A Framework for Risk-Aware Role Based Access Control

Khalid Zaman Bijon*, Ram Krishnan† and Ravi Sandhu*
*Institute for Cyber Security & Department of Computer Science
†Institute for Cyber Security & Department of Electrical and Computer Engineering
University of Texas at San Antonio

***Abstract*—Over the years, role based access control (*RBAC*) has remained a dominant form of access control both in the industry and academia. More recently, the need for risk awareness in access control has received considerable attention in the research community in light of issues such as insider threats. Although *RBAC* facilitates risk mitigation via features such as constraints (e.g. static and dynamic separation of duty), a *quantified* approach of risk awareness/mitigation has emerged as a promising research theme due to its inherent flexibility. In this approach, risk/cost metrics are computed for various entities involved in access control such as users and objects and a risk threshold limits the permissions that can be exercised. The quantified approach accommodates dynamism in access decisions based on contexts/situations such as an employee accessing a sensitive file using a work computer versus accessing using her own device. In this paper, we analyze the difference between the traditional constraint-based risk mitigation and the recent quantified risk-aware approaches in *RBAC* and propose a framework for introducing risk-awareness in *RBAC* models that incorporates quantified-risk. We also provide a formal specification of an adaptive risk-aware *RBAC* model by enhancing the NIST core *RBAC* model.**

Keywords: Access Control, Policy, RBAC, Risk

I. Introduction

The concept of constraints-based risk mitigation has been well-studied in role based access control (*RBAC*) models. For instance, separation of duty and role cardinality constraints of *RBAC* are all concerned about risk mitigation. Generally, constraints are static in nature as they are predefined policies that always give the same outcome regardless of the situation. Such a static risk mitigation approach fails to adapt to varied and changing circumstances under which access decisions are made in modern systems. We call the constraints-based risk mitigation approach as "traditional" risk-awareness approach.

Recently, a *quantified* approach to risk-awareness in access control has drawn much attention as the need for agile and dynamic access control has emerged. Several works have been published in this arena [6]–[8], [15], [18]–[20], mainly attempting to assess and utilize risk metrics in different access control systems. A quantified risk-aware access control system differs from the traditional ones in that it permits/denies access requests dynamically based on estimated risk instead of the predefined ones which always give the same outcome. In quantified risk-aware access control, risk is represented as a metric where, for example, the higher its value associated with a user, the higher the chances that the user will perform inappropriate actions. (Consequently, an access request involving a user or a resource with a higher risk-value poses more security threat/risk to the system than a request by a user or for a resource with a lower risk value.) Although a number of works have investigated quantified risk in *RBAC* [4], [6], [20], a systematic analysis of various components in RBAC that can utilize risk has not been performed. According to [19], a practical quantified 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. Risk assessment and utilization are equally important for a risk-aware access control system. Risk assessment process could be of different types (e.g. probabilistic) that best suits the system context and different factors of the system can influence this process. On the other hand, risk utilization process determines how estimated risk can influence decision making process of an access control system. Assessment of the risk is beyond the scope of this paper, rather, it is solely focused on risk utilization processes in *RBAC* by assuming that the value of risk is somehow computed and readily available.

In this paper, we propose a framework for risk-aware *RBAC* models. We identify the components of *RBAC* that can be made risk-aware and hence require changes in how they behave and interact with other components. We also analyze the basic distinction between traditional and quantified approach of risk-awareness in *RBAC*. We identify two types of quantified risk, i.e. non-adaptive and adaptive, and analyze the necessary functionalities for utilizing them in an *RBAC* system. Finally, we formalize the adaptive risk-aware *RBAC* by enhancing the NIST Core *RBAC* model.

The remainder of the paper is organized as follows. Section II presents our proposed framework for risk-aware *RBAC*. The formal specification of adaptive risk-aware *RBAC* is presented in Section III. Section IV discusses related work. We conclude in Section V.

II. The Framework for Risk-Aware RBAC

This section presents the framework for risk-aware *RBAC*. In this framework, the risk-aware components of *RBAC* are identified around which different approaches of utilizing risk-awareness can be developed. Then it presents the necessary elements in order to support both traditional and quantified risk-awareness in *RBAC*. Finally, it discusses the differences between these two approaches.

A. Risk-Aware RBAC Components

Fig 2 shows the components of *RBAC* that can be risk-aware. We discuss details of each component below:

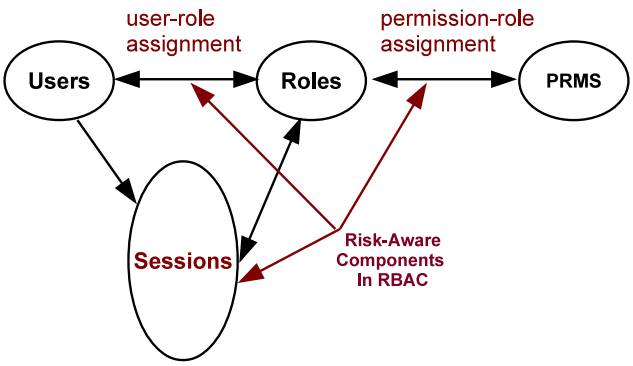

Fig. 1: *Risk-Aware* RBAC *Components*

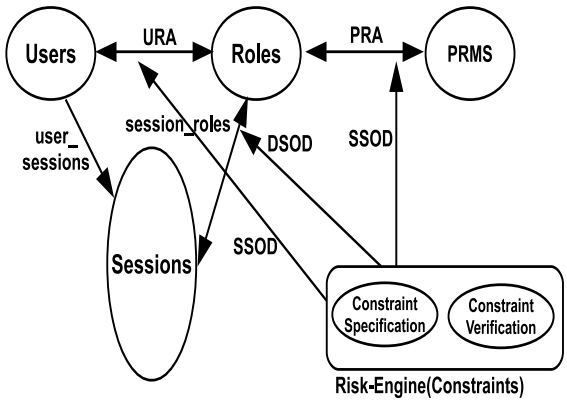

Fig. 2: *Traditional Risk-Aware RBAC*

- **User-Role Assignment (URA):** In *RBAC*, a role is a collection of permissions and a user is assigned one or more roles so they can perform particular tasks. Different users need to perform different tasks in an organization; hence permissions assigned to various roles often varies. For instance, in a banking organization, permissions of the "manager" and the "teller" role are not same. An organization needs to develop processes to prevent users from obtaining certain roles which are unauthorized or risky in a particular situation. In *RBAC*, the URA component is concerned about user-role assignments. In order to mitigate the risks of a user obtaining inappropriate roles, the URA component should be risk-aware.
- **Permission-Role Assignment (PRA):** In *RBAC*, permissions are assigned to roles and roles to users. A combination of permissions can be very powerful, consequently, might pose a significant risk to the system. Hence, these permissions should not be assigned to the same role. Also, assigning certain users to a particular role might might impose restrictions on the kind of permissions that can be assigned to that role. In *RBAC*, PRA provides the functionalities for permission-role assignment. Therefore, PRA should be risk-aware.
- **Session:** In *RBAC*, users need to create a session and activate one or more of their assigned roles to exercise certain privileges. Hence, the access capability of a user at a particular time is determined by the activated roles in their sessions. A certain combination of activated roles can be very risky for the system. Also, a user's different sessions in different situations, i.e. place, time, can pose different security risks to the system. For instance, a user's session from home may be considered as more risky than her session from office. Therefore, role activation/deactivation processes in *RBAC* sessions should be appropriate to address a particular risk and a proper risk-aware approach should be developed around this.

Note that an organization might develop a particular risk-aware approach to one or more of these components based on their requirements. Besides these three components in *RBAC*, role-hierarchy and constraints specification can also be risk-aware. Role-hierarchy provides mechanism to determine which permission a senior role should inherit from a number of junior roles. A combination of permissions from multiple junior roles can be very risky, hence, a particular risk-aware approach can be developed around this component to handle these issues. Similarly, for instance, a risk-aware approach can be developed around constraints to determine which constraint should apply in a particular situation.

B. Types of Risk-Awareness in RBAC

A discussed earlier, in *RBAC*, risk-awareness can be categorized into two types. One is the traditional approach in which risk is basically mitigated by specifying and enforcing constraints on the above identified risk-aware components. The other is a quantified approach where the value of risk of a certain risk-aware component is estimated that dynamically restricts certain activities. In the following, we discuss both types of risk-awareness and also analyze their basic differences.

- **Traditional Risk-Awareness:**
  Traditionally risk-awareness in access control systems are constraints driven. Fig 2 shows the components of the traditional risk-aware *RBAC*. Here, the functionality of the Risk-Engine is to specify and enforce constraints. Basically, the constraints are specified for each risk-aware component of the *RBAC* model, i.e. URA, PRA and sessions, and enforced. The purpose of the risk-engine is to specify and enforce the constraints to exercise certain policies dealing with risk, e.g. static separation of duty (SSOD), dynamic separation of duty (DSOD), etc. SSOD policies are specified for restricting certain conflicting roles to be assigned to a user, while the DSOD policies are specified for restricting certain roles to be simultaneously activated in one or more sessions of a user. Therefore, constraints implementing SSOD policies are applied to the functionalities of URA and constraints that implement DSOD policies are applied to the functionalities of sessions (i.e. role activation/deactivation). A considerable amount of literature exists on constraint specifications in *RBAC* [1], [10], [11], [17], [23], [24], [24]. As an example, RCL-2000 [1] gives a constraints specification language for specifying SSOD, DSOD and other constraints for the risk-aware components

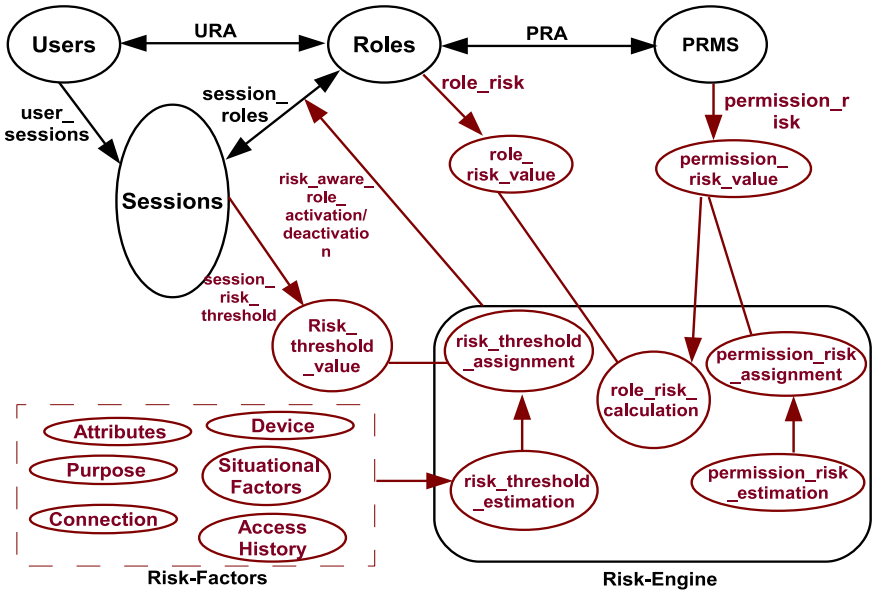

Fig. 3: *Non-Adaptive Quantified Risk-Aware RBAC Session*

of *RBAC*. Besides specifying constraints on user-role assignments and role activation/deactivation, RCL-2000 can also specify constraints on permission-role assignment to restrict certain conflicting permissions to be assigned to a role. Thus this type of constraint is applied to the PRA component of *RBAC*. Note that once constraints are specified, they remain the same unless an administrative user explicitly changes them. Thus during this period, the *RBAC* authorization policies would make the same decision regardless of the system's situation.

- **Quantified Risk-Awareness:** In the quantified approach, risk is represented as a metric where, for example, a higher value is more risky than the lower one. An estimated risk value helps the system to dynamically make decisions for the operations of the risk-aware components of *RBAC*. Fig 3 shows the components of the risk-aware *RBAC* sessions. As shown, for each user session 'risk-threshold' is estimated. Each permission is assigned a risk value, consequently, risk of a role is determined by combining its assigned permission risks. The session 'risk-threshold' and the risk of the roles are then compared to make decisions on role activation/deactivation within a session. Several approaches [6], [20] have been published in literature for calculating risk of roles and permissions. Similarly, the value of session 'risk-threshold' can be calculated and Fig 3 shows examples of risk-factors that might influence the estimated value. Similarly, a risk engine can be developed for the other risk-aware components: URA and PRA. For instance, for a quantified risk-aware URA, a 'risk-threshold' can be estimated for each user that is compared with risk of roles in order to determine which role that user can be assigned to. Similar to sessions, calculation of user 'risk-threshold' can be affected by several risk-factors. The quantified risk-aware approach can be further categorized into two types: non-adaptive and adaptive.

  Non-Adaptive Approach: In this approach, a risk value is estimated for a risk-aware component of *RBAC* and operations supported by that component are permitted/denied dynamically based on the risk value. Fig 3 shows the elements of the non-adaptive approach of the risk-aware *RBAC* session component. A 'risk-threshold' value is dynamically computed for each user session during the session creation. This computation might be influenced by a number of risk factors. Hence, the value of 'risk-threshold' varies for each user session depending on the values of the risk factors in that situation. The system needs to find proper 'risk-threshold' estimation process to combine risk-factors for computing a quantified 'risk-threshold' value. In Fig 3, the Risk-Engine contains a $risk\_threshold\_estimation$ function that estimates the 'risk-threshold' value of each session when it is created by the user. Note that, the computation algorithm is context dependent. The Risk-Engine also contains $risk\_threshold\_assignment$ function that assigns the value to each user session. There are also two functions called $permission\_risk\_estimation$ and $permission\_risk\_assignment$ to calculate and assign risk value for each permission. A function called $role\_risk\_calculation$ calculates the risk value of each role based on the assigned permissions to that role. In an *RBAC*session, a user needs to activate one or more roles to exercise certain permissions available to that user. Activation request for certain roles are authorized by the system by comparing session 'risk-threshold' and the risk value of the respective roles. The total risk of a session is determined by accumulated risks of the activated roles within a session and this risk value need to be always below the session

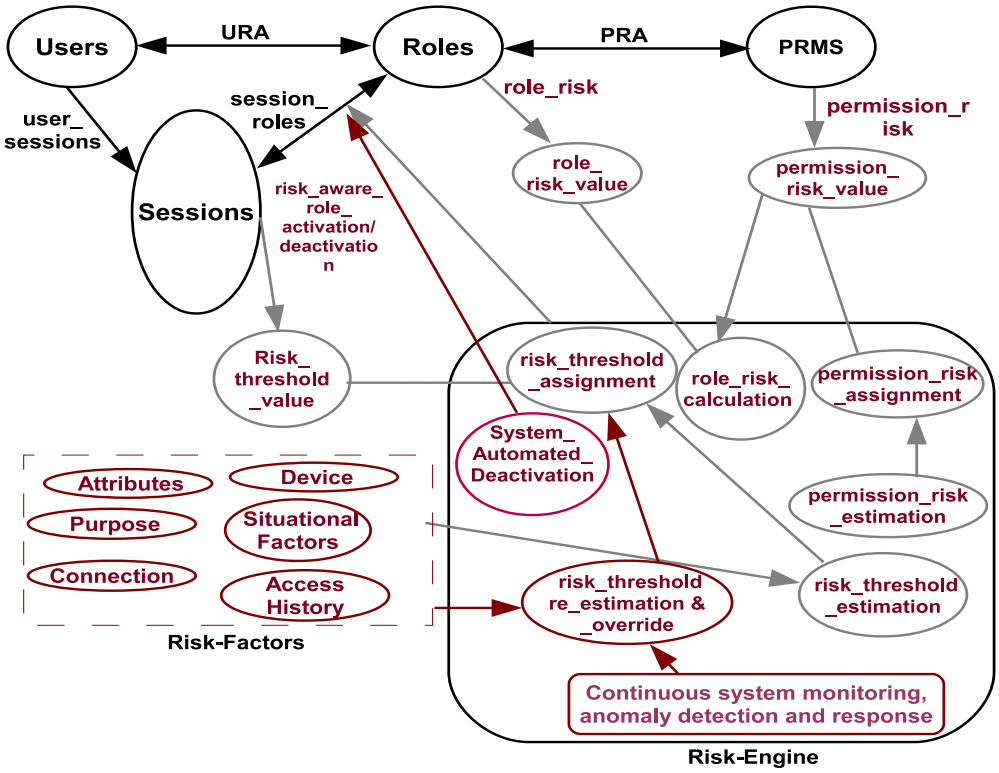

Fig. 4: *Adaptive Quantified Risk-Aware RBAC Session*

'risk-threshold'. Hence, certain role activation might cause deactivation of already activated roles from the session. Therefore, a user's access capability is dynamically determined based on the estimated 'risk-threshold' of the session. A framework for role activation/deactivation process within a risk-aware session is discussed in [4]. Note that, 'risk-threshold' value of each session is estimated during session creation time which remains unchanged throughout the life of that session. However, it might vary across different or subsequent sessions of a user.

Similarly, for a quantified risk-aware URA, the Risk-Engine should contain $risk\_threshold\_estimation$ and $risk\_threshold\_assignment$ functions to calculate 'risk-threshold' of each user to determine how risky the user is for the system. Now, authorization decision on user-role assignment is determined based on user 'risk-threshold' and risk of the respective roles. Role can also be assigned with a 'risk-threshold' based on several risk-factors, e.g. number of users assigned to it, time of the day, etc. Thus, for a quantified risk-aware PRA, the Risk-Engine should have proper $risk\_threshold\_estimation$ function for a role 'risk-threshold' value calculation and assigning permissions to that role is restricted by that the value.

Adaptive Approach: In the adaptive approach, estimated risk in various *RBAC* components can change frequently. There is an additional activity monitoring process in order to detect any abnormal behavior or incident in the system. On successful detection, the estimated 'risk-threshold' should be automatically lowered to stop certain risky operations. In some cases, the threshold can be increased to grant more permissions when the estimated risk is low. Fig 4, shows the elements of an adaptive approach of quantified risk-aware *RBAC* session. It requires a continuous monitoring process of users activities within a session and also a anomaly detection mechanism. If the system identifies any abnormal/malicious behaviors, the session 'risk-threshold' should be decreased appropriately to stop those malicious activities. For this purpose, in addition to the functionalities of the non-adaptive risk-aware session, Risk-Engine should also contain a function called $risk\_threshold\_re\_estimation$ in order to estimate a new 'risk-threshold' value and overwrite the previous one. In this process, a system $automated\_role\_deactivation$ process is required to stop certain risky user activities once the 'risk-threshold' value decreases by deactivating certain roles. (When 'risk-threshold' decreases, the system may automate re-activation of those roles.) Formal specification of these functionalities are presented in section III. Note that, the basic difference between adaptive and non-adaptive approach is that the adaptive process needs a system monitoring process and the Risk-Engine adaptively adjusts 'risk-threshold' based on users' activities during sessions and system context. While, non-adaptive approach can only calculate risk during each session creation and does not have run-

time monitoring and anomaly detection capability. The Risk-Engine of an adaptive risk-aware session, hence, is more complex than the non-adaptive one.

Similar to session, in URA and PRA, Risk-Engines should have monitoring and anomaly detection mechanism as well as $risk\_threshold\_re\_estimation$ process for users and roles respectively. Also, they should have system $automated\_role\_revocation$ and $automated\_permission\_revocation$ to automatically revoke roles and permissions from users and roles respectively.

The basic difference between traditional and quantified risk-aware approaches is that there is no explicit notion of the risk-value in traditional approach, while, the quantified approach has. In traditional approach, the administrative users identify the risky operations for certain users in the system and generate and enforce constraints to restrict these operations. Hence, risk-awareness in this approach is somehow static since a generated constraint always gives same decision on a request in every situation unless that constraint is further modified. Even though DSOD constraints involve users' activities in a session for it's decision making process, the restricting pattern is always same for a DSOD constraint across every-user sessions. For instance, if a DSOD constraint restricts that a user cannot activate two roles simultaneously in a session, this constraint will apply equally on all user-sessions which are created in different situations, e.g. time, place, etc. On the other hand, quantified risk-aware approach has explicit notion of risk value ('risk-threshold') which is estimated using particular formulas/processes. Hence, administrative users need to develop a proper 'risk-threshold' estimation process suitable for their context and the system dynamically calculate the value for a risk-aware component. This approach is more dynamic as 'risk-threshold' can vary based on situations and risk-factors, hence, the user activities are restricted by the 'risk-threshold' also varies. In addition to that, the adaptive quantified risk-awareness provides more automation in which the system can monitor and stop certain risky activities of the users.

III. Formal Specifications

We provide a formal specification of the functionalities for an adaptive risk-aware *RBAC* session. Our formal specification extends the NIST Core *RBAC* model [11]. A non-adaptive risk-aware *RBAC* session already formalized in [4] where user interacts with the system in permission-level. Here, our formal specification is developed in similar fashion where user interacts with the system in role-level. Similar to NIST Core *RBAC* model, the users explicitly request the system to activate a role. Also, user can explicitly deactivate an activated role from a session. Besides, our model introduces a system automated role deactivation process that tries to deactivate certain roles when the session $risk\_threshold$ changes. A session monitoring function is also developed to check if the session $risk\_threshold$ changes at a particular time so that it can call the deactivation function for deactivating necessary roles from the session.

A. Overview of NIST Core RBAC

Core *RBAC* provides a fundamental set of elements, relations and functions required for a basic RBAC system.

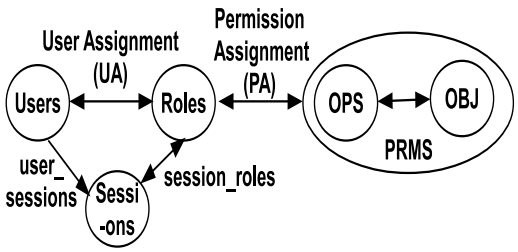

Fig. 5: *Core RBAC*

These elements are shown in 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 [11]. In the following section, we only discuss the additional and modified functions and elements of NIST core *RBAC* that are required for our adaptive risk-aware session model.

B. Specification of NIST Core RBAC Adaptive Risk-Aware Session Model

In this model, each permission is associated with a risk value that is indicative of damages that a system incurs in case of compromise. Since developing a function that calculates risk of each permission is application-context specific we assume that the risk value of each permission is already estimated and available. For simplicity, the risk of a role is considered as the sum of all permissions assigned to it. However more nuanced approaches for calculating a role risk can be employed. Here a user creates a session and tries to activate necessary roles in order to perform certain tasks. During session creation, the system calculates session $risk\_threshold$ and only activates user requested roles that satisfy the session $risk\_threshold$. Similar to the permission risk, we assume a function exists for calculating $risk\_threshold$ of each session. 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.

- $session\_risk$: $SESSIONS \rightarrow \mathbb{R}_{\geq 0}$, 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, we formally specify these functions. Note that, in this model a user can only call the CreateSession and AddActiveRole functions. All the other functions are administrative/system functions. Since it is an adaptive model, role deactivation is automatically taken care of by the system. 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.

function $AssignRisk(ops, obj : NAME, risk : \mathbb{R}_{\geq 0})$
  if $ops \in OPS$ *and* $obj \in OBJ$ **then**
    $assigned\_risk'(ops, obj) \leftarrow risk$
  end if
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.

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

CreateSession: A user creates a session using this function. Initially the session does not contain any role. It utilizes an Eval_RT function to calculate the $risk\_threshold$ based on the user involved and system context. The functionality of Eval_RT should be application specific, thus, we do not specify the details of this function. The $session\_risk$ 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**
    $SESSIONS' \leftarrow SESSIONS \cup \{session\}$
    $user\_sessions'(user) \leftarrow user\_sessions(user) \cup \{session\}$
    $risk\_threshold'(session) \leftarrow$ Eval_RT(session,user)
    $session\_risk'(session) \leftarrow 0$
  end if
end function

AddActiveRole: Similar to $RBAC_0$, this function is explicitly invoked by a user where the user explicitly mentions a role to activate. First, the function checks $assigned\_users$ set to find if the role is authorized for the user. If there is no such role, it returns false as activation failure. If the role is present and can be activated within the session $risk\_threshold$, it activates the role and returns true. Alternatively, requested role can be authorized for the user and its risk value is also less than the session's $risk\_threshold$, however, its addition would exceed the $session\_risk$ due to already activated roles in the session. In such cases, the $Deactivation$ function is called for deactivating already activated roles for decreasing the $session\_risk$ below $risk\_threshold$ and after necessary deactivation, the system activates the selected role and returns true, otherwise, returns false.

function $AddActiveRole(user, session, role : NAME, result : BOOL)$
  if $session \in SESSIONS$ *and* $role \in ROLES$ *and* $user \in USERS$ **then**
    /* if requested role activation satisfies the session $risk\_threshold$ the role is activated */
    if $session\_risk(session) + RoleRisk(r) \leq risk\_threshold(session)$ **then**
      $session\_roles'(session) \leftarrow session\_roles(session) \cup \{role\}$
      $session\_risk'(session) \leftarrow session\_risk(session) + RoleRisk(role)$
      $result \leftarrow true; return$
    else if $RoleRisk(role) \leq risk\_threshold(session)$ **then** /*Call Deactivation function to check if the role can be activated with deactivation of certain roles*/
      if $Deactivation(session, role) = true$ **then**
        $session\_roles'(session) \leftarrow session\_roles(session) \cup \{role\}$
        $session\_risk'(session) \leftarrow session\_risk(session) + RoleRisk(role)$
        $result \leftarrow true; return$
      end if
    end if
  end if
  $result \leftarrow false$
end function

Deactivation: This function deactivates the roles from the session to activate the requested $role$. It displays activated roles of the session to user and the user selects a role from them to deactivate. This function then deactivates the selected role and continues this process until the $session\_risk$ reduces below to $risk\_threshold$ of the session for activating the $role$. On successful required deactivations, it returns $true$ and $false$ otherwise. Note that, user can abort this process by selecting $null$ in DeactivationSelect and this function will return a $false$. This function can not be invoked by users.

function $Deactivation(session, role : NAME, result : BOOL)$
  if $session \in$ SESSIONS **then**
    $roleOptions \leftarrow session\_roles(session)$ /*Set of roles to display, initially all activated roles of the session*/
    while $session\_risk(session) + Rolerisk(role) \geq risk\_threshold(session)$ *or* $roleOptions \neq \{\phi\}$ **do** /*Call $DeactivationSelect$ to get one selected role from $roleOptions$ by user to deactivate*/
      $r = DeactivationSelect(roleOptions)$
      if r $\in session\_roles(session)$ **then**
        $session\_roles'(session) \leftarrow session\_roles(session) - \{r\}$
        $session\_risk'(session) \leftarrow session\_risk(session)$-$RoleRisk(r)$
        $roleOptins' = roleOptins - \{r\}$
      else if $r = null$ **then**

```
            break
        end if
    end while
    result ← true; return
  end if
  result ← false
end function
```

SActivityMonitor: This is one of the core system functions for an adaptive risk-aware session. System needs to call this function continuously during the entire life-cycle of a user session. This function first calls a session activity detection function SADetection to check if there is anything abnormal happening. The functionality of SADetection is application domain specific and beyond our scope. SADetection returns false if something is wrong and returns true otherwise. If it returns false, the function re-estimates the session $risk\_threshold$ by calling RE_Eval_RT and overwrites the current $risk\_threshold$. It then calls system automated deactivation function $SADeactivation$ to deactivate certain roles from the session.

```
function SActivityMonitor(user : NAME, session : NAME)
  if user ∈ USERS and session ∉ SESSIONS then
    if SADetection(session, user) = False then
      risk_threshold(session) ←
        RE_Eval_RT(session, user)
      SADeactivation(session)
    end if
  end if
end function
```

SADeactivation: This function is called by SActivityMonitor each time the $risk\_threshold$ of a session changes due to any abnormal behavior detection in the system. SADeactivation first calls IdentifyAffRoles function that returns a set of roles subset of which needs to deactivate in order to keep $session\_risk$ within $risk\_threshold$. The system can implement any algorithm for IdentifyAffRoles to identify risky roles and this function should be able to select the set of roles in which deactivating subset or all of them should reduce $session\_risk$ below to the $risk\_threshold$ of the session. Then, the roles are displayed to the user and user keep selecting roles for deactivation from the set. This process continues until the $session\_risk$ reduces below to the $risk\_threshold$ of the session.

```
function SADeactivation(session : NAME)
  if session∈SESSIONS then
    roleOptions ← {∅} /*Set of roles to display, initially
      empty set*/
    /*call IdentifyAffRole to create roleOptions that
      contains the roles need to be deactivated*/
    roleOptions = IdentifyAffRoles(session)
    while risk_threshold(session) ≤
session_risk(session) or roleOptions ≠ {φ} do
/*Call DeactivationSelect to get one selected role from
roleOptions by user to deactivate*/
      r = DeactivationSelect(roleOptions)
      if r ∈ session_roles(session) then
        session_roles'(session) ←
        session_roles(session) − {r}
        session_risk'(session) ←
        session_risk(session)-RoleRisk(r)
        roleOptins' = roleOptins − {r}
      else if r = null then
        Continue
      end if
    end while
  end if
end function
```

IV. Related Works

Traditional Risk-Aware Approaches(Constraints): Several authors have focussed on issues in constraints specification in access control systems primarily in RBAC. Constraints in RBAC are often characterized as static separation of duty (SSOD) and dynamic separation of duty (DSOD). These two issues were addressed back to late 1980's when Clark et al [9] introduced SSOD and Sandhu [21] DSOD. A number of attempts have been initiated afterwards to identify numerous forms of SSOD and DSOD policies [10], [24] and to specify them formally [12], [13] in RBAC systems. The RCL-2000 language for specifying these policies in a comprehensive way was proposed by Ahn et al [1]. Recently, Jin et al [14] proposed an attribute based access control model in which they provide an authorization policy specification language that could also specify constraints on attribute assignment. However, their constraints specification focuses on what values the attributes of subjects and objects may take given that users are currently assigned with particular attribute values. More Recently, Bijon et al [5] proposed an attribute based constraint specification language that can generate constraints on attribute assignments to entities based on intrinsic relationship among attributes. Besides *RBAC*, the traditional concept of risk-awareness has been well-studied in different access control systems including lattice-based access control (LBAC). In LBAC, strict $\star$ and liberal $\star$ properties [22] are all conceptually related to the risk-awareness in a system. Also, chinese-wall [22], the ethical barrier between organizations with conflict-of-interest, is another popular risk mitigation approach in LBAC. Recently, two different approaches has been proposed to mitigating risk in LBAC when organizations need to collaborate with outside specialist for ceratin reasons [2], [3].

Quantified Risk-Aware Approaches: Several approaches have been proposed for combining risk issues in different access control systems. Kandala et al [15] provide a framework that identifies different risk components for a dynamic access control environment. The Jason report [19] 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 [8] give a model to quantify risk for access control and provide an example for multilevel information sharing. Ni et al [18] propose a model for estimating risk and induce fuzziness in the access control decision of the Bell-Lapadula model. Moloy et al [16] 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 different risk-aware *RBAC* components. Recently, a quantified risk-aware *RBAC*sessions and role activation/deactivation framework have been proposed in [4]. There are also other approaches to achieve automated threat response in dynamically changing environments.

V. Conclusion and Future Work

In this paper, we developed a framework for risk-aware role based access control (*RBAC*) models. There are two basic requirements for developing a risk-aware *RBAC* model: 1. Identify components which can be risk-aware and thereby utilize risk metrics for various purposes and 2. Select a particular risk-aware approach and its necessary functionalities. We identify that in *RBAC* sessions, user-role assignment and permission-role assignment processes are the main risk-aware components. We also showed that risk-awareness can be of two types: traditional and quantified approaches. The former is the conventional constraint driven approach, while the latter is a risk metric driven approach. Furthermore, the quantified approach are of two types: non-adaptive and adaptive and develop necessary functionalities for them. Finally, we formalized the functionalities of adaptive risk-aware *RBAC* sessions by extending NIST standard *RBAC*. In the future, we plan to investigate introducing risk-awareness in more general models such as attribute-based access control.

Acknowledgement. This work is partially supported by the NSF (CNS-1111925) and AFOSR MURI grants (FA9550-08-1-0265).

References

[1] Gail-Joon Ahn and Ravi Sandhu. Role-based authorization constraints specification. *ACM Trans. Inf. Syst. Secur.*, 3(4):207–226, November 2000.

[2] Khalid Bijon, Ravi Sandhu, and Ram Krishnan. A group-centric model for collaboration with expedient insiders in multilevel systems. In *International Symp. on Security in Collaboration Technologies and Systems*, 2012.

[3] Khalid Zaman Bijon, Tahmina Ahmed, Ravi Sandhu, and Ram Krishnan. A lattice interpretation of group-centric collaboration with expedient insiders. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on*, pages 200–209. IEEE, 2012.

[4] Khalid Zaman Bijon, Ram Krishnan, and Ravi Sandhu. Risk-aware RBAC sessions. In *Information Systems Security*, pages 59–74. Springer, 2012.

[5] Khalid Zaman Bijon, Ram Krishnan, and Ravi Sandhu. Towards an attribute based constraints specfication language. In *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*. IEEE, 2013.

[6] L Chen and J Crampton. Risk-aware role-based access control. In *7th International Workshop on Security and Trust Management*, 2011.

[7] Liang Chen, Luca Gasparini, and Timothy J Norman. XACML and risk-aware access control. *Resource*, 2(10):3–5, 2013.

[8] Pau-Chen Cheng, P. Rohatgi, C. Keser, P.A. Karger, G.M. Wagner, and A.S. Reninger. Fuzzy multi-level security: An experiment on quantified risk-adaptive access control. In *Security and Privacy, 2007.*, pages 222 –230, may 2007.

[9] David D. Clark and David R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *Proc. of the IEEE S&P*, 1987.

[10] David Ferraiolo, Janet Cugini, and Richard Kuhn. Role-based access control (RBAC): Features and motivations. In *1th Annual Computer Security Application Conference*, pages 241–248. IEEE, 1995.

[11] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Tran. Inf. Sys. Sec.*, 2001.

[12] Virgil D Gligor et al. On the formal definition of separation-of-duty policies and their composition. In *Security & Privacy*. IEEE, 1998.

[13] Trent Jaeger. On the increasing importance of constraints. In *fourth ACM workshop on Role-based access control*, pages 33–42. ACM, 1999.

[14] Xin Jin, Ram Krishnan, and Ravi Sandhu. A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. In *DBSec*, 2012.

[15] S. Kandala, R. Sandhu, and V. Bhamidipati. An attribute based framework for risk-adaptive access control models. In *Avail., Reliab. and Sec. (ARES)*, aug. 2011.

[16] Ian Molloy, Luke Dickens, Charles Morisset, Pau-Chen Cheng, Jorge Lobo, and Alessandra Russo. Risk-based security decisions under uncertainty. CODASPY '12, 2012.

[17] Michael J Nash and Keith R Poland. Some conundrums concerning separation of duty. In *Research in Security and Privacy*, pages 201–207. IEEE, 1990.

[18] Qun Ni, Elisa Bertino, and Jorge Lobo. Risk-based access control systems built on fuzzy inferences. ASIACCS '10, pages 250–260, New York, NY, USA, 2010. ACM.

[19] MITRE Corporation Jason Program Office. Horizontal integration: Broader access models for realizing information dominance. *Technical Report JSR-04-132, MITRE Corporation*, 2004.

[20] F. Salim, J. Reid, E. Dawson, and U. Dulleck. An approach to access control under uncertainty. In *Avail., Reliab. and Sec. (ARES)*, pages 1 –8, aug. 2011.

[21] Ravi Sandhu. Transaction control expressions for separation of duties. In *Proc. of the 4th ACSAC*, 1988.

[22] Ravi S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11), 1993.

[23] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-based access control models. *Computer*, 29(2):38 –47, feb 1996.

[24] Richard T Simon and Mary Ellen Zurko. Separation of duty in role-based environments. In *CSFW*, pages 183–194. IEEE, 1997.