based access control policies. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 184–191. Springer, Cham (2014). doi:10.1007/978-3-319-08867-9_12
15. Ranise, S., Truong, A., Vigano, L.: Automated analysis of RBAC policies with temporal constraints and static role hierarchies. In: The Proceeding of the 30th ACM Symposium on Applied Computing (SAC 2015), pp. 2177–2184. ACM (2015)
16. Ranise, S., Truong, A., Armando, A.: Scalable and precise automated analysis of administrative temporal role-based access control. In: Proceedings of the 19th ACM Symposium on Access Control Models and Technologies, pp. 103–114. ACM (2014)

17. Ranise, S., Truong, A.: ASASPXL new clother for analysing ARBAC policies. In: International Conference on Future Data and Security Engineering, FDSE (2016)
18. Ghilardi, S., Ranise, S.: MCMT: a model checker modulo theories. In: Giesl, J., Hähnle, R. (eds.) IJCAR 2010. LNCS, vol. 6173, pp. 22–29. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14203-1_3
19. Harrison, M., Ruzzo, W., Ullman, J.: Protection in operating systems. Commun. ACM **19**(8), 461–471 (1976)
20. Bertino, E., Bonatti, P., Ferrari, E.: TRBAC a temporal role based access control model. ACM TISSEC **4**(3), 191–233 (2001)
21. Joshi, J., Bertino, E., Latif, U., Ghafoor, A.: A generalized temporal role-based access control model. IEEE Trans. Knowl. Data Eng. **17**(1), 4–23 (2005)
22. Kumar, M., Newman, R.: STRBAC - an approach towards spatio-temporal role-based access control. In: Communication, Network, and Information Security, pp. 150–155 (2006)
23. Aich, S., Mondal, S., Sural, S., Majumdar, A.: Role based access control with spatio-temporal context for mobile applications. Trans. Comput. Sci. **IV**, 177–199 (2009)
24. Uzun, E., Atluri, V., Sural, S., Vaidya, J., Parlato, G., Ferrara, A.: Analyzing temporal role based access control models. In: SACMAT, pp. 177–186. ACM (2012)
25. Ghilardi, S., Ranise, S.: Backward reachability of array-based systems by SMT solving termination and invariant synthesis. Logical Methods Comput. Sci. LMCS **6**(4), 1–48 (2010)
26. http://research.microsoft.com/en-us/um/redmond/projects/z3
27. Ranise, S., Truong, A., Armando, A.: Scalable and precise automated analysis of administrative temporal role-based access control, pp. 103–114 (2014)
28. Ranise, S.: Symbolic backward reachability with effectively propositional logic: applications to security policy analysis. FMSD **42**(1), 24–45 (2013)
29. Piskac, R., Moura, L., Bjørner, N.: Deciding effectively propositional logic using DPLL and substitution sets. J. Autom. Reasoning **44**(4), 401–424 (2010)
30. Sasturkar, A., Yang, A., Stoller, S., Ramakrishnan, C.: Policy analysis for administrative role based access control. In: 19th IEEE Computer Security Foundations Workshop, pp. 124–138 (2006)