RABAC : Role-Centric Attribute-Based Access Control

Xin Jin[1], Ravi Sandhu[1] and Ram Krishnan[2]

[1] Institute for Cyber Security & Department of Computer Science
[2] Institute for Cyber Security & Dept. of Elect. and Computer Engg.
University of Texas at San Antonio
xjin@cs.utsa.edu, {ravi.sandhu, ram.krishnan}@utsa.edu

Abstract. Role-based access control (RBAC) is a commercially dominant model, standardized by the National Institute of Standards and Technology (NIST). Although RBAC provides compelling benefits for security management it has several known deficiencies such as role explosion, wherein multiple closely related roles are required (e.g., attending-doctor role is separately defined for each patient). Numerous extensions to RBAC have been proposed to overcome these shortcomings. Recently NIST announced an initiative to unify and standardize these extensions by integrating roles with attributes, and identified three approaches: use attributes to dynamically assign users to roles, treat roles as just another attribute, and constrain the permissions of a role via attributes. The first two approaches have been previously studied. This paper presents a formal model for the third approach for the first time in the literature. We propose the novel role-centric attribute-based access control (RABAC) model which extends the NIST RBAC model with permission filtering policies. Unlike prior proposals addressing the role-explosion problem, RABAC does not fundamentally modify the role concept and integrates seamlessly with the NIST RBAC model. We also define an XACML profile for RABAC based on the existing XACML profile for RBAC.

Keywords: NIST-RBAC, attribute, XACML, access control

1 INTRODUCTION AND MOTIVATION

Role-based access control (RBAC) [12, 26] is a commercially successful and widely used access control model. Access permissions are assigned to roles and roles are assigned to users. Roles can be created, modified or disabled with evolving system requirements. Since the first formalizations [26] it has been recognized that traditional formulations of RBAC are inefficient in handling fine grained access control. RBAC can accommodate fine grained authorizations by dramatically increasing the number of distinct roles with slightly different sets of permissions. However, this solution incurs significant cost of correctly assigning permissions to large

numbers of roles. For instance, consider the familiar doctor-patient problem. In a hospital, a doctor is only allowed to view the record of his own patients. In the NIST RBAC model [12], a doctor role needs to be defined for each patient. Thus, the number of roles will be dramatically increased while they share mostly the same permissions. Anecdotal information indicates that in practice organizations work around these limitations in ad hoc ways. The research community has also proposed several ad hoc extensions to RBAC (see section 2).

Recently Kuhn et al [23] announced a NIST initiative to unify and standardize various RBAC extensions by integrating roles with attributes, thereby combining the benefits of RBAC and attribute-based access control (ABAC) to synergize the advantages of each. An informal review of ABAC concepts is provided in Karp et al [22]. Even with the relative immaturity of ABAC formal models the NIST approach is a promising avenue for injecting the benefits of ABAC into RBAC and vice versa. We note that there are access control proposals that go beyond attributes such as [14, 21]. However, we are motivated by the NIST ongoing initiative in extending RBAC through attributes, so models which go beyond ABAC are beyond our scope.

Kuhn et al [23] identify three alternatives for integrating attributes into RBAC as follows.

- **Dynamic Roles.** The first option uses user and context attributes to dynamically assign roles to users. It is similar to attribute-based user-role assignment [4]. This does help with automated user-role assignment to the myriad roles arising from role explosion, but does not address the corresponding role-permission assignment explosion (which has been considered in a recent model [19]). Context attributes have been studied in the literature [9–11].
- **Attribute Centric.** In this option roles are simply another attribute of users [7, 20]. There is no permission-role assignment relationship. This method largely discards the advantages of RBAC which are well demonstrated and mature [15].
- **Role Centric.** The general idea in the third option is that the maximum permissions available in each session are determined by the roles activated, which can be further reduced based upon attributes. However, Kuhn et al [23] do not elaborate on this option or provide details about this approach. Moreover, to our knowledge, there are no published formal models in the literature corresponding to this option.

Our central contribution is to develop a formal model for the role-centric approach for the first time. We propose the role-centric attribute-

based access control (RABAC) model which extends the NIST RBAC model with permission filtering policies. RABAC is a more convenient term otherwise identical to "RBAC-A, role-centric" in [23]. RABAC overcomes role explosion without fundamentally modifying RBAC. In particular, RABAC integrates seamlessly with the NIST RBAC model thereby offering a path for practical deployment. We also establish feasibility of implementation by providing an XACML profile for RABAC based on the existing standard XACML profile for the NIST RBAC model.

The rest of this paper is as follows. Section 2 discusses related work. Section 3 develops RABAC along with its formal definition and functional specifications. Section 4 defines the XACML profile for RABAC and presents implementation example. Section 5 concludes the paper.

2 RELATED WORK

The role explosion problem, wherein multiple closely related roles need to be defined to achieve fine-grained access control, has been recognized since the early days of RBAC, predating publication of the NIST RBAC model [12]. There has been considerable previous work on extending RBAC to avoid role explosion. Giuri [18] introduced the concepts of parameterized privileges and role templates to restrict a role to access a subset of objects based on the instantiated parameters. Other similar proposals include parameterized role [3, 17], conditional role [6], object sensitive role [13] and attributed role [28]. The above proposals change the fundamental process of role-permission assignment as permissions assigned to roles can only be determined when a role is instantiated or activated. There is a lack of accompanying administrative models for these extensions in such context and they do not fit into the existing administration models such as [25]. Compared with roles in the NIST RBAC model, these extensions increase the complexity of role mining and engineering, which is the costliest component of RBAC [16].

Numerous other extensions of RBAC have been proposed [15]. We briefly mention a few here. TrustBAC [8] incorporated the advantages of both RBAC and credential based access control models. But only user attribute trust level is considered. A family of extended RBAC models called role and organization based access control (ROBAC) models were proposed and formalized in [29]. However, it is not designed for access control within the same organization. Kumar et al [24] extended RBAC by introducing the notions of role context and context filters. However, context filters are applied only during the process of defining roles.

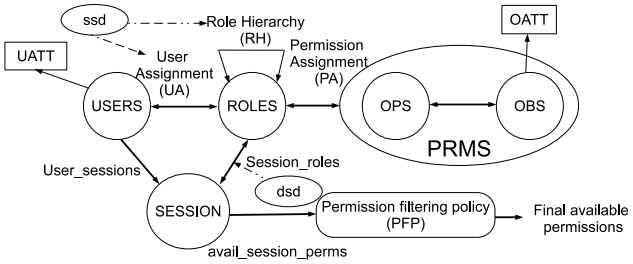

Fig. 1. *RABAC Model*

3 RABAC MODEL

In this section, we present the RABAC model as an extension of the NIST RBAC model. The model is first discussed informally and then formally defined in two parts similar to NIST RBAC model: reference model and functional specification.

3.1 Model Overview

The RABAC model is informally depicted in figure 1. It fully incorporates the NIST RBAC model and adds the following new elements: user attributes (UATT), object attributes (OATT) and permission filtering policy (PFP). We give a brief overview of these new elements below.

Attributes are functions which take certain entities and return values for defined properties of that entity (user or object).[3] Each user and object is associated with a finite set of attributes. Examples of user attributes are Department, Title and Specialization. Examples of object attributes are Type and Status. The range of each attribute is represented by a finite set of atomic values. For example, the range of Department is a set of all department names in the organization. Additionally we allow attributes to be set-valued. For instance, a set-valued Department attribute would allow a user to belong to multiple departments. Each attribute can either be atomic or set-valued from its declared range. Every attribute must be declared to be either atomic or set-valued.

The **Permission Filtering Policy (PFP)**, as suggested by its name, constrains the available set of permissions based on user and object attributes. It is depicted conceptually in figure 2. The avail_session_perm function, as defined by NIST RBAC model, gives the permission set

[3] More generally, attributes can be associated with other entities including sessions, environment, system, etc. User and object attributes suffice for purpose of RABAC.

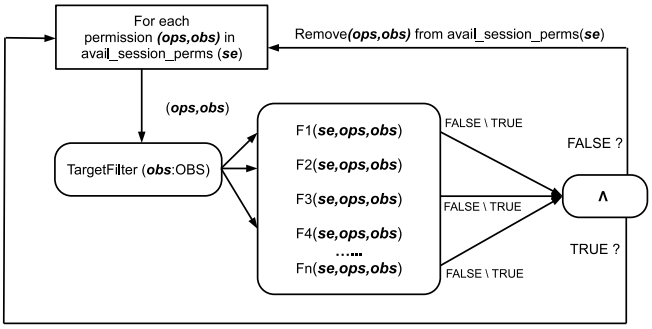

Fig. 2. *Permission Filtering Process*

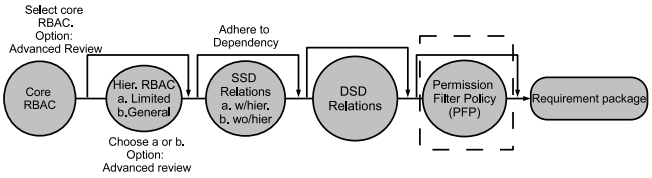

Fig. 3. *Methodology for Creating Functional Packages*

associated with the roles activated in a given session. In RABAC the avail_session_perm function represents the maximum permission set available in a session. These permission sets are further constrained by filtering policy. The security architect specifies a set of filter functions $\{F_1, F_2, F_3 \ldots F_n\}$ for this purpose. Each filter function is a boolean expression based on user and object attributes. The *TargetFilter* function maps each object to a subset of the filter functions. This mapping is based on the attributes of the object via attribute expressions called *conditions* which determine whether or not each filter function is applicable. The applicable filter functions are invoked one by one against each of the permissions in avail_session_perm. If any of the functions return FALSE, the permission is blocked and removed from the available permission set for this session. At the end of this process, we get the final available permission set. It should be noted that this description specifies the net result. Various optimizations can be used so long as the net result is as indicated.

With the newly defined PFP component, we are able to modify the logical approach for defining packages of functional components in the NIST RBAC model [12] as shown in figure 3. RABAC adds the dashed rectangle at the last stage. This indicates that PFP can be integrated into each of the RBAC model components independently.

Table 1. *NIST RBAC Sets and Functions used in RABAC*

- USERS, ROLES, OPS, and OBS (users, roles, operations, and objects);
- PRMS = $2^{(\text{OPS} \times \text{OBS})}$, the set of permissions;
- SESSIONS, the set of sessions;
- user_sessions(u: USERS) $\rightarrow 2^{\text{SESSIONS}}$, the mapping of user u onto a set of sessions;
- avail_session_perms(s: SESSIONS) $\rightarrow 2^{\text{PRMS}}$, the permissions available to a user in a session.
- PA $\subseteq$ PRMS $\times$ ROLES, a many-to-many mapping permission-to-role assignment;
- assigned_permissions(r: ROLES) $\rightarrow 2^{\text{PRMS}}$, the mapping of role r onto a set of permissions;

Table 2. *Additional Sets and Functions of RABAC*

- UATT and OATT represent finite sets of user and object attribute functions respectively.
- For each *att* in UATT $\cup$ OATT, Range(*att*) represents the attribute's range, a finite set of *atomic* values.
- attType: UATT $\cup$ OATT $\rightarrow$ {set, atomic}. Specifies attributes as set or atomic valued.
- Each attribute function maps elements in USERS and OBS to atomic or set values.

$$\forall ua \in \text{UATT}.\, ua : \text{USERS} \rightarrow \begin{cases} \text{Range(ua)} & \text{if attType}(ua) = \text{atomic} \\ 2^{\text{Range(ua)}} & \text{if attType}(ua) = \text{set} \end{cases}$$

$$\forall oa \in \text{OATT}.\, oa : \text{OBS} \rightarrow \begin{cases} \text{Range(oa)} & \text{if attType}(oa) = \text{atomic} \\ 2^{\text{Range(oa)}} & \text{if attType}(oa) = \text{set} \end{cases}$$

- FILTER = {F_1, F_2, F_3, ... F_n} is a finite set of boolean functions.
  For each $F_i \in$ FILTER. F_i: SESSIONS $\times$ OPS $\times$ OBS$\rightarrow$ {T, F}.

3.2 RABAC Reference Model

The basic sets and functions in the NIST RBAC model are shown in table 1. These sets and functions will also apply to RABAC. We define additional sets and functions for RABAC in table 2. UATT is a set of attribute functions for the existing users (i.e., USERS). Each attribute function in UATT maps a user to a specific value. This could be atomic or set valued as determined by the type of the attribute (as specified by *attType*). We specify similar sets and functions for objects. The notation used here for attributes is adapted from [20]. FILTER is a set of boolean functions defined by the security architects. The F_i are applied to sessions to constrain permissions associated with that session (discussed below).

Table 3. *Permission Filtering for RABAC*

1. Permission filtering policy.
Language LFilter is used to specify each filter function F_i(*se*:SESSIONS, *ops*:OPS, *obs*:OBS) in FILTER, where *se*, *ops* and *obs* are formal parameters.

2. Conditions.
For each $F_i \in$ FILTER there is a condition$_i$ which is a boolean expression specified using language LCondition.

3. TargetFilter is a function which maps each object to its applicable filter functions as a set. It is illustrated with the pseudo code shown as follows:

```
TargetFilter(obs:OBS)
{
    filter := {};
    condition1: filter := filter ∪ F1;
    condition2: filter := filter ∪ F2;
    ...
    conditionn: filter := filter ∪ Fn;
    return filter;
}
```

Where F_1, F_2 ... $F_n \in$ FILTER and *obs* is formal parameter.

Table 4. *Common Policy Language*

$\varphi ::= \varphi \wedge \varphi | \varphi \vee \varphi | (\varphi) | \neg\varphi | \exists$ x $\in$ set.$\varphi | \forall$ x $\in$ set.$\varphi |$ set setcompare set | atomic $\in$ set | atomic atomiccompare atomic

setcompare ::=$\subset$ | $\subseteq$ | $\nsubseteq$

atomiccompare ::= $<$ | $=$ | $\leq$

The permission filtering process is configured in three steps. As illustrated in the first part of table 3, security architects firstly define each filter function F_i in terms of user and object attributes by means of the language LFilter (defined below). Security architects also need to select a subset of the filter functions that apply to an object. This is done by the $TargetFilter$ function which requires specification of a boolean condition based on object attributes for each filter function F_i. As shown in the second part of table 3, there are n such conditions, one for each F_i. Each condition is defined using the language LCondition (defined below). For an object, the $TargetFilter$ function is illustrated in the third part of table 3. It evaluates each condition$_i$ based on the object's attributes to determine whether or not the filter function F_i is applicable. Thus it selects a subset of the filter functions applicable for any specific object.

The languages LFilter and LCondition are defined by adopting the common policy language (CPL) from [20] as shown in table 4. CPL de-

fines the logical structure but is not a complete language. It is required to specify the non-terminal symbols *set* and *atomic* to build complete instances of CPL. LFilter, the language used to specify each filter function F_i, is an instance of CPL where *set* and *atomic* are as follows.

set::= setua (sessionowner(*se*)) | setoa(*obs*) | ConsSet
atomic::= atomicua (sessionowner(*se*)) | atomicoa(*obs*) | ConsAtomic
setua ∈ {ua | ua ∈ UATT ∧ attType(ua) = set }
atomicua ∈ {ua | ua ∈ UATT ∧ attType(ua)= atomic }
setoa ∈ {oa | oa ∈ OATT ∧ attType(oa) = set }
atomicoa ∈ {oa | oa ∈ OATT ∧ attType(oa) = atomic }

ConsSet and *ConsAtomic* are constant sets and atomic values. *se* and *obs* are formal parameters of each filtering function. LFilter use the attributes of the involved user and object. Thereby, LFilter is able to constrain permissions dynamically based on various relationships between user and object attributes. We define the *sessionowner* function to return the owner of a session as follows.

sessionowner(*se*:SESSIONS) = u such that *se*∈user_sessions(u)

In the above definition, user_sessions(u: USERS) is already defined in the NIST RBAC model to return the sessions for a given user. LCondition, the language for specifying conditions, is an instance of CPL where *set* and *atomic* are as follows.

set::= setoa(*obs*) | ConsSet
atomic::= atomicoa(*obs*) | ConsAtomic

Each condition can only refer to the attributes of the object *obs* being accessed. *setoa* and *atomicoa* are the same as in LFilter.

3.3 Functional Specification

Our definitions of functional specifications for RABAC are based on those already defined in NIST RBAC model. The key extensions of this model focus on access decisions. Thus, we redefine the **CheckAccess** function from NIST RBAC and define a new function called **FilteredSessionPerm**. We specify these functions in table 5. Function **FilteredSessionPerm** returns final available permissions for each specific session. Function **CheckAccess** is used to check each request (ops, obs).

4 XACML PROFILE FOR RABAC

XACML [1] is a standard language for specifying attribute based access control policy. Because of its reputation, considerable work has been done

Table 5. *Functional Specifications*

Functions	**Updates**

FilteredSessionPerm
(se: SESSIONS)
	perset = avail_session_perm(se);
	For each (ops, obs) ∈ perset **do**
	if TargetFilter(obs) = {} break;
	For each function ∈ TargetFilter(obs) **do**
	if ¬function(se, ops, obs)
	perset = perset \ {(ops, obs)}; **break;**
	return perset;

CheckAccess (se: SESSIONS, ops: OPS, obs: OBS, result: BOOLEAN)	result = ((ops, obs)∈FilteredSessionPerm(se));

for XACML in implementing RBAC as well as its administration model [27]. XACML profile for RBAC [5] has been defined to guide implementing RBAC via XACML. For the purpose of demonstrating implementation feasibility of RABAC, we show that RABAC can be easily implemented in XACML. Specifically, we propose a XACML profile for RABAC based on that for RBAC. We then give a specific implementation example for this profile.

4.1 Proposed Profile

The standard XACML RBAC profile is limited to core and hierarchical RBAC. Our RABAC profile is similarly limited. We will only discuss those components of the standard XACML RBAC profile that need to be changed for RABAC. The RABAC profile is guided by the following.

- Permission Filtering Policies (PFP) are stored in a separate file from permission and role policy files for ease of administration.
- The result of role policy and PFP policy may be different. We need policy combination algorithm which gives deny if and only if PFP returns deny (Note that only positive permissions are defined for role policy in NIST RBAC model). Otherwise, the final result is permit.
- The result from different filter functions upon the same group of objects should be deny-override. Thereby if any one of them returns false, the final result for PFP will be false.

In light of these observations, we design an extension where the PDP (Policy Decision Point) loads one more kind of policy files for PFP components in addition to the role policy file as shown in figure 4. The

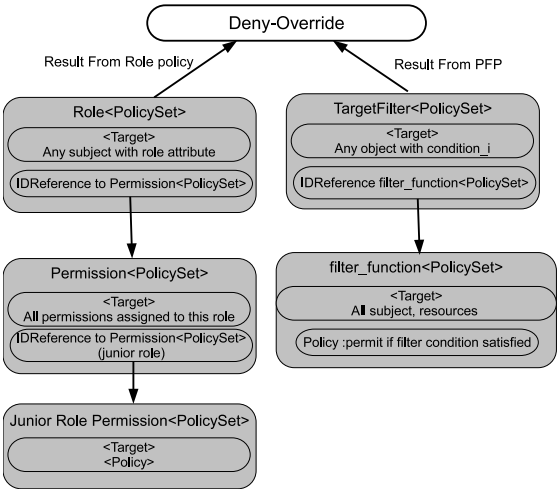

Fig. 4. *Part of Proposed XACML Profile for RABAC*

implementation for role and permission policy set remains the same. To implement PFP, a $TargetFilter$ <PolicySet> should be defined for each condition in the $TargetFilter$ function defined in the model. Conditions in $TargetFilter$ are implemented with *target* tag in XACML policy. Each $TargetFilter$ <PolicySet> contains policy references to actual $filter\_function$ <Policy>. Each reference represents a filter function defined in the model. The role policy and PFP policy may return different results. Since PFP is used only to reduce permissions there should not be PFPs that evaluate to permit. Thus, the role policy returns permit while the PFP policy may return two kinds of result $NotApplicable$ (no policy specified) or *Deny* (not allowed). We can determine that policy combining algorithm should be deny-override. The request is with the same format as that in XACML profile for RBAC except that the XACML subject is associated with multiple attributes in addition to *role*.

4.2 Example

We show the usage of our model in the doctor-patient problem in collaborative hospitals. The scenario is: *Doctor*, *Patient* and $VisitDoc$ are roles in each hospital. *Doctor* are allowed to read their *Patient*s' record at any time. $VisitDoc$ are only allowed to read authorized documents which are revealed for collaboration purpose with other hospitals. The request will only be approved during working hours made from any hospital certified devices. In addition, visiting doctors from other hospital

Table 6. RABAC Configuration for Doctor-Patient Problem

1. Basic sets and function
UATT={doctorof, uproj} OATT={type, recordof, oproj}
attType(doctorof)= attType(uproj) = attType(oproj)= set
attType(type)= attType(recordof)= atomic
Range(uproj) = Range(oproj)={proj1, proj2, proj3 ... }
Range(type)= {PatientRecord, AuthorizedDoc ...}
Range(doctorof)= Patient
Patient is all patients maintained by the hospital, Patient⊆U.
Range(recordof)= U
FILTER= {FPatient, FAuthorized}
2. Permission filtering policy

```
FPatient(se: SESSIONS, o: OBS, read)
{
   recordof(o)∈doctorof(sessionowner(se));
}
FAuthorized(se: SESSION, o: OBS, read)
{
   ( ∃ proj1 ∈ oproj(o). ∃ proj2 ∈ uproj(sessionowner(se)).proj1=proj2 )∧
   (8:00≤time(sessionowner(se)) ∧ time(sessionowner(se)) ≤ 17:00) ∧
   device(sessionowner(se)) ∈ { set of hospital certified devices }
}
TargetFilter(o: OBS)
{
   filter = {};
   case type(o) = PatientRecord: filter = filter ∪ FPatient;
   case type(o) = AuthorizedDoc: filter = filter ∪ FAuthorized;
   return filter;
}
```

are only allowed to view authorized documents pertaining to the projects they participate in. We present the configuration in RABAC in table 6. The elements are to be added to original RBAC solution. In traditional RBAC, a $VisitDoc$ role for each collaborative project should be defined. As new projects are created and accomplished, $VisitDoc$ roles have to be created and deleted. In addition, roles for each project are only different in the permissions regarding the specific projects. In our solution, a general $VisitDoc$ role is defined to be able to read all authorized projects documents. Then simple filtering policy can be specified in a straightforward manner (shown below). Thus, the role needed to be defined in traditional RBAC is the same as the number of projects while only one is needed in RABAC. Note that if the hospital requirements changes, e.g. a visit doc can read all authorized documents in his department, the role-permission and user-role relationship need to be changed in RBAC

while such change is not needed in RABAC. Rather we need to change the filtering policy in RABAC, which in this case would simply delete the corresponding filtering policy.

Following the above RABAC XACML profile we have implemented the fore-mentioned doctor-patient problem based on SUN's XACML implementation [2]. As per the standard RBAC XACML profile the role policy is straightforward. The TargetFilter <PolicySet> defines a policy for filtering access on *PatientRecord* and *AuthorizedDoc*. We take the *PatientRecord* as an example. The target is all patient records and there is a reference to the corresponding filter_function<Policy>. This policy defines a deny rule for reading patient records and the rule takes effect if resource (i.e., patient record) owner does not belong to the *doctorof* attribute value of a subject. One technical problem with this implementation is that *string-not-equal* is not natively embedded into XACML standard. Thus, we need to define this function which is straightforward and not explicitly shown here. An abbreviated portion of the XACML code for FPatiendRecord is shown below (role policy is the same as RBAC XACML profile and policy file for FAuthorized is similar).

XACML Code for PFP in Example

```
<Policy PolicyId="PFPPatiendRecord" RuleCombiningAlgId="deny-
    overrides">
  <Target></Any Subject>
    <Resources><!--Any PatientRecord--></Resources> </Any Action>
  </Target>
  <Rule RuleId="ReadRule" Effect="Deny">
    <Target><Any Subject Resource/>
      <Actions><Action><ActionMatch MatchId="string-equal">
        <AttributeValue DataType="string">read</AttributeValue>
          <ActionAttributeDesignator DataType="string"
            AttributeId="action:action-id"/>
          </ActionMatch></Action></Actions> </Target>
    <Condition FunctionId="string-not-equal">
        <Apply FunctionId="string-one-and-only">
        <SubjectAttributeDesignator DataType="string"
        AttributeId="doctorof"/></Apply>
     <Apply FunctionId="string-one-and-only">
      <ResourceAttributeDesignator DataType="string"
          AttributeId="owner"/></Apply>
   </Condition>
   </Rule>
</Policy>
```

5 CONCLUSION AND FUTURE WORK

In this paper, we proposed RABAC, a novel extension to the NIST RBAC model in an effort to address the role explosion problem of RBAC with-

out modifying significant components of RBAC model and retaining the static relationships between roles and permissions. It is the first model to integrate roles and attributes using the role centric approach identified by Kuhn et al [23]. RABAC integrates roles and attributes in a flexible and reliable manner. In particular, we define an independent component called the permission filtering policy (PFP) adding to the existing components of the NIST RBAC model. We also extend the functional specification of the NIST RBAC model and XACML profile for RBAC. Our solution essentially retains the administration convenience of RBAC while ensuring flexibility and scalability without role explosion.

There are several interesting directions for future work. Formal analysis of tradeoffs between roles and attributes may provide practically useful insights and results. The language CPL, which is used for specifying the filtering function as well as conditions in $TargetFilter$ functions, can be extended to leverage the power of XACML as these functions can be expressed through XACML policy files.

Acknowledgment

The authors are partially supported by grants from AFOSR MURI and the State of Texas Emerging Technology Fund.

References

1. OASIS, Extensible access control markup language (XACML), v2.0 (2005).
2. Sun's XACML implementation. Available at http: //sunxacml.sourceforge.net/index.html.
3. Ali E. Abdallah and Etienne J. Khayat. A Formal Model for Parameterized Role-Based Access Control. In *Formal Aspects in Security and Trust*, 2004.
4. MA Al-Kahtani and R. Sandhu. A model for attribute-based user-role assignment. *ACSAC*, 2002.
5. Anne Anderson. XACML profile for role based access control (RBAC). Technical Report Draft 1, OASIS, February 2004.
6. Yubin Bao, Jie Song, Daling Wang, Derong Shen, and Ge Yu. A Role and Context Based Access Control Model with UML. In *ICYCS*, 2008.
7. D.W. Chadwick, A. Otenko, and E. Ball. Implementing Role Based Access Controls Using X.509 Attribute Certificates. *IEEE Internet Computing*, 2003.
8. Sudip Chakraborty and Indrajit Ray. TrustBAC: integrating trust relationships into the RBAC model for access control in open systems. In *SACMAT*, 2006.
9. L. Cirio, I. F. Cruz, and R. Tamassia. A Role and Attribute based Access Control System Using Semantic Web Technologies. In *OTM Workshops*, 2007.
10. Michael J. Covington, Wende Long, Srividhya Srinivasan, Anind K. Dev, Mustaque Ahamad, and Gregory D. Abowd. Securing context-aware applications using environment roles. In *SACMAT*, 2001.

11. Michael J. Covington and Manoj R. Sastry. A Contextual Attribute-Based Access Control Model. In *OTM Workshops*, 2006.
12. David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Trans. on Infor and Sys Sec*, 2001.
13. Jeffrey Fischer, Daniel Marino, Rupak Majumdar, and Todd D. Millstein. Fine-grained access control with object-sensitive roles. In *ECOOP*, pages 173–194, 2009.
14. Philip W. L. Fong. Relationship-based access control: protection model and policy language. In *CODASPY*, 2011.
15. Ludwig Fuchs, Günther Pernul, and Ravi S. Sandhu. Roles in information security-A survey and classification of the research area. *Computers & Security*, 2011.
16. M. P. Gallagher, A.C. O'Connor, and B. Kropp. The economic impact of role-based access control. In *Planning report 02-1, NIST,*, March 2002.
17. Mei Ge and Sylvia L. Osborn. A design for parameterized roles. In *DBSec*, 2004.
18. Luigi Giuri and Pietro Iglio. Role templates for content-based access control. In *Proc. of the second ACM workshop on RBAC*. ACM, 1997.
19. Jingwei Huang, David Nicol, Rakesh Bobba, and Jun Ho Huh. A Framework Integrating Attribute-based Policies into RBAC. In *SACMAT*, 2012.
20. Xin Jin, Ram Krishnan, and Ravi Sandhu. A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. In *DBSec*, 2012.
21. Anas Abou El Kalam, Salem Benferhat, Alexandre Miege, Rania El Baida, Frederic Cuppens, Claire Saurel, Philippe Balbiani, Yves Deswarte, and Gilles Trouessin. Organization based access control. In *POLICY*, 2003.
22. A. H. Karp, H. Haury, and M. H. Davis. From ABAC to ZBAC: the evolution of access control models,. In *tech.report*, HP Labs, 2009.
23. D. Richard Kuhn, Edward J. Coyne, and Timothy R. Weil. Adding Attributes to Role-Based Access Control. *IEEE Computer*, 43(6):79–81, 2010.
24. Arun Kumar, Neeran Karnik, and Girish Chafle. Context sensitivity in role-based access control. *SIGOPS Oper. Syst. Rev.*, 36(3):53–66, 2002.
25. R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based administration of roles. *ACM Trans. on Info and Sys Sec*, 1999.
26. Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
27. M. Xu, D. Wijesekera, X. Zhang, and D. Cooray. Towards Session-Aware RBAC Administration and Enforcement with XACML. In *POLICY*, 2009.
28. Jianming Yong, Elisa Bertino, Mark Toleman, and Dave Roberts. Extended RBAC with role attributes. In *10th Pacific Asia Conf. on Info Sys*, 2006.
29. Zhixiong Zhang, Xinwen Zhang, and Ravi Sandhu. ROBAC: Scalable role and organization based access control models. In *IEEE TrustCol*, 2006.