Risk-Aware RBAC Sessions

Khalid Zaman Bijon[1], Ram Krishnan[2], and Ravi Sandhu[1]

[1] Institute for Cyber Security & Department of Computer Science,
[2] Institute for Cyber Security & Department of Electrical and Computer Engineering
University of Texas at San Antonio

Abstract. Role Based Access Control (RBAC) has received considerable attention as a model of choice for simplified access control over the past decade. More recently, *risk awareness* in access control has emerged as an important research theme to mitigate risks involved when users exercise their privileges to access resources under different contexts such as accessing a sensitive file from work versus doing the same from home. In this paper, we investigate how to incorporate "risk" in RBAC—in particular, in RBAC *sessions*. To this end, we propose an extension to the core RBAC model by incorporating risk awareness in sessions where the risk is bounded by a session-based "risk-threshold." We develop a framework of models for role activation and deactivation in a session based on this threshold. Finally, we provide formal specification of one of these models by enhancing the NIST core RBAC model.

1 Introduction

Over the past decade, considerable research has been conducted in Role Based Access Control (RBAC) [12]. In RBAC, *session* is an important risk mitigating feature in which a user interacts with the system by enabling a limited set of roles (although, in the absence of constraints it is feasible for the user to sequentially interact with the system using all the privileges based on roles assigned to that user). Risk awareness in access control is a new but prominent issue as the need for enabling access in an agile and dynamic way has emerged. Several authors have conducted research in this arena [2–4,7,9–11], mainly, attempting to combine risk with different access control systems. According to [10], a practical risk aware access control system should have a risk assessment process relevant to the context of the application as well as proper utilization of the estimated risk for granting or denying access requests. A risk aware access control system differs from traditional access control systems in that it permits/denies access requests dynamically based on estimated risk instead of predefined access control policies which always give same outcomes.

The concept of a session in classical RBAC has dual motivation. It serves as a basis for dynamic separation of duties whereby some roles cannot be combined in a single session. It also serves as a basis for a user to exercise least privilege with respect to powerful roles that can remain inactivated until they are really required. Both motivations are conceptually related to risk. Thus it seems natural to build additional risk mechanisms around the session concept.

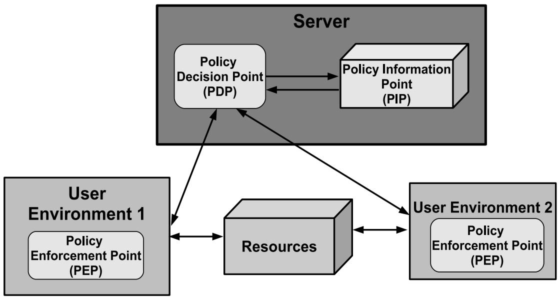
Server
Policy Decision Point (PDP)
Policy Information Point (PIP)
User Environment 1
Policy Enforcement Point (PEP)
Resources
User Environment 2
Policy Enforcement Point (PEP)

Fig. 1. A simple PDP/PEP based Access Control Enforcement Model

The core idea in this paper is to set a risk-threshold that limits a user's attempt to activate roles to enhance the session's access capability. Consider a typical access control enforcement framework that consists of one or more policy information, decision and enforcement points (PIP, PDP and PEP respectively). A PEP enforces policy decisions made by the PDP on the client. The PDP makes this decision by consulting one or more PIPs. The PDP/PIP might reside in a central server while the PEPs could be implemented in different user environments. Figure 1 illustrates such an access control enforcement framework in which there are two different user environments each containing a PEP. For each user access request, the PEP contacts the PDP residing in the centralized server. The PDP consults the PIP for each requested access and responds with an authorization decision to the PEP.

Consider how RBAC role activation and deactivation would work under this enforcement model. When a user creates a session and requests to activate a set of roles, the PEP on the user's system forwards the request to the PDP. The PDP responds with allow or deny after verifying whether the user has been assigned the requested set of roles to be activated. If allowed, the PDP sends the aggregate set of permissions based on the role permission assignment information for each role in the requested role set (by consulting with the PIP). From here on, when the user requests to access a specific resource, the PEP checks if the request is allowed based on this set of permissions without having to contact the PDP for each request. Note that if the user needs to activate a new role, the PEP would have to verify this with the PDP and fetch the corresponding additional set of permissions if allowed. Also, if a role is deactivated by the user, the PEP can appropriately adjust the permissions dropping those that are exclusively authorized by the deactivated role.

Now, if the session were to be compromised or hijacked, say by some malware in the user's computer, the attacker would be freely able to operate with the privileges of the user enabled in that session. The attacker could completely impersonate that user in the system by further activating all the roles of the user. A session risk threshold can mitigate this threat. For instance, if each permission can be assigned a risk value, the total risk of a role can be computed (e.g., as the sum of risk of each permission assigned to that role [11]) The session risk-threshold defines the maximum risk that the session can carry at any time. Effectively, the threshold limits the set of roles that can be activated in a given session. Under this scenario, if the session were to be compromised, the threshold places an upper limit on the maximum damage that can occur. For instance, an intelligent system can detect the malicious context within which a user is operating and place a very low risk threshold that prevents the user from ever activating certain powerful and hence highly risky roles in that session. This is a useful and practical mitigation strategy given that "bring your own device" and smart phones have become common platforms in the modern IT environment.

In this paper, we investigate various design issues with respect to role activation and deactivation in RBAC sessions where a session risk-threshold exists. We develop a framework and identify various models for the above and formally specify one of them by enhancing the NIST core RBAC model. We assume that a session risk-threshold already exists or can be computed. That is, we do not focus on how to compute session risk based on user's context in this paper. This issue has been the focus of prior work in this area such as [3, 11].

We categorize risk-threshold of sessions into three different types based on when and how it is computed as well as type of information it uses. We then develop a framework that identifies several system functionalities and issues for modeling different role activation-deactivation processes within such risk aware sessions. We show that some of the existing work on risk in RBAC fits well within our framework and the framework identifies a rich scope for further research in this arena. To this end, we formally specify one of the models in the proposed framework by enhancing the NIST Core RBAC model.

2 Risk-Aware RBAC Session Characteristics

The characteristics of role activation and deactivation model design can vary depending on when and how the session risk-threshold is computed. There are at least three points in time at which it may be computed. We term each of these points as static, dynamic and adaptive respectively. We discuss this below.

Session with Static Risk-Threshold (SSR): In SSR, every session of a user has a constant risk-threshold. An administrator might statically calculate session risk-threshold for a user by evaluating several properties, e.g., user's credential and assigned role-set, and it remains unchanged for every session of a given user. This session is useful to enforce certain well-known RBAC functionalities such as cardinality constraint or dynamic separation of duty. For instance, static risk-threshold value could be such that a user could not activate and keep more than two roles in a session simultaneously.

Session with Dynamic Risk-Threshold (SDR): In SDR, the risk-threshold may vary from session to session for a given user. Unlike SSR, the risk should be estimated before every session creation. Thus certain dynamic properties of the user and system (e.g. time, place and currently activated roles) might influence this process. Once calculated, risk-threshold remains unchanged in a session.

Session with Adaptive Risk-Threshold (SAR): This is the most sophisticated session risk-threshold estimation model. In SAR, session risk-threshold is first estimated before the creation of the session, as in SDR. However, based on the user activities during the session, the system could decrease or increase the value. Therefore, the system needs to monitor user activities during the session. Any detected abnormal or malicious activities should lower the risk-threshold and thereby limit further suspicious activities. Therefore, a system automated role deactivation process is required in SAR to deactivate risky roles according to adjusted risk threshold and prevent further re-activation of such roles.

As mentioned earlier, we assume that each role is associated with a quantified risk value that is indicative of the criticality of that role. Given this, the session risk-threshold as estimated by various schemes discussed above (SSR, SDR and SAR) limits what activities a user can perform in that session. Risk measurement of a role might be affected by several factors, e.g., the cost of the permissions that are assigned to it and role dependencies. Any role activation request triggers the system to verify the session risk-threshold with the risk of this new role to be activated. If activation of a role does not exceed the session risk-threshold then the activation is permitted. Otherwise, it is denied or could cause deactivation of already activated roles from the session. These details are discussed in section 3.

In a session, any user attempt to perform a task might require role activation which could happen either by a user's direct attempt to activate a role in *role level* user-system interaction or user's attempt to perform a task in the system (i.e., exercise a permission) with *permission level* interaction. In role level interaction, a user explicitly mentions the role that she wants to activate. In permission level, the system needs to find if there is a role assigned to the user with the requested permission. Section 3 discusses different issues in role activation for these two types of interactions. For example, role activation could be completely controlled by the user or could be aided or completely automated by the system without user involvement. Also, certain roles may need to be deactivated as a consequence of activation of other roles to maintain session's risk under the threshold.

3 User Driven Role Activation Frameworks

Our overall approach is as follows. A session risk_threshold parameter places an upper bound on the risk that a session can carry. A present_risk parameter specifies the current risk of a session. As mentioned earlier, many techniques could be employed to estimate these two parameters. Our goal is to develop a framework of models for role activation given these two parameters. We develop two separate frameworks identifying various issues related to role activation based on role-level and permission-level interactions.

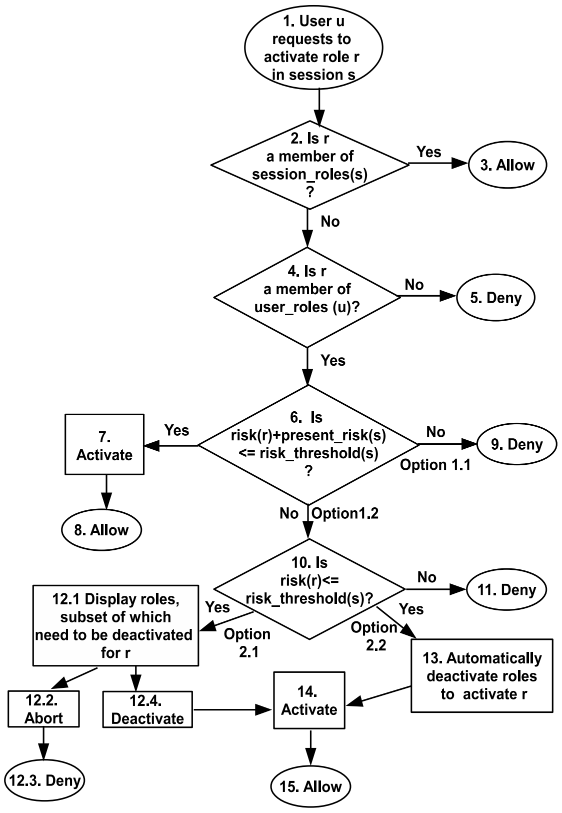
1. User u requests to activate role r in session s
2. Is r a member of session_roles(s) ?
Yes
3. Allow
No
4. Is r a member of user_roles (u)?
No
5. Deny
Yes
6. Is risk(r)+present_risk(s) <= risk_threshold(s) ?
Yes
7. Activate
8. Allow
No
Option 1.1
9. Deny
No
Option1.2
10. Is risk(r)<= risk_threshold(s)?
No
11. Deny
Yes
Option 2.1
12.1 Display roles, subset of which need to be deactivated for r
Yes
Option 2.2
13. Automatically deactivate roles to activate r
12.2. Abort
12.4. Deactivate
14. Activate
12.3. Deny
15. Allow

Fig. 2. Role Activation in Role-Level User Interaction

3.1 Role Level Interaction

Generally users in RBAC request to activate a particular role in a session and the system could allow or deny the request. Figure 2 shows the steps involved in this process. It starts with a user request to activate a role in a session and ends with an Allow or Deny decision. After receiving a request from user u, the system first checks if the requested role r is available in session_roles(s), which is the set of currently activated roles in session s. If so, the system returns otherwise, it checks the assigned_roles set that contains all roles authorized for u. The request is simply denied if role r is not present in assigned_roles.[1] Otherwise, the system compares if addition of r increases present_risk of the session beyond the risk_threshold. If not, the activation is allowed. Note that present_risk is the combined risk of all activated roles in a session that indicates the risk the session is currently carrying. If the risk_threshold would be exceeded, the system can either deny the request or attempt to deactivate some prior activated role(s) from the session in order to

[1] For simplicity, we assume core RBAC with no hierarchy among the roles. Extension to hierarchical RBAC is straightforward but tedious.

reduce present_risk before activating r, provided the risk of r is less than session risk_threshold. If so, the system could automatically deactivate necessary roles. Alternatively, it can display possible combinations of roles for deactivation from which the user might select one option. On successful deactivation, the system activates the requested role. The user may also cancel deactivation and abort the activation process.

There are three different types of activation models that could be constructed by choosing different options from this framework:

– **Strict Activation:** This activation could be constructed if option 1.1 in Figure 2 is chosen. In this approach, the system activates the requested role if it satisfies risk_threshold or denies otherwise.
– **Activation with System Guided Deactivation:** Combination of options 1.2 and 2.1 in Figure 2 yields this model. If activation of a role exceeds the risk_threshold, the system suggests the user to deactivate prior activated roles from session_roles to keep present_risk within session risk_threshold.
– **Activation with System Automated Deactivation:** In this process the system automatically deactivates roles from session_roles for activating a role. This model could be constructed by options 1.2 and 2.2 in Figure 2.

Many strategies could be employed at different levels of sophistication for each of the above models. For example, in system automated deactivation above, the system could employ simple algorithms for deactivation such as least recently used role or more sophisticated algorithms based on machine learning and heuristics that captures user activity patterns and selects the most appropriate role.

3.2 Permission Level Interaction

In many practical systems, user-system interaction is permission level instead of role level. Users keep doing their job and the system automatically checks authorization, e.g., a bank teller may try to obtain a statement for a customer and the system checks if she has the necessary role(s) activated in the session. If not, it may find an appropriate role to be activated to enable the user's action. Figure 3 shows a framework of role activation models for such interactions. It starts when a user tries to exercise a permission or perform a task and ends with an Allow or Deny. A request for permission p from user u can be approved if p is present in the session_permissions set. Otherwise, the system finds a role assigned with p in the assigned_roles set of u. The request is simply denied if no such role is found. Otherwise, if there is such a role r, the system activates it provided increased present_risk from activation of r stays within the risk_threshold of the session s. If there is more than one such role, the system might automatically select and activate one. There are different ways this selection could be performed, e.g., less risky role, role with minimum permissions or role relevant to user activities in that context. Alternatively, the system could ask the user to select one of them for activation. Again, there might be a case when there are multiple roles with p whose individual risk is less than risk_threshold of s, however, activation is not

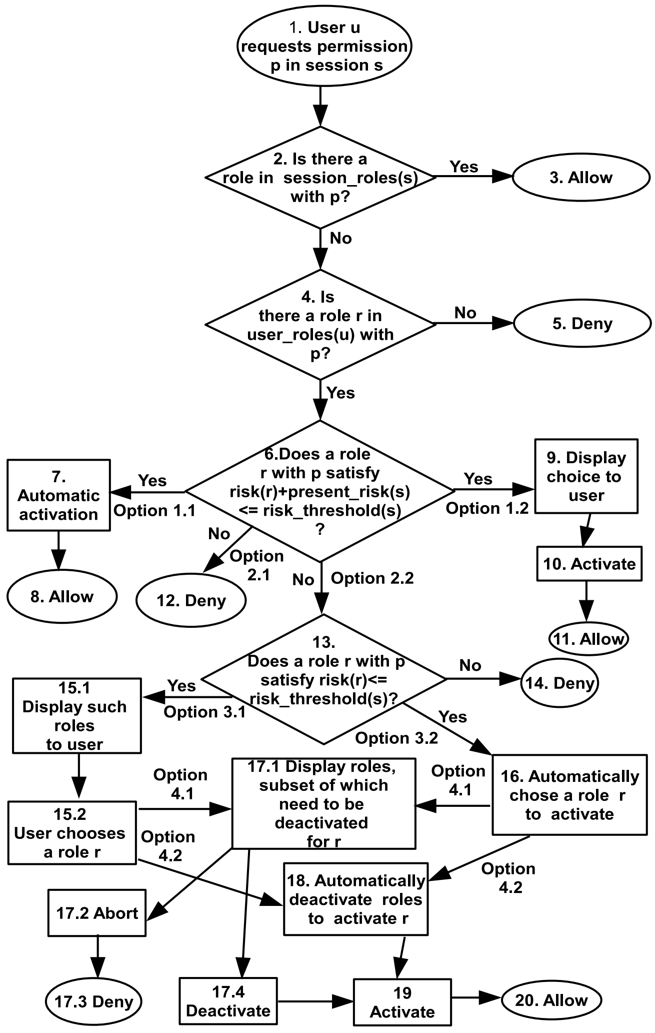
1. User u requests permission p in session s
2. Is there a role in session_roles(s) with p?
Yes
3. Allow
No
4. Is there a role r in user_roles(u) with p?
No
5. Deny
Yes
6.Does a role r with p satisfy risk(r)+present_risk(s) <= risk_threshold(s) ?
Yes
Option 1.1
7. Automatic activation
8. Allow
Yes
Option 1.2
9. Display choice to user
10. Activate
11. Allow
No
Option 2.1
12. Deny
No
Option 2.2
13. Does a role r with p satisfy risk(r)<= risk_threshold(s)?
No
14. Deny
Yes
Option 3.1
15.1 Display such roles to user
15.2 User chooses a role r
Yes
Option 3.2
16. Automatically chose a role r to activate
Option 4.1
Option 4.2
17.1 Display roles, subset of which need to be deactivated for r
Option 4.1
Option 4.2
18. Automatically deactivate roles to activate r
17.2 Abort
17.3 Deny
17.4 Deactivate
19 Activate
20. Allow

Fig. 3. Role Activation in Permission-Level User Interaction

possible without deactivation of other roles to maintain present_risk under the threshold. At this point, following the selection of a role r to be activated by the user, the system determines roles that need to be deactivated. As in role level interaction, there are two different deactivation processes. Finally, a successful deactivation allows activation of r. From this framework different activation models could be constructed as follows.

– **Strict Role Activation:** Activation is allowed if it satisfies risk_threshold of the session, otherwise denied. This model could be constructed by combining either options 1.1 and 2.1 or 1.2 and 2.1 in Figure 3.

- **System Automated Role Activation:** In this scheme, the system automatically chooses a role r for activation of the requested permission p and the activation process might need deactivation of prior activated roles. This deactivation could be done automatically or by user's choice. Such an activation model could be constructed by combining options 1.1, 2.2, 3.2 and 4.1 or 1.2, 2.2, 3.2 and 4.2 or 1.1, 2.2, 3.2 and 4.1 or 1.1, 2.2, 3.2 and 4.2 in Figure 3.
- **System Guided Role Activation:** In this scheme, the system asks the user to select a role r from a possible set of roles with requested permission p and activation of any of them might cause deactivation of prior activated roles. This model could be constructed by choosing either options 1.2, 2.2, 3.1 and 4.1 or options 1.2, 2.2, 3.1 and 4.2 in Figure 3.

3.3 Instantiation of Prior Models

In this section, we discuss how existing risk-aware RBAC models in the literature relate to our framework. Specifically, we show that these models are special instances of our role activation framework.

Baracaldo et al [2] provide a trust-and-risk aware role activation algorithm. Besides restricting role activation by well-known dynamic separation of duty and cardinality constraints, it further restricts roles with risk value higher than trust of user. It also finds role-set with minimum risk for a requested set of permissions. In our framework for permission level interaction, this is a strict activation with options 1.1 and 2.1 in Figure 3 in a session with dynamic risk threshold. Here, session risk_threshold is the trust of the user and step 6 in Figure 3 creates candidate role-set for activation in which risk of each role is less than risk_threshold. Then the system activates roles with minimum risk from the role-set.

Salim et al [11] consider user risk as a budget and allow accesses according to budget availability. A user interacts with the system at permission level and for each requested permission, corresponding roles are displayed to the user with their individual weight (risk) so that the user can activate minimum weighted role. They do not discuss session or role activation processes. We assume a role can only be activated if its weight is within user's available budget and after exercising each permission, the cost of the permission is deducted from the user budget. We also assume that it is a one time activation, that is, after exercising the permission the role is deactivated and the next requested permission repeats the role activation process. We configure this process as strict activation with options 1.1 and 2.1 in Figure 3 and risk_threshold is simply the user budget in a session with dynamic risk_threshold. In step 6 of Figure 3, the role's individual cost is compared with the risk_threshold and after exercising the permission the role is deactivated.

Chen et al [3] provide three different ways to estimate user-permission pair risk and allow user access if risk of respective pair stays below the permission risk threshold. Here an access might create user obligatory actions which is beyond our consideration in this paper. Risk of a user-permission pair might also vary

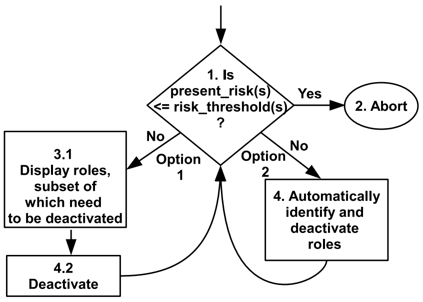
1. Is present_risk(s) <= risk_threshold(s) ?
Yes
2. Abort
No
Option 1
3.1 Display roles, subset of which need to be deactivated
4.2 Deactivate
No
Option 2
4. Automatically identify and deactivate roles

Fig. 4. System Automated Role Deactivation

for different sessions and they consider each of them as user-session-permission risk. If more than one role of a user in a session can exercise a permission p, only the role with lowest risk is allowed to exercise p. In our framework, this is a strict activation with options 1.1 and 2.1 of Figure 3 in a session. Instead of a risk_threshold, it contains risk values for each user-session-permission for activated roles and user can activate any role that she is authorized. However, to exercise a permission the system automatically picks the role with lowest risk.

4 Risk-Adaptive Role Deactivation

Systems that employ sessions with static or dynamic risk threshold (SSR/SDR) discussed in section 2 have certain limitations. A malicious entity that takes control of a session may still obtain all the power of the user that owns the session in a piecemeal manner. For example, suppose that a session risk_threshold is set at 30 and that every role assigned to the user of the session has a risk value below 30. Even though the aggregate risk of all roles assigned to the user may be above the risk_threshold of 30, the malicious entity can activate and deactivate one role at the time and accomplish most, if not all the tasks that the user is capable of. Since SSR and SDR schemes do not adjust the session risk_threshold over the period of the session, they cannot address this issue. However, sessions with adaptive risk_threshold (SAR) adjust session risk_threshold value by monitoring the activities in a session. By adaptively reducing the threshold, the user is forced to deactivate certain roles and prevented from further reactivation of such roles. This contains further damages that could be caused by a malware that takes control of the session. Nevertheless, note that SSR and SDR are still useful and practical schemes. Following the earlier example, in an SDR scheme, a malware would never be able to activate roles whose risk value is above 30 since the risk_threshold is set at 30. Thus the risk_threshold could prevent certain roles from being ever activated in a suspicious session, for example.

Figure 4 shows a framework for system automated role deactivation models. We believe a continuous monitoring process is necessary to detect abnormal or malicious activities within a session. On successful detection, the system lowers the risk_threshold to stop certain activities. Every time the threshold changes,

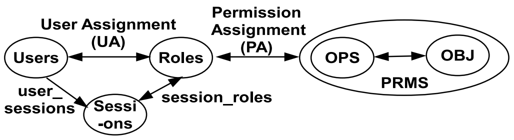

Fig. 5. Core RBAC

the system automatically calls the deactivation function to remove certain roles affected by the changing threshold. There are two different ways this process could happen: the system could automatically deactivate the roles or force the user to deactivate them by providing her some choices on what roles to be deactivated. Unlike SSR and SDR, this system initially might grant certain user permissions in less risky situation and be able to revoke them if the situation becomes more risky.

5 Formal Specifications

We provide a formal specification of a system guided role activation model for a session with dynamic risk_threshold (SDR) in permission level user-system interaction. Our specified model could be constructed by selecting options 1.2, 2.2, 3.1 and 4.1 in Fig 3. Our formal specification extends the NIST Core RBAC model [6].

5.1 Overview of NIST Core RBAC

Core RBAC provides a fundamental set of elements, relations and functions required for a basic RBAC system. These elements are shown in Fig 5. The set of elements contain users ($USERS$), roles ($ROLES$), operations (OPS), objects (OBJ) and permissions (PRMS). There are many-to-many mapping relations such as user-to-role (UA) and permission-to-role (PA) assignment relations. $PRMS = 2^{OPS \times OBJ}$, is a set of permissions in which each (OPS,OBJ) pair indicates an operation that could be performed on an object. The Core RBAC model also includes a set of sessions ($SESSIONS$) where each session is a mapping between a user and an activated subset of roles that are assigned to the user. Each session maps one user to a set of roles, that is, a user establishes a session during which the user activates some subset of roles that he or she is assigned. Each session is associated with a single user and each user is associated with one or more sessions. A *session_roles* function gives the roles activated in the session and a *user_sessions* function gives the set of sessions that are associated with a user. Details of the relation and functional specification of this model are provided in [6]. In the following section, we only discuss the additional and modified functions and elements that are required for our selected model.

5.2 Specification of NIST Core RBAC Risk-Aware Session Model

In this model, each permission is associated with a risk value that is indicative of damages that can occur if compromised. For simplicity, the risk of a role is considered as the sum of all permissions assigned to it. Here a user creates a session and continues requesting permissions to perform her job. During session creation, the system dynamically calculates session *risk_threshold* and keeps activating roles for requested permissions within the threshold. Both permission risk and *risk_threshold* are positive real numbers ($\mathbb{R}_{\geq 0}$). We formally define:

– $assigned\_risk : PRMS \rightarrow \mathbb{R}_{\geq 0}$, a mapping of permission p to a positive real value, which gives the risk assigned to a permission.
– $risk\_threshold : SESSIONS \rightarrow \mathbb{R}_{\geq 0}$, a mapping of session s to a positive real number that gives the maximum risk the session could contain.
– $present\_risk : SESSIONS \rightarrow \mathbb{R}_{\geq 0}$, a mapping of session s to a positive real number that gives the present risk value of the session.

We assume that the above three pieces of information are always available in our model. It also contains administrative functions to create and maintain elements and system functions for session activity management. In the following, note that a regular user can only call the CreateSession and PerformTask functions. All the other functions are administrative/system functions. In the function parameter, NAME is an abstract data type whose elements represent identifiers of various entities in the RBAC system.

AssignRisk: This administrative function assigns a risk value to a permission.

```
1: function $AssignRisk(ops, obj : NAME, risk : \mathbb{R}_{\geq 0})$
2:     if $ops \in OPS$ and $obj \in OBJ$ then
3:         $assigned\_risk'(ops, obj) \leftarrow risk$
4:     end if
5: end function
```

RoleRisk: This function returns estimated risk of a role. It takes role as an input and returns the sum of its assigned permissions' risk.

```
 1: function $RoleRisk(role : NAME, result : \mathbb{R}_{\geq 0})$
 2:     /*The value of $result$ is initially 0*/
 3:     if $role \in ROLES$ then
 4:         for all $ops \in OPS$ and $obj \in OBJ$ do
 5:             if $((ops, obj) \mapsto role) \in PA$ then
 6:                 $result' \leftarrow result + assigned\_risk(ops, obj)$
 7:             end if
 8:         end for
 9:     end if
10: end function
```

CreateSession: A user creates a session using this function. Initially the session does not contain any role. It utilizes an *evaluate_risk* function to calculate the *risk_threshold* of a given user. Functionality of *evaluate_risk* should be application specific, thus, we do not specify the details of this function. The *present_risk* contains the sum of activated roles' risk in the session which is initially 0.

```
function $CreateSession(user : NAME, session : NAME)$
    if $user \in USERS$ and $session \notin SESSIONS$ then
        $SESSIONS' \leftarrow SESSIONS \cup \{session\}$
        $user\_sessions'(user) \leftarrow user\_sessions(user) \cup \{session\}$
        $risk\_threshold'(session) \leftarrow evaluate\_risk(session, user)$
        $present\_risk'(session) \leftarrow 0$
    end if
end function
```

PerformTask: In a session, a user can invoke this function to access a permission. Note that, this is the first step of the flowchart shown in Fig 3. This function takes a access request of the user in a session and calls *CheckAccess* to verify if the necessary role is activated in that session. If *CheckAccess* returns true it allows the user request and deny otherwise.

```
function $PerformTask(user, session, obj, ops : NAME, result : BOOL)$
    if $session \in SESSIONS$ and $ops \in OPS$ and
       $obj \in OBJS$ and $user \in USERS$ then
        if $CheckAccess(user, session, obj, ops) = true$ then
            $result \leftarrow true$
        else
            $result \leftarrow false$
        end if
    end if
end function
```

CheckAccess: *CheckAccess* is called for each user access requests to check whether the session has the necessary role activated . If the role is not activated, it calls *AddActiveRole* for activating the necessary role if any. On successful activation, it returns true.

```
function $CheckAccess(user, session, obj, ops : NAME, result : BOOL)$
    if $session \in SESSIONS$ and $ops \in OPS$ and
       $obj \in OBJS$ and $user \in USERS$ then
        for all $r \in session\_roles(session)$ do
            if $((ops, obj) \mapsto r) \in PA$ then
                $result \leftarrow true; return$
            end if
        end for
        if $AddActiveteRole(user, session, obj, ops) = true$ then
            $result \leftarrow true$
        else
            $result \leftarrow false$
        end if
    end if
end function
```

AddActiveRole: Unlike RBAC_0, this function cannot be explicitly invoked by a user, rather, it is called by the system to activate a role for a permission requested by a user within a session. First, the function checks *assigned_users* set to find if there is a role with the requested permission that is authorized for the user. If there is no such role, it returns false as activation failure. If roles are present and could be activated within the session *risk_threshold*, it asks the user to select a role. After the user's selection, it activates the role and returns true. Alternatively, roles with the requested permission might be available with risk value less than the session's *risk_threshold*, however, its addition would exceed the *present_risk* due to already activated roles in the session. In such cases, the system displays the roles that could be deactivated and the user selects some of them. Then the *Deactivation* function is called and after necessary deactivation, the system activates the selected role and returns true, otherwise, returns false.

```
function $AddActiveRole(user, session, obj, ops : NAME, result : BOOL)$
    if $session \in SESSIONS$ and $ops \in OPS$ and
       $obj \in OBJS$ and $user \in USERS$ then
        $roleOptions \leftarrow \{\emptyset\}$ /*Set of roles to display, initially empty set*/
        /*Find roles that can be activated within risk_threshold*/
        for all $r \in ROLES$ and $user \in assigned\_users(r)$ do
            if $((ops, obj) \mapsto r) \in PA$ and $session\_risk(session)+$
             $RoleRisk(r) \leq risk\_threshold(session)$ then
                $roleOptions' \leftarrow roleOptions \cup \{r\}$
            end if
        end for
        if $roleOptions \neq \{\emptyset\}$ then /* If there are roles to activate*/
            $sr = SelectRoles(roleOptions)$ /* Roles are displayed to $user$*/
            /*and $user$ select role $sr$ to activate and system activates the $sr$*/
            $session\_roles'(session) \leftarrow session\_role(session) \cup \{sr\}$
            $session\_risk'(session) \leftarrow session\_risk(session) + RoleRisk(r)$
            $result \leftarrow true; return$
        else/*Find relevant roles with $RoleRisk$ less than the risk_threshold*/
            for all $r \in ROLES$ and $user \in assigned\_users(r)$ do
                if $((ops, obj) \mapsto r) \in PA$ and $RoleRisk(r) \leq$
                $risk\_threshold(session)$ then
                    $roleOptions' \leftarrow roleOptions \cup \{r\}$
                end if
            end for
            if $roleOptions \neq \{\emptyset\}$ then /*User selects roles from $roleOptions$*/
          /*and $Deactivation$ function is called*/
                $sr \leftarrow SelectRoles(roleOptions)$
                if $Deactivation(session, sr) = true$ then
                    $session\_roles'(session) \leftarrow session\_role(session) \cup \{sr\}$
                    $session\_risk'(session) \leftarrow$
                        $session\_risk(session)+RoleRisk(sr)$
                    $result \leftarrow true; return$
```

```
32:                 end if
33:             end if
34:         end if
35:     end if
36:     $result \leftarrow false$
37: end function
```

Deactivation: This function deactivates the roles from the session to activate the requested role *sr*. On successful deactivation, it returns *true* and *false* otherwise. Similar to *AddActiveRole* this function can not be invoked by a user.

```
 1: function $Deactivation(session, sr : NAME, result : BOOL)$
 2:     if $session \in$SESSIONS then
 3:         $roleOptions \leftarrow \{\emptyset\}$ /*Set of roles to display, initially empty set*/
 4:         /*Create $roleOptions$ that contains roles to be deactivated*/
 5:         for all $r \in session\_roles(session)$ do
 6:             if $session\_risk(session) + RoleRisk(rs) - RoleRisk(r)$
 7:               $\geq risk\_threshold(session)$ then
 8:                 $roleOptions' \leftarrow roleOptions \cup \{r\}$
 9:             end if
10:         end for
11: /*Call $DeactivationSelect$ to get approval from user to deactivate $roleOptions$*/
12:         if $DeactivationSelect(roleOptions) = true$ then
13:             for all $r \in roleOptions$ do
14:                 $session\_roles'(session) \leftarrow session\_role(session) - \{r\}$
15:                 $session\_risk'(session) \leftarrow session\_risk(session)$-$RoleRisk(r)$
16:             end for
17:             $result \leftarrow true; return$
18:         end if
19:     end if
20:     $result \leftarrow false$
21: end function
```

6 Related Work

Several approaches have been proposed for combining risk issues in different access control systems. Kandala et al [7] provide a framework that identifies different risk components for a dynamic access control environment. The Jason report [10] proposes three core principles for a risk-aware access control system: measuring risk, identifying tolerance levels of risk and controlling information sharing. Cheng et al [4] give a model to quantify risk for access control and provide an example for multilevel information sharing. Ni et al [9] propose a model for estimating risk and induce fuzziness in the access control decision of the Bell-Lapadula model. Moloy et al [8] propose a risk-benefit approach for avoiding communication overhead in distributed access control. All of these models mostly focus on how to estimate risk. In contrast, our work focusses on how to utilize such risk measures in role activation and deactivation in a concrete RBAC model. There are also other approaches to achieve automated threat response in dynamically changing environments. Autrel et al [1] propose a reaction policy model for organizations in dynamic organizational environments and different threat contexts (e.g. buffer overflow, brute force attack, etc.). Debar et at [5] propose a more sophisticated approach in which threat contexts and new policy instances are dynamically derived for every threat alert.

7 Conclusion and Future Work

We enrich a system's capabilities, that implements RBAC, by dynamically controlling user activities in a session according to risk in the current situation. We show that there are three different points of time and processes a risk could be estimated in a session: static, dynamic and adaptive. We also develop two separate frameworks for role activation models where the user-system interaction is either role level or permission level. We also develop system automated role deactivation process by which a session with adaptive risk threshold can decrease access capability of a user in session whenever it is necessary. Finally, we provide NIST RBAC style formal specification of one of the models instantiated from our framework. In the future, we plan to investigate other models in our framework and study them in the context of more advanced NIST RBAC models.

Acknowledgement. This work is partially funded by a AFOSR MURI project.

References

1. Autrel, F., Cuppens-Boulahia, N., Cuppens, F.: Reaction Policy Model Based on Dynamic Organizations and Threat Context. In: Gudes, E., Vaidya, J. (eds.) Data and Applications Security 2009. LNCS, vol. 5645, pp. 49–64. Springer, Heidelberg (2009)
2. Baracaldo, N., Joshi, J.: A trust-and-risk aware rbac framework: tackling insider threat. In: SACMAT 2012, pp. 167–176. ACM, New York (2012)
3. Chen, L., Crampton, J.: Risk-Aware Role-Based Access Control. In: Meadows, C., Fernandez-Gago, C. (eds.) STM 2011. LNCS, vol. 7170, pp. 140–156. Springer, Heidelberg (2012)
4. Cheng, P.-C., Rohatgi, P., Keser, C., Karger, P., Wagner, G., Reninger, A.: Fuzzy multi-level security: An experiment on quantified risk-adaptive access control. In: Security and Privacy, 2007, pp. 222–230 (May 2007)
5. Debar, H., Thomas, Y., Cuppens, F., Cuppens-Boulahia, N.: Enabling automated threat response through the use of a dynamic security policy. Journal in Computer Virology, 195–210 (2007)
6. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed nist standard for role-based access control. ACM Tran. Inf. Sys. Sec. (2001)
7. Kandala, S., Sandhu, R., Bhamidipati, V.: An attribute based framework for risk-adaptive access control models. In: Avail., Reliab. and Sec., ARES (August 2011)

8. Molloy, I., Dickens, L., Morisset, C., Cheng, P.-C., Lobo, J., Russo, A.: Risk-based security decisions under uncertainty. In: CODASPY 2012 (2012)
9. Ni, Q., Bertino, E., Lobo, J.: Risk-based access control systems built on fuzzy inferences. In: ASIACCS 2010, pp. 250–260. ACM, New York (2010)
10. M. C. J. P. Office: Horizontal integration: Broader access models for realizing information dominance. MITRE Corporation, Tech. Rep. JSR-04-132 (2004)
11. Salim, F., Reid, J., Dawson, E., Dulleck, U.: An approach to access control under uncertainty. In: Avail., Reliab. and Sec., ARES, pp. 1–8 (August 2011)
12. Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role-based access control models. Computer 29(2), 38–47 (1996)