Article

Permission-Based Separation of Duty in Dynamic Role-Based Access Control Model

Muhammad Umar Aftab 1,*; Zhiguang Qin 1,*; Negalign Wake Hundera 1; Oluwasanmi Ariyo 1; Zakria 1; Ngo Tung Son 2* and Tran Van Dinh 3

1 School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China; nigaccna21@gmail.com (N.W.H.); ariyo@std.uestc.edu.cn (O.A.);
zakria.uestc@hotmail.com (Z.)
2 Computing Fundamental Department, FPT University, Hanoi 10000, Vietnam; sonnt69@fe.edu.vn
3 Department of Computer Science, University of Freiburg, 79098 Freiburg, Germany;
dinh@informatik.uni-freiburg.de
* Correspondence: ms.umaraftab@yahoo.com (M.U.A.); qinzg@uestc.edu.cn (Z.Q.)

Received: 8 April 2019; Accepted: 9 May 2019; Published: 15 May 2019

Abstract: A major development in the field of access control is the dominant role-based access control (RBAC) scheme. The fascination of RBAC lies in its enhanced security along with the concept of roles. In addition, attribute-based access control (ABAC) is added to the access control models, which is famous for its dynamic behavior. Separation of duty (SOD) is used for enforcing least privilege concept in RBAC and ABAC. Moreover, SOD is a powerful tool that is used to protect an organization from internal security attacks and threats. Different problems have been found in the implementation of SOD at the role level. This paper discusses that the implementation of SOD on the level of roles is not a good option. Therefore, this paper proposes a hybrid access control model to implement SOD on the basis of permissions. The first part of the proposed model is based on the addition of attributes with dynamic characteristics in the RBAC model, whereas the second part of the model implements the permission-based SOD in dynamic RBAC model. Moreover, in comparison with previous models, performance and feature analysis are performed to show the strength of dynamic RBAC model. This model improves the performance of the RBAC model in terms of time, dynamicity, and automatic permissions and roles assignment. At the same time, this model also reduces the administrator’s load and provides a flexible, dynamic, and secure access control model.

Keywords: Efficient SOD; Dynamic RBAC; Hybrid Access Control; Attributed RBAC; Permission based SOD

1. Introduction

One of the building blocks of information and network security is the access control that is used to give or revoke access to the resources of the organization [1]. Access control cannot preclude the happening of cyber-attacks; however, there are many access control models and protection schemes that are used for the right enforcement of particular behavior in the organization [2]. A glimpse of the practical environment where a user accesses resources through access control mechanism is shown in Figure 1. Furthermore, the network administrator depends on policy-based models for the management of access control because the targeted network systems are complex [3]. There are various models used for implementing access control and enforcing security in an organization like discretionary access control (DAC) [4,5], mandatory access control (MAC) [1,6], access control lists (ACL) [7], role-based access control (RBAC) [8,9], and attribute-based access control (ABAC) [10,11]. In this paper, a hybrid model is proposed, merging RBAC and ABAC.
RBAC is a significant revolution in the field of access control, due to its security and access control implementation. Roles are habitually used to differentiate the authorization of users in RBAC. In addition, a role is a vital entity that is placed between permissions and users in the RBAC model [8,12]. The roles are used for the management of permissions. To the best of the authors’ knowledge, previously, access control models have not offered any mechanism for managing the permissions. The concept of role is similar to the use of folders in computers. The users create folders for managing different files on the computer. Similarly, administrators create different roles for managing the permissions in RBAC. A permission is the combination of object and action. Objects are the resources of the system such as files, folders, and systems. On the other hand, actions are the operations that a user can perform on objects, such as read, write, and delete [13].

In the RBAC model, permissions are allocated to roles, and roles are associated with users. RBAC is used in many organizations as a secure framework, as it protects their information from attacks and threats [14]. The key inadequacy of the DAC, MAC, and RBAC is their static behavior as well as that they are used to take decisions on the user identity. This type of dependency on user’s identity makes such models inappropriate for dynamic behavior [15]. More specifically about RBAC, it has complexity problems regarding designing access control systems and role structuring for a particular organization [16]. Moreover, the complexity can also be increased in the RBAC systems when the number of users increases [17]. In recent years, a dynamic access control model has been proposed that is ABAC [18–20]. The authorization and access decision are used to take on the concept of attributes in ABAC model. In the ABAC system, attributes are used to describe a specific characteristic of an entity. For example, the attributes of a user are name, user_id, and location. Similarly, there are a set of attributes and a well-defined set of values against every attribute in the ABAC system. For instance, name = ‘Bob’, user_id = 123456, location = ‘England’ where name, user_id, and location are the set of attributes for a user and ‘Bob’, 123456, and ‘England’, are the values for the attributes, respectively [15].

Separation of duty (SOD) is a prevailing restraint that helps to apply the idea that illustrates avoiding one-person control and least privileged [21]. SOD in RBAC standard [9] depends on the number of conflicting roles [22], whereby a role that contains one or more conflicting permissions is called a conflicting role. Roles that are conflicting with others are also declared as mutually exclusive roles (MER). Moreover, all the permissions that belong to the MER will also be in mutual exclusion (ME). If we dive deeper into permissions and roles, it comes to our understanding that the roles designed for the access control do not initiate a conflict of interest (COI) within themselves, but the permissions make the roles to be in COI amongst them. In fact, all the permissions of MER do not create COI with all the permissions of other MER. Normally, there are a small number of permissions from MER that create a COI with little consent from other MER.
If a permission has a COI with another permission, then both permissions are called mutually exclusive permissions (MEP). For example, permission one is “submit application” and permission two is “approve application”. Permission one and two both have a COI with each other because a single user cannot activate both permissions at a time. The reason is the question of how an employee or student can submit an application as well as approve their application at the same time. Thus, if a student or employee activates or submits an application, then they cannot approve their application. In typical SOD, the occurrence of single conflicting permission makes the entire role a conflicting role. In addition, all the permissions in an MER are supposed to be present in ME [23]. Thus, those particular permissions are recommended to state as MEPs that truly create COI instead of declaring all the permissions as MEP. Therefore, the implementation should be made on the level of permissions rather than on the level of roles in SOD [9]. However, different problems may occur if we implement SOD on the level of roles that are discussed in Section 3.

There are two main contributions of this paper; the first contribution is adding attributes in the typical RBAC model that makes it dynamic RBAC model. In this way, the assignment of permissions to roles and roles to users is automatic. In addition, the declaration and assignment of conflicting and non-conflicting permissions are also performed with the help of attributes. Previously, these permissions were declared and assigned manually in a typical RBAC model. In addition, the attributed RBAC model [24] is unable to deal with SOD. The second contribution is the implementation of SOD on the level of permissions instead of the level of roles. In this way, the authority level of end users remains the same and the system will not violate SOD in our proposed model. Previously, permission-based SOD was implemented in static RBAC model [23]; we implemented the SOD in the dynamic RBAC model.

This paper is divided into five sections. The first section is an introduction of the paper and the concepts that are necessary to know for understanding the proposed model. The second section contains related work to our proposed model. The third section contains a discussion of the flaws in access control and violation of SOD in RBAC. The fourth and important section is the proposed work that contains the hybrid model including the section-wise discussion of this work. The fifth section is based on the implementation of the model as well as comparative analysis. Finally, section six forms a conclusion and discusses the future direction of the paper.

2. Related Work

Several contributions have been found in the field of hybrid access control, merging RBAC and ABAC, ease of administration in access control and removal of problems in RBAC. The main contribution of the paper is related to the merger of RBAC and ABAC model then implements SOD on the permission level in that dynamic RBAC. In this way, we categorize our related work in different sections like a merger of ABAC and RBAC, SOD implementation in access control, and hybrid access control.

2.1. RBAC and ABAC Merger

Several approaches have been proposed by different researchers for merging RBAC and ABAC. The main contribution in this field is the addition of attributes in the RBAC model. One of the luring approaches is the assignment of users to roles dynamically. The dynamic allocation is based on the users and roles attributes. Researchers have given the concept of users and roles attributes in traditional RBAC model [25]. However, the model was limited to users and roles automatic assignment with the help of attributes. The concept of attributes was only introduced between users and roles.

Moreover, the merger is possible in three ways: dynamic roles, attribute-centric, and role-centric but the only approach dynamic roles are suitable for the merger of RBAC and ABAC [16]. Authors conceptually proposed three approaches without implementation. In fact, an attributed RBAC model was proposed that works on the principle of the RBAC model. However, the permissions assigned to roles and users assignment to roles is based on the attributes. This type of assignment makes it a
dynamic RBAC. The automatic assignment of permissions and users to roles save time of administrator and decreases the load as well [24]. However, the model did not support the SOD and role hierarchies. In addition, the permission creation process is rapid but only for some cases. For example, if administrator wants to create one permission, the process will take more time as compared to the typical RBAC model. In this way, the model had no solution to deal with this problem. There are more examples of RBAC and ABAC merger as well as RBAC with other systems in which the researcher proposed some fascinating techniques in the form of hybrid models [10,11,26,27].

2.2. SOD Implementation in Access Control Models

The SOD implementation in access control models has fascinated extensive research interest in recent years. In [28], the researcher discussed an object based dynamic separation of duty (DSD) in RBAC under the observations but they did not propose any model and its implementation. Moreover, researchers have discussed the problems of the RBAC standard [9] related to DSD and elaborated the benefits of the proposed definition of object-based DSD in RBAC [28]. In addition, researchers have also proposed a model that implemented the DSD on the level of permissions rather than roles, in traditional RBAC model [23]. However, the model was not dynamic and implemented in typical RBAC model. Therefore, the load of the administrator was not focused and the work was manual. However, in [29], researchers analyzed the problems and complexity of SOD implementation in ABAC systems. In fact, the authors provided the solution to the problem with encouraging results. Moreover, SOD was discussed in the context of ABAC, but not in terms of RBAC. RBAC is considered more secure than ABAC due to the concept of roles. Some more examples can be seen in the implementation of SOD in access control models. For example, the SOD implementation in ABAC, temporal RBAC model, and novel approach for SOD in MAC models [15,30,31].

2.3. Hybrid Access Control

In the past three decades, many access control models have been proposed and developed. In this era, the trend is changed. The focus of authors is to work on the hybrid access control so that the efficiency of the previously proposed models can be improved. In this way, the researchers are achieving significant results from the hybrid models [12–14,17,20]. More specifically, fine-grained access control models are proposed for data encryption using attribute-based encryption schemes. The proposed models are more reliable, efficient, trustworthy, and ensure secure keyword searching. In addition, their comparison with traditional schemes is also given, which shows that the hybrid schemes give better results [32,33]. Furthermore, machine learning techniques are proposed to automate the traditional RBAC model. The role assignment is accomplished using the supervisory control and data acquisition primitive [34]. The high-level security can be achieved in fog computing using the internet of medical things. The proposed model is efficient for cloud-based systems. The novelty is the addition of an extra layer of fog server that implements an access control mechanism in it [35].

In RBAC, administration becomes easier with the conception of roles. On the contrary, access control implementation on the concept of roles also arises several difficulties that are discussed in the following section.

3. Access Control Flaws

Operations of SOD on the level of roles make the schemes safer from internal security threats [9]. However, it generates numerous difficulties for the users of RBAC that includes end users and security administrators. Additionally, it also disturbs and violates safety by the ways of authorization. Furthermore, there are some problems in the RBAC and ABAC models that can be solved by merging both of these models. In the following section, these problems are discussed in detail.

3.1. The Decrease in RBAC User’s Authority Domain

When a role is professed as MER to apply SOD, all the permissions are affected because all permissions become MEP, as given in RBAC standard [9]. In this way, non-conflicting permissions in
an MER will also start acting like conflicting permissions. Thus, according to the stated method above, users who are approved for the authorization of non-contradictory permissions will not be permitted to access those permissions because of being members of MER. In this way, the contradiction arises and users are unable to access the authorized permissions. The permissions were previously available for the users, but when a role becomes MER, all the permissions become MEP. In this way, the user’s authority is disturbed. This evidently demonstrates that this is an unreasoned and pointless degrade in the expert domain of RBAC operators, as well as affecting user’s authority domain. On the one hand, SOD is implemented to evade one-man control and to lessen the frauds increasing channels. On the other hand, the authority territory of RBAC users is reduced, with the implementation of SOD on the level of roles. Reducing the authority domain means the access is denied for authorized users. As such, this is not an ideal solution [23].

3.2. RBAC End-Users Violation in SOD

According to the definition and concept of SOD in RBAC standard [9], a user can activate only one of two MERs at the same time or in a similar session. Ultimately, whenever an operator desires to activate other MER besides the one being operated, the user only needs to quit his formerly initiated role. As a result, the user will be terminating the current session and logging again with the new session ID. Moreover, the user would be prepared to activate other MER. If we look at the SOD definition, we come to recognize that SOD is imposed to avoid COI. In this way, the user can activate both conflicted roles just by changing the session ID. This is a violation of the SOD from the end user’s side [23].

3.3. Security Administrators Violation in SOD

In the above subsection, we established that the SOD can be violated by the end user as well as by the security administrator accidentally and unintentionally. As the RBAC standard [9] implements SOD on the level of roles so the security administrator observes all of the MERs to take a custodian eye on the contradictory consents. Furthermore, there is no mechanism for keeping information about conflicting permissions. Therefore, in the stated respect, it may be probable that the security administrator assigns two conflicting permissions to the same role accidently due to lack of enough information of conflicting permissions. In this way, the users who will be assigned with such a role can violate SOD by activating and accessing the two MEPs. Therefore, this shows that SOD can be desecrated by security administrators and users unintentionally [23].

3.4. Problems in RBAC and ABAC models

RBAC model popularity is based on its tight security, but the working criteria are based on the manual assignment of permissions and users to roles. Moreover, role structuring is a difficult task for the administrator. In this way, the load of the administrator is much more as well as the chances of the human errors are there. On the other hand, the ABAC model is a dynamic model. The system dynamically deploys access control by using attributes. However, the ABAC model is unable to provide the ease of management for the administrator like the RBAC model. In addition, ABAC model is unable to provide the least privilege principle. The merger of RBAC and ABAC can overcome their drawbacks and provide benefits of both models [11,24,36].

4. Proposed Model

4.1. Overview

The proposed model is an efficient way to avoid the previously stated problems and difficulties. In this paper, authors investigated how to efficiently implement the SOD on the level of permissions in dynamic RBAC model. Previously, SOD was implemented on the basis of permissions rather than roles [23]; however, the implementation was done in the typical and static RBAC model. In addition, an attributed RBAC model was proposed [24]; however, the model is good only for the basic RBAC
working. The proposed model implements the permission-based SOD in dynamic RBAC. The proposed model implementation is based on two parts. In the first part, the dynamic RBAC is developed and the working of the model starts with the basic elements of RBAC like objects, actions, users, roles, and permissions. In the second part, the SOD is implemented on the level of permissions in dynamic RBAC model. The two parts are briefly elaborated in the following subsections.

4.2. Dynamic RBAC Model

In the dynamic RBAC model, we make the model more flexible as compared to the previously proposed model [24]. The basic entities of RBAC are objects, roles, and users contain attributes. This means that when the administrator creates objects, roles, and users, they will also assign attributes to these entities. Thus, the entities will become attributed entities such as attributed objects, attributed roles, and attributed users. Now, we explain the dynamic RBAC model from the permission creation level towards permission assignment to roles and roles assignment to users. In the start, objects are created with the assignment of attributes. For example, an object (obj1) is created with the attributes: time = 9:00 am to 6:00 pm, IP = 192.168.1.10, day = Monday to Friday. In this way, this object is only accessible if the user’s attributes match with the set of attributes as well as their values. Furthermore, objects are assigned to different categorize-containers. The concept of categorize-container is to contain the objects of the same or relevant department. For example, an administrator creates some objects for the finance department, objects for the lecturers of the computer science department, and objects for the students of the same class. In this way, the administrator can create three different categorize-containers so that the administrator can assign the necessary objects for a specific category to the same container.

The benefit of categorize-container is to create multiple permissions for the same category on a single click. After the creation of attributed objects and categorize-containers, actions will be created and assigned to a generic action set. For example, different action sets are designed for enforcing the level of security; action set 1 contains three actions (read, write, edit), action set 2 contains four actions set (read, write, edit, delete), and action set 3 contains one action (read). The administrator can define many action sets from the action list and design the action sets according to the need of the organization. The concept of generic action set and categorize-containers is to create multiple permissions on a single click and more efficiently as compared to typical RBAC standard [9]. More specifically, the dynamic RBAC model is more flexible as compared to the typical RBAC and attributed RBAC models. The reason for flexibility is the permission creation process in four different ways.

Firstly, the administrator can create permissions by applying the actions on objects for creating the permissions one by one, as the typical RBAC does. Secondly, the administrator can apply actions on categorize-containers for creating multiple permissions and all the permissions have the same action with different objects. Thirdly, the administrator can apply generic action set on the objects for creating multiple permissions and all the permissions have the same object with different actions. Lastly, the administrator can apply generic action set on the categorize-containers for creating multiple permissions at once. For example, if the generic action set has four actions and the categorize-container has five objects, the merger of these containers will create 20 permissions ($4 \times 5 = 20$). The reason for this feature is to reduce the complexity as well as making the system more flexible and time efficient.

The administrator can create permissions in different ways according to the situation. In some cases, the administrator wants to create one permission; thus, by merging one object and one action, the administrator can fulfill the requirement. In this way, if an administrator has no choice to apply object and action directly then they will assign the object to categorize-container and action to generic action set. After that, the administrator can create permission, like the previously proposed hybrid model. However, this is not an efficient approach. A system should be flexible enough so that the administrator can perform their duties efficiently. Hence, the proposed model reduces the efforts of the administrator by creating a link between objects and actions. In this way, the administrator can create a single permission by directly applying the action to the object, in special cases. Moreover, if there is only one object, while more than one action is needed to apply for creating permissions, the
administrator can apply a generic action set to the objects. On the other hand, if there is more than one object and the administrator wants to apply the same action on all objects, the administrator can apply action on the categorize-containers for creating multiple permissions with common action. In this way, the administrator can create a variety of permissions in less time as compared to the traditional RBAC model. The dynamic RBAC model is given below in Figure 2 for a better understanding of the model.

![Figure 2. Dynamic role-based access control (RBAC) model with the addition of attributes.](image)

When the permissions are created by applying the actions and generic action sets on attributed objects and categorize-containers, created permissions are also attributed. The permissions attributes will be the same as the object attributes because permission is the combination of object and action. Thus, the attributed objects will create attributed permissions. The attributed permissions automatically assigned to the attributed roles. The system automatically assigns the attributed permissions to attributed roles by matching their attributes. If the attributes of permissions matched with the role’s attributes, then the permissions will be assigned to those specified roles automatically, with the help of attributes. For instance, permissions P1, P2, and P3 have attributes: time = 9:00 am to 6:00 pm and IP = 192.168.1.10. Similarly, there are two roles R1 and R2. R1 has attributes: time = 9:00 am to 6:00 pm and IP = 192.168.1.10.

On the other hand, R2 has attributes time = 9:00 am to 5:00 pm and IP = 192.168.1.10. System matches the attributes of permissions and roles. The system automatically assigns the P1, P2, P3 to R1 only because the attributes of permissions are only matched with R1 and no permission is granted to R2. Moreover, roles assignment to users will also be done on the basis of attributes. The matching criteria between the users and roles will also be the attributes of roles and users. The system matched the attributes of the users and roles. This is to ensure that the roles can be assigned to users that have the same attributes with users’ attributes, as shown in Figure 2. For example, users U1 and U2 have attributes time = 10:00 am to 2:00 pm and IP = 192.168.2.11. There are two additional roles: R4 and R5. R4 has attributes time = 9:00 am to 6:00 pm and IP = 192.168.1.10. In addition, R5 has attributes time = 10:00 am to 2:00 pm and IP = 192.168.2.11. Therefore, the system will grant access to U1 and U2 based
on role R5 because their attributes and attribute values are matched. On the other hand, users U1 and U2 are not authorized to access role R4 because their attributes and attribute values are not matched.

4.3. Permission-based SOD

In this subsection, SOD is implemented on the level of permissions instead of the level of roles. In permission-based RBAC, a user can access more than one conflicted roles, however the user cannot access two permissions that have a conflict of interest with each other. In this part of the model, the process again starts with action and object creation. In addition, the actions are created with the addition of COI with other actions. For example, an action ‘write’ has a conflict with another action ‘evaluate’, and an action ‘submit’ has a conflict with another action ‘approve’. This means that if a student or employee submits their leave application, then they cannot approve their leave application and vice versa. Moreover, if a student writes an assignment or exam paper then they cannot evaluate their assignment or exam paper. In this way, the administrator creates the actions and specifies their COI with other actions. Thus, a user cannot try to execute two conflicted actions on the same object. It is necessary to know the concept of COI between two actions and permissions. If two actions that have COI with each other are applied on the same object, they create two conflicted permissions that also have COI with each other. In this way, a user cannot access both permissions at the same time. Moving forward on the proposed model working, the actions are applied to objects and permissions are created with two categories that are conflicted and non-conflicted permissions, as shown in Figure 3.

![Figure 3. Permission based SOD with Conflicted and Non-Conflicted Actions.](image)

There are different categories of actions, namely action with COI and actions with no COI. Similarly, the permissions can be conflicted and non-conflicted because the conflict is raised in permission on the basis of actions. For example, the same object (Obj5) in two permissions P10 and P20 with two different actions submit and approve, causing the two permissions P10 and P20 to be conflicted with each other due to the COI between actions. Thus, a user can only access one permission from P10 and P20. The permissions in blue color with names P1, P3, and P5 are non-conflicted permissions and permissions in light orange color with names P2, P4, and P6 are conflicted permissions, as shown in Figure 3.
After the creation of conflicted and non-conflicted permissions, the permissions are assigned to different roles. Previously, in RBAC standard, if a role contains a conflicted permission, that role will be considered the conflicted role. Moreover, a user can only access one conflicted role from a set of two conflicted roles. For instance, there are two conflicted roles R6 and R7 that have COI with each other. The administrator grants access to the user on both roles R6 and R7. If the user accesses conflicted role R6, the user is unable to access the second conflicted role R7. In this way, the user authority is decreased due to the COI between R6 and R7. In this model, we implement the SOD on the level of permissions instead of roles. Thus, a user can access two conflicted roles, but they can only access one conflicted permission from the set of two. Therefore, the user’s authority domain is not decreased in our model. Moreover, the system implements the least amount of privileges in an efficient manner.

Example Scenario

Consider that the permissions are assigned to different roles and roles are assigned to users. The users can access different roles and the permissions inside them, as per the user’s authority domain. There are four roles with names role 1, role 2, role 3, and role 4; these are the containers that are used to contain several permissions. These roles are usually assigned to users after assigning them permissions. These roles can be assigned to a teacher, student, manager, etc. For example, role 1 is designed for students and teacher assistants, role 2 is designed for lecturers, role 3 is planned for managers and admin staff, and role 4 is designed for the deans. After designing these roles administrator assigns them to particular users. This is just an example of the clarity of the role’s concept. Role 1 contains five non-conflicted permissions (P1, P3, P5, P7, and P9). Role 2 also contains five permissions but three of them are conflicted (P2, P4, and P6), and two non-conflicted permissions (P11, and P13), as shown in Table 1. Role 3 contains five permissions, from which four permissions are conflicted (P8, P10, P12, and P14) and P15 is the only non-conflicted permission. Role 4 contains four conflicted permissions (P16, P18, P20, and P22), as given in Table 1.

| Role Name | Total Permissions | Conflicted Permissions | Non-Conflicted Permissions | Assigned to Users |
|-----------|-------------------|------------------------|-----------------------------|------------------|
| Role 1    | Five              | None                   | P1, P3, P5, P7, P9          | User 1, User 2   |
| Role 2    | Five              | P2, P4, P6             | P11, P13                    | User 1, User 3, User 4 |
| Role 3    | Five              | P8, P10, P12, P14      | P15                         | User 2, User 5, User 6, User 7 |
| Role 4    | Four              | P16, P18, P20, P22     | None                        | User 4, User 6, User 7, User 8 |

1 The alphabet P is used for the permissions (conflicted and non-conflicted). Moreover, if a role has no conflicted or non-conflicted permission, ‘None’ is written in that column for that particular role. This is the case for role 1 in the conflicted permission column and for role 4 in the non-conflicted permission column.

The conflicted permissions are given a light orange color as well as the even number for a better understanding of the researchers. On the other hand, non-conflicted permissions are highlighted with blue color and assign odd numbers, as shown in Figure 4a,b. In addition, role 1 is assigned to user 1, and user 2. Role 2 is assigned to user 1, user 3, and user 4, as shown in Table 1 and Figure 4a. Role 3 is assigned to user 2, user 5, user 6, and user 7. Role 4 is assigned to user 4, user 6, user 7, and user 8, as shown in Table 1 and Figure 4a.
Moreover, the COI between conflicted permissions with other conflicted permissions is given in Table 2. For example, the permission P2 has COI with permissions P12 and P22. Similarly, permission P4 from role 2 has COI with permission P14 from role 3. The complete list of conflicted permissions with their COI can be viewed from Table 2. A user can only access one permission from the set of two

Figure 4. Roles assignment to users and users’ access on conflicting permissions.
to three permissions that have COI with each other. If a user accesses a conflicted permission from the list, that user is unable to access the other permission that has COI with the permission accessed first. This means that if a user accesses permission P8, the user cannot access permission P18 because both permissions have COI with each other, according to the list in Table 2.

Table 2. Conflicted Permissions with their conflict of interest (COI) with other Conflicted Permissions.

| Role Name | Permission Name | COI with Permission | COI Permission’s Role |
|-----------|-----------------|---------------------|-----------------------|
| Role 2    | P2              | P12, P22            | Role 3, Role 4        |
| Role 2    | P4              | P14                 | Role 3                |
| Role 2    | P6              | P16                 | Role 4                |
| Role 3    | P8              | P18                 | Role 4                |
| Role 3    | P10             | P20                 | Role 4                |
| Role 3    | P12             | P2                  | Role 2                |
| Role 3    | P14             | P4                  | Role 2                |
| Role 4    | P16             | P6                  | Role 2                |
| Role 4    | P18             | P8                  | Role 3                |
| Role 4    | P20             | P10                 | Role 3                |
| Role 4    | P22             | P2                  | Role 2                |

2 $P$ is used as a variable for the conflicting permissions. COI with permission is a column that contains all conflicted permissions that have a COI with the other permissions of the second column (permission name). P12 permission belongs to role 3 and P22 permission belongs to role 4.

According to the model, a user can access two conflicting roles at a time but the user is unable to access two conflicting permissions at a time. As the user 6, can access role 3 and role 4 but they can only access one conflicting permission from the two permissions that have COI with each other. The decision always depends upon the first access of the user. According to the given scenario, user 6 accesses the conflicted permissions P8 and P10 first, as shown in Figure 4b. When they try to access the conflicted permissions P18 and P20, they will receive the message “access denied”, because P8 and P10 have COI with P18 and P20. If user 6 accesses permissions P18 and P20 first, then user 6 cannot access the permissions P8 and P10. The same happens if user 7 accesses the permissions P18 and P20. Now, user 7 is unable to access the permissions P8 and P10, as shown in Figure 4b.

The same access mechanism is applied for all users against their access towards conflicting permissions. The users can only access one conflicting permission from the set of two to three permissions that have COI, according to the list provided in Table 2. Meanwhile, there is no restriction on the non-conflicting permissions. The users can freely access all the non-conflicting permissions from the assigned roles. In this way, the user’s authority domain remains the same and SOD is also implemented in an efficient manner. Moreover, there is no violation from the administrator side because SOD is implemented on the level of permissions instead of roles.

4.4. Permission-based SOD in Dynamic RBAC Model

After explaining both parts of the model in detail, now we merge the previously explained parts of the model and discuss the model as a complete dynamic RBAC model that implements permission-based SOD. The process of building up this model starts with objects and actions creation. Moreover, attributes are added with objects to make attributed objects and conflicted actions information, while creating new actions. This is to ensure the system can maintain the record of COI between actions, as shown in Figure 5. The attributed objects are assigned to categorize-containers according to the departmental categorization. Moreover, there is a possibility that some of the objects are not assigned to any categorize-container and administrator directly create permissions by applying actions on objects. On the other hand, actions with and without COI are assigned to generic action sets. Furthermore, the option still exists that the attributed objects and actions with and without COI are not assigned to any container. Then, the administrator can directly use actions and objects for permission creation. After
the creation of objects and actions as well as their assignment to containers, the process of permission creation will be started.

Figure 5. Permission based SOD implementation in Dynamic RBAC Model.

The administrator can create permissions in four ways. First, the administrator can apply the actions with and without COI, directly to attributed objects to create permissions. In this way, the permission creation process is just like the typical RBAC model. It means the administrator can create a single permission at a time. Second, the administrator can apply the actions with and without COI, to categorize-containers to create more than one permission with the same action name. This type of creation is different from the previously proposed models. Third, the administrator can apply generic action set on the attributed objects to create multiple permissions with the same object and different actions. This type of permission creation is also not available in the previously proposed models. Finally, the administrator can apply generic action set on the categorize-containers to create multiple permissions at a time. The permissions created from all the ways are attributed-conflicted permissions and attributed-non-conflicted permissions, as shown in Figure 5. These type of customize permissions were not introduced previously.

The administrator creates roles and users with different attributes. In the next step, the attributed conflicted and non-conflicted permissions are automatically assigned to attributed roles because of the attributes criteria between roles and permissions. All the permissions are automatically moved to particular roles that have the same attributes, similar to the conflicted and non-conflicted permissions. In the final step, the attributed roles are assigned to attributed users with the help of attributes. This process will also be done automatically without any interference from the administrator, as shown in Figure 5. The attributed users have access to the roles that have the same attributes as the attributed
users have. The users can access roles as well as permissions inside them. A user can freely access all the non-conflicting permissions from the assigned roles and perform the daily routine tasks in the organization. In addition, a user can only access one conflicting permissions from the set of two to three permissions that have COI. For example, when a user accesses conflicting permission A, they cannot access conflicting permission B because permission A and B have COI with each other. If they access permission B first, they cannot access the permission A due to COI.

The proposed model implements the SOD on the level of permissions. Users can access different permissions; however, users can only access one conflicting permission. Thus, the SOD is not violated in our proposed model as well as the user’s authority domain is also not compromised. Moreover, the proposed model saves the time and decreased the load of administrator by dynamically assign the conflicting and non-conflicting permissions to roles. In addition, roles assignment is also dynamic. In this way, the performance of our proposed model is better from the previously proposed models.

4.5. Formal Specification and Algorithm of Proposed Model

An adequate lightweight modeling system is provided by the ALLOY that is used for the verification on the internal consistency of RBAC and some algebraic properties [37]. RBAC has been specified without any conflict using alloy language [38]. The proposed model is formally specified below.

- USERS, UATT, ROLES, RATT, OPS, OBS, and OATT (users, user’s attributes, roles, role’s attributes, operations, objects and object’s attributes, respectively).
- CatCon: is used for the categorize-containers that contain attributed objects.
- GActS: is used for the generic action set that contains action with conflict of interest (COI).
- ops(g_act : GActS) → {ops ⊆ OPS}, the mapping of actions with COI onto generic action set. In this way, the operations (actions with COI) linked with generic action set g_act.
- obj(cat_con: CatCon) → {obj ⊆ OBJ}, the mapping of attributed objects onto categorize-containers. In this way, the attributed objects linked with categorize-container cat_con.
- PERMS ∈ 2^{OPSxOBS} ∪ 2^{OPSxCatCon} ∪ 2^{GActSxCatCon} ∪ 2^{GActSxOBS}, set of conflicted and non-conflicted permissions with attributes.
- PASIGN ⊆ PERMS × ROLES, a many to many mapping of attributed permissions to attributed role assignment relation.
- Permission_auto_assigned (r : ROLES) → 2^{PERMS}, the automatic mapping of attributed role r onto a set of conflicted and non-conflicted permissions by using the attributes.
- Permission_auto_assigned(r) = {p ∈ PERMS | (p, r) ∈ PASIGN}
- UASIGN ⊆ USERS × ROLES, a many to many attributed users to attributed role assignment relation
- Users_auto_assigned: (r : ROLES) → 2^{USERS}, the automatic mapping of attributed role r onto a set of attributed users by using attributes.
- Users_auto_assigned(r) = {u ∈ USERS | (u, r) ∈ UASIGN}
- Activate: this is a function that grants access to a user on particular permission so that user can activate the permission for performing the tasks.
- ¬Activate: this function is used to restrict a user to access a particular permission.
- Activated: this is a function that is used to return an indication. Moreover, it also informs that the user already activated specific permission from the return value.
- ¬Activated: this function is used to indicate that the user has not activated particular permission.
- ConfP: a function that nominates two permissions as the conflicting permissions. Moreover, a single user cannot activate these two permissions.
- ConfOP: a function that declares two actions as confliction actions. This is so that a user cannot access two permissions that have the same objects but two COI actions at the same time.
• User_request_activate: A query that is used to activate a particular permission from a specified role; this query originates from the user.

\[
\forall p_8, p_{10}, p_{18}, p_{20} \in \text{PERMS}, \ user_6, \ user_7 \in \text{USERS}, \\
role_3, \ role_4 \in \text{ROLES}, \ user_6 \in \text{role}_3, \ role_4, \ user_7 \in \text{role}_3, \ role_4, \\
p_8, p_{10} \in \text{role}_3, \ p_{18}, p_{20} \in \text{role}_4 : \ \text{mex}(p_8, p_{18}), \ \text{mex}(p_{10}, p_{20}) \land \\
\text{user}_6 \text{\_request\_activate}(user_6, \text{role}_3, p_8) \land \text{user}_6 \text{\_request\_activate}(user_6, \text{role}_3, p_{10}) \land \\
\text{user}_7 \text{\_request\_activate}(user_7, \text{role}_4, p_{18}) \land \text{user}_7 \text{\_request\_activate}(user_7, \text{role}_4, p_{20}) \land \\
\neg \text{activated}(user_6, \text{role}_3, p_{18}) \Rightarrow \text{activates}(user_6, \text{role}_3, p_8) \land \\
\neg \text{activated}(user_6, \text{role}_4, p_{20}) \Rightarrow \text{activates}(user_6, \text{role}_3, p_{10}) \land \\
\neg \text{activated}(user_7, \text{role}_3, p_8) \Rightarrow \text{activates}(user_7, \text{role}_4, p_{18}) \land \\
\neg \text{activated}(user_7, \text{role}_3, p_{10}) \Rightarrow \text{activates}(user_7, \text{role}_4, p_{20})
\]

Whenever a user tries to access or activate conflicting permission, they will request to activate that particular conflicting permission. However, this type of request and check access is not necessary for the non-conflicting permissions. We used the permission, roles, and user’s names according to the example, given in Figure 4b. A user can access conflicting permissions and activate them through the function activate. If the user is not authorized for permission or when a user tries to activate two conflicting permissions that have COI, access is denied and the user receives a message. After that, the access is granted on the permission that is accessed first and the second permission access is denied. Meanwhile, the functions also check whether a user is authorized for a particular role or not. After checking the access of the role, the functions will move forward to check the authorization of a user on permissions as well as permissions with COI. In the last of this subsection, the proposed model’s algorithm is described in six steps.

**Step 1:** Create basic entities of RBAC: attributed-objects, actions with COI, attributed-roles, and attributed-users

**Step 2:** Create categorize-containers and assign attributed-objects. Create generic-action-set and assign actions with COI.

**Step 3:** Conflicted and Non-Conflicted Attributed-Permission can be created in four different ways (see Algorithm 1).

- Apply Actions on objects to create permissions one by one (Traditional RBAC)
- Apply Actions on categorize-containers to create multiple permissions with same actions and different objects (New feature)
- Apply generic-action-set on objects to create multiple permissions with same objects and different actions (new feature)
- Apply generic-action-set on categorize-containers to create multiple permissions with different objects and different actions (new feature)

**Step 4:** Conflicted and Non-Conflicted permissions automatically assigned to attributed-roles with the help of attributes.

**Step 5:** The attributed-roles automatically assigned to attributed-users.

**Step 6:** User can access the assigned attributed roles and permissions. User cannot access two conflicted permissions at a time and receives message “Access Denied” (see Algorithm 2).
Algorithm 1: Permissions creation through four methods

1: \[\text{if users choose multiple-to-multiple relationship method then}\]
2: \[\text{Get all Categorize Containers from Database}\]
3: \[\text{Get all Generic Action Sets from Database}\]
4: \[s_{\text{Container}} = \text{the selected Categorize Container}\]
5: \[s_{\text{Set}} = \text{the selected Generic Action Set}\]
6: \[\text{for } obj \in s_{\text{Container}} \text{ do}\]
7: \[\text{for } act \in s_{\text{Set}} \text{ do}\]
8: \[T = \text{new Permission}(obj, act)\]
9: \[\text{Update the new permission } T \text{ to the database}\]
10: \[\text{end for}\]
11: \[\text{end for}\]
12: \[\text{end if}\]
13: \[\text{if users choose multiple-to-single relationship method then}\]
14: \[\text{Get all Categorize Containers from Database}\]
15: \[\text{Get all Actions from Database}\]
16: \[s_{\text{Container}} = \text{the selected Categorize Container}\]
17: \[s_{\text{Action}} = \text{the selected Action}\]
18: \[\text{for } obj \in s_{\text{Container}} \text{ do}\]
19: \[T = \text{new Permission}(obj, s_{\text{Action}})\]
20: \[\text{Update the new permission } T \text{ to the database}\]
21: \[\text{end for}\]
22: \[\text{end if}\]
23: \[\text{if users choose single-to-multiple relationship method then}\]
24: \[\text{Get all Objects from Database}\]
25: \[\text{Get all Generic Action Sets from Database}\]
26: \[s_{\text{Object}} = \text{the selected Object}\]
27: \[s_{\text{Set}} = \text{the selected Generic Action Set}\]
28: \[\text{for } act \in s_{\text{Set}} \text{ do}\]
29: \[T = \text{new Permission}(s_{\text{Object}}, act)\]
30: \[\text{Update the new permission } T \text{ to the database}\]
31: \[\text{end for}\]
32: \[\text{end if}\]
33: \[\text{if users choose single-to-single relationship method then}\]
34: \[\text{Get all Objects from Database}\]
35: \[\text{Get all Actions from Database}\]
36: \[s_{\text{Object}} = \text{the selected Object}\]
37: \[s_{\text{Action}} = \text{the selected Action}\]
38: \[T = \text{new Permission}(s_{\text{Object}}, s_{\text{Action}})\]
39: \[\text{Update the new permission } T \text{ to the database}\]
40: \[\text{end if}\]
Algorithm 2 User’s access to Roles and Permissions

1: Roles = List of all roles in database
2: Permissions = List of all permissions in database
3: privilegedRoles = []
4: privilegedPermissions = []
5: for permis ∈ Permissions do
6: if isRightTime(permis.Object.Time) = = 1 then
7: privilegedPermissions.Append(permis)
8: end if
9: end for
10: for role ∈ Roles do
11: if isRightTime(role.Time) = = 1 then
12: privilegedRoles.Append(role)
13: end if
14: end for
15: historyPermissions = []
16: if doExecute() = = 1 then
17: sPermission = the selected Permission
18: sAction = sPermission.Action
19: sObject = sPermission.Object
20: isConflicted = 0
21: for permission ∈ historyPermissions do
22: if sObject = = permission.Object.ConflictedAction then
23: isConflicted = 1
24: end if
25: end for
26: if isConflicted = = 1 then
27: Warning “being conflicted actions on the same object” to the user
28: else
29: historyPermissions.Append(sPermission)
30: Notice the permission accessed successfully
31: end if

4.6. Benefits of the Proposed Model

There are several benefits to the proposed model, as it gives the solutions to the previously stated problems. We discuss their benefits with regards to different aspects.

4.6.1. Decrease Load of Administrator

The proposed model decreases the burden of the administrator as most of the work in this model is based on the attributes. In the typical RBAC standard, the administrator used to assign the permissions to roles and roles to users manually. This is a hectic job for the administrator. Moreover, the permissions assigned to roles as well as roles assignment to users will be done with the help of attributes. In this way, the permission assignment and roles assignment is automatic in our proposed model. Thus, the administrator load is ultimately decreased as compared to typical RBAC standard.

4.6.2. No Decrement in User’s Authority

The proposed model does not decrease the authority domain of the user because it implements the SOD on the level of permissions, not on the level of roles. In this way, a user can access all non-conflicting permissions of the assigned roles as well as one the conflicting permission from the two permissions that have COI with each other. Previously, in RBAC standard, if a user accesses conflicting permission
from one role then the user cannot access the second conflicting permission as well as user cannot access the whole assigned role of that conflicting permission. In this way, the RBAC standard decreases the authority domain of users. In our model, the users can access the non-conflicting permissions of all assigned roles because SOD is implemented on the level of permissions, not on the level of roles.

4.6.3. No SOD Violation by the Users

The proposed model does not allow end users to violate SOD. If a user activates a conflicting permission A, they are not allowed to activate the conflicting permission B because permission A and B have COI with each other. In the RBAC standard, a user can access conflicting permission in a session but after terminating the session, the user can access the second conflicting permission. In this way, end users violate the SOD in typical RBAC standard. On the other hand, the end users of our proposed model are not allowed to access the two conflicting permissions that have COI, whether they try to access in the same session or try to access in different sessions.

4.7. Limitations

RBAC is a popular access control model with several useful features. The proposed model works as a hybrid model that dynamically assigns permissions to roles, as well as, assigns users to roles with the help of different attributes. This particular approach is only related to the basic entities of RBAC. Moreover, the proposed model implements the SOD on the level of permissions. The proposed model is only covering the sub-category of RBAC that is SOD. The proposed model does not deal with role hierarchy, negative authorization, and inheritance concepts related to the RBAC model. Moreover, the model is significantly suitable for the desktop applications, but if the requirements of the users are related to android-based and web-based systems, this model is not suitable. It is because of the design of the model that is not suitable for online applications. Moreover, the model can be used for the wireless sensor network (WSN) security enhancement, but right now, this model is unable to directly cover the security of WSN. The model can be used for WSN after the addition of some functionalities and slightly changing the model with respect to WSN working style.

5. Implementation and Comparative Analysis

This section consists of two subsections. The first subsection is based on the implementation of the proposed model and the second subsection is based on the discussion with the comparison of other models.

5.1. Implementation

We have simulated the proposed model by developing an application as a proof of concept. The application design is based on the proposed model. There are two user panels: one is for administrator and second is for end users. The administrator can create attributed objects, actions with and without mentioning their conflicts, attributed roles, and attributed users. According to the model, the administrator can assign objects and actions to different containers according to the access control policy, in the application. In addition, the administrator can create conflicted and non-conflicted permissions with four different methods. The screenshot of permission creation is attached in Figure 6. The created permissions are also attributed. There is even a view of every permission in the form of actions and objects. The permissions are automatically assigned to roles with the help of attributes. The administrator can view the roles and permissions assigned to that role, in the roles and permissions section. In the final step, SOD implementation on the level of permission can be checked. When an end user login to the system and access the assigned roles. In addition, the roles are also automatically assigned to users by using attributes of users and roles. The user can only view and activate those roles that are assigned to him/her. After selecting a role, the user can view and activate the permissions inside that role.
When a user activated conflicted permission then he/she cannot access second conflicted permission that has COI with first permission. The example is tested in the application and the user receives the message “access denied”, as shown in Figure 7. The user already activated the permission 1 that action = ‘approve’ and object = ‘obj1’. After that user tries to access permission 2 that has action = ‘submit2’ and object = ‘obj1’. In this way, both of these permissions are conflicted due to conflicted actions approve and submit. Thus, the user can only access one of these permissions. As the user already activated permission 1 so the user is not authorized to access the permission 2. This is the reason, the user received the message “access denied”. The application is developed by using the following software and development tools.

![Figure 6. Permission Creation with four Different Methods.](image)

![Figure 7. User’s Try to Activate Conflicted Permission after Activating the First One.](image)
The software is briefly tested on Windows 10 OS Version 1809 (build 17763.348). The source code is written in Java language. NetBeans IDE 8.2 (Build 201609300101) is used to develop the software. JDK 1.8.0_172 is the version of the development environment for building the application. In this project, gui.LoginForm class is the main class. On the backend side, Microsoft SQL Server 2017 is used to test the application with the database connection, the detail about the version of the database system as follows:

- Microsoft SQL Server Management Studio 14.0.17224.0
- Microsoft Analysis Services Client Tools 14.0.1016.244
- Microsoft Data Access Components (MDAC) 10.0.17763.1
- Microsoft MSXML 3.0 6.0
- Microsoft Internet Explorer 9.11.17763.0
- Microsoft .NET Framework 4.0.30319.42000
- Operating System 6.3.17763

The complete source code of the simulated-application is online and can be downloaded from the given link: https://github.com/projectcnvn/PBSODDRBAC (see Supplementary Materials).

5.2. Comparative Analysis

The comparative analysis is performed in two forms. Firstly, we have done the performance analysis of the proposed model with traditional RBAC model, in terms of time efficiency. Secondly, we have done the feature comparison of the proposed model with the previous models. The comparison of proposed permission based dynamic RBAC (PDRBAC) with different access control models is given in the form of a graph in Figure 8a–c.

The proposed model (PDRBAC) outperforms the RBAC model, while creating multiple permissions at a time. On the contrary, RBAC model only creates one permission at a time. By applying generic-action-sets and categorize-containers, our work is creating three, four, and nine permissions at a time. The administrator can also create a single permission. In this manner, this work is significantly performing well by creating more permissions against every administrator effort, as shown in Figure 8a. The orange bars show the permission ratio of the proposed model against the typical RBAC model in blue bars. In addition, our work is compared with other access control models with respect to flaws ratio. There are a number of flaws in the previous access control models such as the three flaws in access control models that implement SOD on the level of roles, extensive burden on the administrator, and static behavior. These flaws are discussed in the previous sections. In this work, we overcome these flaws and implement the proposed model in the form of a hybrid model. According to Figure 8b, the flaws ratio shows that typical RBAC model (sky blue bar) is containing the more flaws due to SOD violations and administrator work load, RBAC smart contract (RBAC-SC) (grey bar) is better than RBAC model. Attributed RBAC (ARBAC) (yellow bar) and Context-aware access control (CaAC) (orange bar) models are on the third number due to SOD violations, but they decrease the administrator work load that is why they are much better. Permission based SOD in RBAC (PSD-RBAC) (green bar) is on the second last number because of administrator’s work load and static behavior but there are no SOD violations. In last, our work, permission based SOD in dynamic RBAC (PDRBAC) (dark blue smaller bar) has efficiently overcome the flaws by implementing the efficient SOD and decreasing the administrator work load with its dynamic nature, as shown in Figure 8b.

The proposed model performance is better compared to the typical RBAC model. The blue line shows the performance of typical RBAC model [9], the yellow line shows the performance of PSD-RBAC [23], and the red line shows the performance of the proposed model. The proposed model is efficient and facilitates the system by creating permissions and assignment of roles and permissions in less time. There are two main processes that play an important role in system performance. Firstly, the permission creation process, in which administrator can create multiple permissions at once. On the other hand, administrator can create permissions one by one, in typical RBAC model. In this way, more time is required for the creation of permissions in RBAC model. Moreover, administrator saves
time by creating multiple permissions in the proposed model. Secondly, the assignment of permissions to roles and roles to users is automatic, in the proposed model. In this way, the system automatically assigns the permissions to roles and roles to users. On the other hand, the administrator has to assign permissions to roles and roles to users manually, in the RBAC model. It is a hectic and time-consuming job, in the RBAC model. Ultimately, if we talk about the proposed model as a whole, then our proposed model can perform significantly better as compared to traditional RBAC model, as shown in Figure 8c.

Figure 8. Comparison of the proposed model with previous access control models.
The proposed model outperforms as compared to the previous models, as shown in Table 3. The previous models offering fewer features and proposed model is providing more features like dynamic RBAC along with efficient SOD implementation. RBAC [9], PSD-RBAC [23], and RBAC-SC [14] are not dynamic because these models are based on typical RBAC model that is static by nature. The assignment of users to roles is static in these models. On the other hand, permissions to roles assignment are manual and performed by the administrator in RBAC and PSD-RBAC. In this way, the implementation and designing of access control are manual that put extensive burden on administrator. On the other hand, our proposed model is dynamic because the assignment of permissions to roles and roles to users is performed with the help of several attributes that makes this work dynamic.

Table 3. Feature analysis of access control models with the proposed model.

| Features                        | RBAC [9] | ABAC [19] | Attributed RBAC [24] | PSD-RBAC [23] | RBAC-SC [14] | CaAC [17] | Proposed Model |
|---------------------------------|----------|-----------|----------------------|---------------|--------------|-----------|----------------|
| Least Privileges                | ✓        | ✓         | ✓                    | x             | x            | ✓         | ✓              |
| Simplicity                      | ✓        | ✓         | ✓                    | ✓             | ✓            | ✓         | ✓              |
| Flexibility                     | x        | ✓         | ✓                    | ✓             | ✓            | ✓         | ✓              |
| Efficient SOD Implementation    | x        | x         | ✓                    | ✓             | x            | ✓         | ✓              |
| Policies specification & Maintenance | ✓       | ✓         | ✓                    | ✓             | ✓            | ✓         | ✓              |
| Less Load of Administrator      | x        | ✓         | ✓                    | ✓             | ✓            | ✓         | ✓              |

Table 3: The feature analysis is highlighted through tick ✓ and cross x, in this table. ✓ means that the feature is available as well as in good form, in the models. x means that feature is not available or in poor quality.

As the RBAC model is well-liked due to the least privilege feature, if any model’s working style is based on the RBAC entities, the model is also capable to provide tight security. All the model listed in Table 3 are enriched to provide the least privileges, except the ABAC model. The least privilege and simplicity features are associated with the RBAC model due to the concept of roles and tight security. In this manner, ABAC is unable to provide the least privileges, tight security, and simplicity as much as the RBAC can [39]. Moreover, the simplicity is provided by all the models except the ABAC [19] and CaAC [17] due to their complex design, as compared with other models. Furthermore, the least privileges and simplicity are the stunning features of our work because this work is implementing SOD and using the concept of roles that indulge the tight security and simplicity in the proposed model.

Moreover, the dynamic behavior also makes our model flexible like ABAC [19] and CaAC [17] because these models can efficiently behave according to the environmental conditions. The attributed-RBAC model is also a dynamic model but the permission creation process is fixed. The administrator can create permissions only with the help of containers but single permissions are not allowed directly that makes it out of flexibility scope. On the other hand, the rest of the models do not provide flexibility due to their static nature. However, the SOD is implemented efficiently in our work as well as in PSD-RBAC [23] model because both models implement SOD on the level of permissions rather than roles. On the contrary, all other models are failed to implement SOD in an efficient manner and they are violating SOD and RBAC during the implementation as discussed in Section 3. In RBAC, attributed-RBAC, PSD-RBAC, RBAC-SC, CaAC, and the proposed model, policy specification and maintenance, usually used after the implementation of access control for managing the security mechanism, is available in good form because of the concept of roles. In this manner, these models can manage the number of permissions as well as view the access control mechanism easily. In spite of this, ABAC model is not providing this feature and management of ABAC is a difficult task [24,39].

More specifically, PSD-RBAC [23] is a static model and administrator has sole responsibility to create permissions, assign conflicting and non-conflicting permissions to roles. In addition, administrator has to assign roles to users also. In this way, the model is efficient enough for implementing SOD in a significant way; however, the model is unable to reduce the burden of administrator. In addition, RBAC and RBAC-SC are also based on the manual assignment of entities to each other like the roles to users’ assignment. Thus, these two models put extensive burden on the administrator. Conversely, our model decreases the load of administrator, with the help of attributes. However, attributed-RBAC [24]
is a dynamic model but there is no concept of SOD dealing, as discussed in their future work also. Furthermore, the permission creation process is time consuming when the administrator has to create one or two permissions. For example, the administrator can create separate containers and then assign the objects and actions to the containers for creating permissions. This may be considered an inefficient solution.

Comparatively, our model deals with the stated problems in a well-mannered way. This work is dynamic as well as implements SOD in an efficient manner. Moreover, the permission creation process with four different ways is not a burden on the administrator because it is provided for the ease of administrator and time-saving. The administrator can create one or two permissions directly instead of creating containers first, as the previous attributed-RBAC model does. As the proposed model’s basic structure is based on the RBAC model, our model provides simplicity, least privileges, and ease of maintenance. Meanwhile, efficient SOD implementation is one of the stunning features of the proposed model. The performance in terms of time is comparatively better due to the involvement of attributes in the RBAC model that completes the access control designing and implementation in less time, in our model. Hence, our work is efficient and has more features as compared to previous work.

6. Conclusions and Future Directions

A dynamic RBAC model that considers the SOD on the level of permissions has been implemented to create flexibility, reduce load and address dynamic behavior. The true dimensions and inspirations behind access control models are identified in a way to propose an efficient access control model. In addition, in this work, a dynamic RBAC model is introduced to enhance dynamic features, least privileges and more flexibility with the addition of attributes. The proposed model provides ease of administration, dynamic behavior, tight security, and efficient SOD implementation. We discussed different problems with regards to SOD implementation on the level of roles. In addition, we also discussed the problems in RBAC and ABAC models. As a result, the proposed model is an efficient solution to overcome the stated problems: decrease administrator load, designing access control in less time, and no SOD violation. The results and feature analysis also strengthened our research. In the future, the work will continue to implement a role hierarchy’s concept in dynamic RBAC model. Moreover, the development of a secure and dynamic access control model for the blockchain, wireless sensor networks (WSN), and internet of things (IoT) systems, would be a significant topic for future work.

Supplementary Materials: The complete source code of the simulation is online and can be found here: https://github.com/projectcnvn/PBSODDRBAC.

Author Contributions: Conceptualization, M.U.A. and Z.Q.; methodology, M.U.A.; formal analysis, O.A., and T.V.D.; writing—original draft preparation, M.U.A.; writing—review and editing, Z., N.T.S., and N.W.H.; supervision, Z.Q.

Funding: The research study is supported by the funding committee “National Natural Science Foundation of China”. The funding is granted under the Project Name “Key theories and techniques of Privacy and Security based on data life-cycle” and the Project Number is “61520106007”.

Acknowledgments: We are thankful to the anonymous editors and reviewers for their unconditional support, valuable comments and kind suggestions of this article.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the work, writing of the manuscript, or decision to publish the results.

References

1. Samarati, P.; de Vimercati, S.C. Access control: Policies, models, and mechanisms. In Proceedings of the International School on Foundations of Security Analysis and Design, Bertinoro, Italy, 18–30 September 2000; pp. 137–196.
2. Cheminod, M.; Durante, L.; Seno, L.; Valenza, F.; Valenzano, A. A comprehensive approach to the automatic refinement and verification of access control policies. Comput. Secur. 2018. [CrossRef]
3. Verma, D.C. Simplifying network administration using policy-based management. *IEEE Netw.* 2002, 16, 20–26. [CrossRef]

4. Sandhu, R.; Munawer, Q. How to do discretionary access control using roles. In Proceedings of the Third ACM Workshop on Role-Based Access Control, Fairfax, VA, USA, 22–23 October 1998; pp. 47–54.

5. Li, N. Discretionary access control. In *Encyclopedia of Cryptography and Security*; Springer: Berlin, Germany, 2011; pp. 353–356.

6. Juueneman, R.R. Integrity controls for military and commercial applications. In Proceedings of the Fourth Aerospace Computer Security Applications, Orlando, FL, USA, 12–16 September 1988; pp. 298–322.

7. Barkley, J. Comparing simple role based access control models and access control lists. In Proceedings of the second ACM workshop on Role-Based Access Control, Fairfax, VA, USA, 6–7 November 1997; pp. 127–132.

8. Sandhu, R.S.; Coyne, E.J.; Feinstein, H.L.; Youman, C.E. Role-based access control models. *Computer* 1996, 29, 38–47. [CrossRef]

9. Incits, A. Incits 359-2004. role-based access control. *Am. Natl. Stand. Inf. Technol* 2004, 359, 2–10.

10. Zhu, Y.; Huang, D.; Hu, C.-J.; Wang, X. From RBAC to ABAC: Constructing flexible data access control for cloud storage services. *IEEE Trans. Serv. Comput.* 2015, 8, 601–616. [CrossRef]

11. Batra, G.; Atluri, V.; Vaidya, J.; Sural, S. Enabling the Deployment of ABAC Policies in RBAC Systems. In Proceedings of the 32nd IFIP Annual Conference on Data and Applications Security and Privacy, Bergamo, Italy, 16–18 July 2018; pp. 51–68.

12. Alam, M.; Emmanuel, N.; Khan, T.; Xiang, Y.; Hassan, H. Garbled role-based access control in the cloud. *J. Ambient Intell. Humaniz. Comput.* 2018, 9, 1153–1166. [CrossRef]

13. Nazarian, F.; Motamemi, H.; Nematzadeh, H. Emergency role-based access control (E-RBAC) and analysis of model specifications with alloy. *J. Inf. Secur. Appl.* 2019, 45, 131–142. [CrossRef]

14. Cruz, J.P.; Kaji, Y.; Yanai, N. RBAC-SC: Role-Based Access Control Using Smart Contract. *IEEE Access* 2018, 6, 12240–12251. [CrossRef]

15. Jha, S.; Sural, S.; Atluri, V.; Vaidya, J. Specification and Verification of Separation of Duty Constraints in Access-Based Control. *IEEE Trans. Inf. Forensics Secur.* 2018, 13, 897–911. [CrossRef]

16. Kuhn, D.R.; Coyne, E.J.; Weil, T.R. Adding attributes to role-based access control. *Computer* 2010, 43, 79–81. [CrossRef]

17. Zheng, R.; Jiang, J.; Hao, X.; Ren, W.; Xiong, F.; Zhu, T. CaACBIM: A Context-aware Access Control Model for BIM. *Information* 2019, 10, 47. [CrossRef]

18. Jin, X.; Krishnan, R.; Sandhu, R. A unified attribute-based access control model covering DAC, MAC and RBAC. In Proceedings of the 26th IFIP Annual Conference on Data and Applications Security and Privacy, Paris, France, 11–13 July 2012; pp. 41–55.

19. Hu, V.C.; Ferraiolo, D.; Kuhn, R.; Friedman, A.R.; Lang, A.J.; Cogdell, M.M.; Schnitzer, A.; Sandlin, K.; Miller, R.; Scarfone, K. Guide to attribute based access control (ABAC) definition and considerations (draft). *NIST Spec. Publ.* 2013, 800. [CrossRef]

20. Xu, R.; Chen, Y.; Blasch, E.; Chen, G. Blendcac: A smart contract enabled decentralized capability-based access control mechanism for the IOT. *Computers* 2018, 7, 39. [CrossRef]

21. Lan-sheng, H.; Fan, H.; Kojo, A.B. Least privileges and role’s inheritance of RBAC. *Wuhan Univ. J. Nat. Sci.* 2006, 11, 185–187. [CrossRef]

22. Sandhu, R.S. Separation of Duties in Computerized Information Systems. In Proceedings of the IFIP WG11.3 Workshop on Database Security, Halifax, UK, 18–21 September 1990; pp. 179–190.

23. Habib, M.A.; Mahmood, N.; Shahid, M.; Aftab, M.U.; Ahmad, U.; Faisal, C.M.N. Permission Based Implementation of Dynamic Separation of Duty (DSD) in Role Based Access Control (RBAC). In Proceedings of the 8th International Conference on Signal Processing and Communication Systems, Gold Coast, Australia, 15–17 December 2014; pp. 1–10.

24. Aftab, M.U.; Habib, M.A.; Mehmood, N.; Aslam, M.; Irfan, M. Attributed role based access control model. In Proceedings of the Conference on Information Assurance and Cyber Security, Rawalpindi, Pakistan, 18 December 2015; pp. 83–89.

25. Al-Kahtani, M.A.; Sandhu, R. A model for attribute-based user-role assignment. In Proceedings of the 18th Annual Computer Security Applications Conference, Las Vegas, NV, USA, 9–13 December 2002; pp. 1–10.
26. Rajpoot, Q.M.; Jensen, C.D.; Krishnan, R. Integrating attributes into role-based access control. In Proceedings of the 29th IFIP Annual Conference on Data and Applications Security and Privacy, Fairfax, VA, USA, 13–15 July 2015; pp. 242–249.
27. Chen, B.-C.; Yang, C.-T.; Yeh, H.-T.; Lin, C.-C. Mutual Authentication Protocol for Role-Based Access Control Using Mobile RFID. Appl. Sci. 2016, 6, 215. [CrossRef]
28. Habib, M.A.; Praher, C. Object based dynamic separation of duty in RBAC. In Proceedings of the 4th International Conference for Internet Technology and Secured Transactions, London, UK, 9–13 November 2009; pp. 1–5.
29. Jha, S.; Sural, S.; Atluri, V.; Vaidya, J. Enforcing separation of duty in attribute based access control systems. In Proceedings of the International Conference on Information Systems Security, Kolkata, India, 16–20 December 2015; pp. 61–78.
30. Joshi, J.B.; Bertino, E.; Latif, U.; Ghafoor, A. A generalized temporal role-based access control model. IEEE Trans. Knowl. Data Eng. 2005, 17, 4–23. [CrossRef]
31. Veloudis, S.; Niksanke, N. A Novel Permission Hierarchy for RBAC for Dealing with SoD in MAC Models. Comput. J. 2016, 59, 462–492. [CrossRef]
32. Ghosh, S.; Karar, V. Blowfish Hybridized Weighted Attribute-Based Encryption for Secure and Efficient Data Collaboration in Cloud Computing. Appl. Sci. 2018, 8, 1119. [CrossRef]
33. Yin, H.; Xiong, Y.; Zhang, J.; Ou, L.; Liao, S.; Qin, Z. A Key-Policy Searchable Attribute-Based Encryption Scheme for Efficient Keyword Search and Fine-Grained Access Control over Encrypted Data. Electronics 2019, 8, 265. [CrossRef]
34. Zhou, L.; Su, C.; Li, Z.; Liu, Z.; Hancke, G.P. Automatic fine-grained access control in SCADA by machine learning. Future Gener. Comput. Syst. 2019, 93, 548–559. [CrossRef]
35. Wang, X.; Wang, L.; Li, Y.; Gai, K. Privacy-aware efficient fine-grained data access control in Internet of medical things based fog computing. IEEE Access 2018, 6, 47657–47665. [CrossRef]
36. Fatima, A.; Ghazi, Y.; Shibli, M.A.; Abassi, A.G. Towards Attribute-Centric Access Control: An ABAC versus RBAC argument. Secur. Commun. Netw. 2016, 9, 3152–3166. [CrossRef]
37. Zao, J.; Wee, H.; Chu, J.; Jackson, D. RBAC schema verification using lightweight formal model and constraint analysis. In Proceedings of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT), Villa Gallia, Como, Italy, 2–3 June 2003.
38. Schaad, A.; Moffett, J.D. A lightweight approach to specification and analysis of role-based access control extensions. In Proceedings of the seventh ACM symposium on Access control models and technologies, Monterey, CA, USA, 3–4 June 2002; pp. 13–22.
39. Umar Aftab, M.; Qin, Z.; Zakria; Ali, S.; Pirah; Khan, J. The Evaluation and Comparative Analysis of Role Based Access Control and Attribute Based Access Control Model. In Proceedings of the 15th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), Chengdu, China, 14–16 December 2018; pp. 35–39.

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).