A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC

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

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

Abstract. Recently, there has been considerable interest in attribute based access control (ABAC) to overcome the limitations of the dominant access control models (i.e, discretionary-DAC, mandatory-MAC and role based-RBAC) while unifying their advantages. Although some proposals for ABAC have been published, and even implemented and standardized, there is no consensus on precisely what is meant by ABAC or the required features of ABAC. There is no widely accepted ABAC model as there are for DAC, MAC and RBAC. This paper takes a step towards this end by constructing an ABAC model that has "just sufficient" features to be "easily and naturally" configured to do DAC, MAC and RBAC. For this purpose we understand DAC to mean owner-controlled access control lists, MAC to mean lattice-based access control with tranquility and RBAC to mean flat and hierarchical RBAC. Our central contribution is to take a first cut at establishing formal connections between the three successful classical models and desired ABAC models.

Keywords: Attribute, XACML, DAC, MAC, RBAC, ABAC

1 INTRODUCTION

Starting with Lampson's access matrix in the late 1960's, dozens of access control models have been proposed. Only three have achieved success in practice: discretionary access control (DAC) [24], mandatory access control (MAC, also known as lattice based access control or multilevel security) [22] and role-based access control (RBAC) [11, 23]. While DAC and MAC emerged in the early 1970's it took another quarter century for RBAC to develop robust foundations and flourish. RBAC emerged due to increasing practitioner dissatisfaction with the then dominant DAC and MAC paradigms, inspiring academic research on RBAC. Since then RBAC has become the dominant form of access control in practice.

Recently there has been growing practitioner concern with the limitations of RBAC, which has been met by researchers in two different ways. On one hand researchers have diligently and creatively extended RBAC in

numerous directions. Conversely there is growing appreciation that a more general model, specifically attribute-based access control (ABAC), could encompass the demonstrated benefits of DAC, MAC and RBAC while transcending their limitations. Identities, clearances, sensitivity, roles and other properties of users, subjects and objects can all be expressed as attributes. Languages for specifying permitted accesses based on the values and relationships among these attributes provide policy flexibility and customization. However, the proliferation and flexibility of policy configuration points in ABAC leads to greater difficulty in policy expression and comprehension relative to the simplicity of DAC, MAC and RBAC. It will require strong and comprehensive foundations for ABAC to flourish.

Intuitively, an attribute is a property expressed as a name:value pair associated with any entity in the system, including users, subjects and objects. Appropriate attributes can capture identities and access control lists (DAC), security labels, clearances and classifications (MAC) and roles (RBAC). As such ABAC supplements and subsumes rather than supplants these currently dominant models. Moreover any number of additional attributes such as location, time of day, strength of authentication, departmental affiliation, qualification, and frequent flyer status, can be brought into consideration within the same extensible framework of attributes. Thus the proliferation of RBAC extensions might be unified by adding appropriate attributes within a uniform framework, solving many of these shortcomings of core RBAC. At the same time we should recognize that ABAC with its flexibility may further confound the problem of role design and engineering. Attribute engineering is likely to be a more complex activity, and a price we may need to pay for added flexibility.

Much as RBAC concepts were around for decades before their formalization [13], nascent ABAC notions have been around for a while (see related work). The ABAC situation today is analogous to RBAC in its pre-1992 pre-RBAC and 1992-1996 early-RBAC periods [13]. Although considerable literature has been published, there is no agreement on what ABAC means. Fundamental questions such as components of core models lack authoritative answers, let alone a widely accepted ABAC model.

In this paper, we take a first step towards our eventual goal of developing an authoritative family of foundational models for attribute based access control. We believe this goal can be achieved only by means of incremental steps that advance our understanding. ABAC is a rich platform. Addressing it in its full scope from the beginning is infeasible. There are simply too mnay moving parts. A reasonable first step is to develop a formal ABAC model that is just sufficiently expressive to capture DAC,

MAC and RBAC. This provides us a well-defined scope while ensuring that the resulting model has practical relevance. There have been informal demonstrations, such as [8, 21], of the classical models using attributes. Our goal is to develop more complete and formal constructions.

The paper is organized as follows. We review previous work in section 2. In section 3, we characterize the three classical models from an ABAC perspective and informally identify the minimal features of an unifying ABAC model. In section 4, we give an overview of the ABAC_α model. In section 5, we present the formal definition of the model as well as functional specifications. In section 6, we show the configurations for DAC, MAC and RBAC in ABAC_α. Section 7 concludes the paper.

2 RELATED WORK

Extensions to RBAC by combining attributes and roles have been widely studied. [15] defines parameterized privileges to restrict access to a subset of objects. Similar literature such as parameterized role [3, 10, 14], object sensitive role [12] and attributed role [27] are also proposed. RB-RBAC model [4] use attributes to assist automatic user-role assignment.

Several attribute based access control systems and models have been proposed. The UCON model [21] focuses on usage control where authorizations are based on the attributes of the involved components. It is attribute-based but, rather than dealing with core ABAC concepts, it focuses on advanced access control features such as mutable attributes, continuous enforcement, obligations and conditions. UCON more or less assumes that an ABAC model is in place on top of which the UCON model is constructed. [21] sketches out instantiation of DAC, MAC and RBAC in UCON but the constructions are informal and not complete. Informal mappings of an ABAC system into DAC, MAC and RBAC are also described in [8]. Damiani et al [9] describe an informal framework for attribute based access control in open environments. Bonatti et al [6, 7] present a uniform structure to logically formulate and reason about both service access and information disclosure constraints according to related entity attributes. Similarly, [28–30] develop a service negotiation framework for requesters and providers to gradually expose their attributes. However, none of these investigates their connections with DAC, MAC and RBAC. Wang et al [26] proposes a framework that models an attribute-based access control system using logic programming with set constraints of a computable set theory. This work mainly focus on how set theory helps define the policy, rather than the model itself. Flexible access

control system [16, 5] can specify some features of attribute based access policies. Yuan and Tong [31] describe ABAC in the aspects of authorization architecture and policy formulation. This work focus on enforcement level rather than policy level of the model. Bo et al [19] mention that DAC, MAC and RBAC is configurable through ABAC. However, neither formal model nor details of the configurations are provided. Role-based trust management [20] is a flexible approach for access control in distributed systems where access control decisions are based on tracking chaining credentials. However, its core idea is extensions to role based access control. XACML [1] and SAML [2] are access control-related web services standards that both support attribute-based access control. These standard languages are designed without a formal ABAC model.

3 $ABAC_\alpha$: COVERING DAC, MAC AND RBAC

Our goal is to develop an ABAC model that has "just sufficient" features to be "easily and naturally" configured to do DAC, MAC and RBAC. We recognize these terms are qualitative, hence the quotation marks. For clarity of reference we designate this model as $ABAC_\alpha$ and understand ABAC to denote the larger concept. Our goal is to eventually develop a family of ABAC models, analogous to RBAC96 [23], which will become the de facto standard for defining, refining and evolving ABAC. The contributions of this paper are one step towards this goal.

We very much expect ABAC to include advanced features that go significantly beyond $ABAC_\alpha$, e.g., mutable attributes [21], environment attributes [31] and connection attributes [18]. At this point it is premature to consider whether $ABAC_\alpha$ might be the core ABAC model, an advanced model or a special case of some model in a prospective ABAC family. Which features belong in a core ABAC model, which belong in advanced models and which are outside the scope of ABAC are crucial questions that researchers must eventually resolve. However, for the moment, we deliberately limit our scope to developing $ABAC_\alpha$.

$ABAC_\alpha$ is motivated by the fact that the three classical models have been widely deployed and remain in active widespread use. The value of ABAC has been perceived in benefits it provides beyond DAC, MAC and RBAC, such as dynamic access control [25]. Nonetheless, it is of interest to develop $ABAC_\alpha$ that captures these three without incorporating "extraneous" features. We anticipate that $ABAC_\alpha$ will eventually fit somewhere within the yet-to-be-developed authoritative family of ABAC models.

For purpose of $ABAC_\alpha$ we understand DAC to mean owner-controlled access control lists [24], MAC to mean lattice-based access control with tranquility [22] (i.e.,subject and object label do not change)and RBAC to mean core or flat RBAC ($RBAC_0$), and hierarchical RBAC ($RBAC_1$) [11, 23]. Extensions beyond these interpretations of DAC, MAC and RBAC may or may not require extensions to $ABAC_\alpha$, comprehensive study of which is outside the scope of this paper.

Table 1. $ABAC_\alpha$ intrinsic requirements.

	Subject attribute values constrained by creating user?	Object attribute values constrained by creating subject?	Attribute range ordered?	Attribute functions return set value?	Object attributes modification?	Subject attribute modification by creating user?
DAC	YES	YES	NO	YES	YES	NO
MAC	YES	YES	YES	NO	NO	NO
$RBAC_0$	YES	NA	NO	YES	NA	YES
$RBAC_1$	YES	NA	YES	YES	NA	YES
$ABAC_\alpha$	YES	YES	YES	YES	YES	YES

The intrinsic features of $ABAC_\alpha$ that follow from the above interpretation of DAC, MAC and RBAC are highlighted in Table 1. This table recognizes three kinds of familiar entities: users, subjects (or sessions in RBAC) and objects. Each user, subject and object has attributes associated with it. The range of each attribute is either atomic valued or set valued, with atomic values partially ordered or unordered and set values ordered by subset. Let us consider each column in turn.

Column 1. In all cases subject attribute values are constrained by attributes of the creating user. In MAC, users can only create subjects whose clearance is dominated by that of the user. In RBAC, subjects can only be assigned roles assigned to or inherited by the creating user. In DAC, MAC and RBAC, the subject's creator is set to be the creating user. Interestingly this is the only column with YES values for all rows.

Column 2. For object attributes in MAC a subject can only create objects with the same or higher classification as the subject's clearance. In DAC there is no constraint on the access control list associated with a newly created object. It is up to the creator's discretion. However, we recognize that DAC has a constraint on newly created objects in that

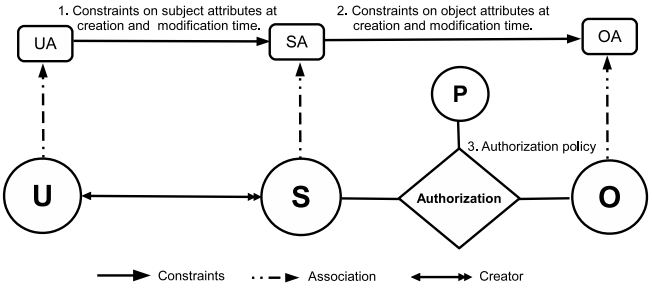

Fig. 1. *Unified ABAC model structure.*

root user usually has all access rights to every object and the owner can not forbid this. RBAC does not speak to object creation.
Column 3. In MAC clearances are values from a lattice of security labels. In RBAC_1 roles are partially ordered by permission inheritance. DAC and RBAC_0 do not require ordered attribute values.
Column 4. In MAC the clearance attribute is atomic valued as a single label from a lattice. In RBAC_0 and RBAC_1 attributes are sets of roles, and in DAC each access control list is a set of user identities.
Column 5. In DAC the user who created an object can modify its access control lists. MAC (with tranquility) does not permit modification of an object's classification. RBAC_0 and RBAC_1 do not speak to this issue.
Column 6. Modification of subject attributes by the creating user is explicitly permitted in RBAC_0 and RBAC_1 to allow dynamic activation and deactivation of roles. DAC and MAC do not require this feature.

Each column imposes requirements on ABAC_α so we have YES across the entire row. Table 1 is, of course, not a complete list of all required features to configure the classical models, but rather highlights the salient requirements that stem from each classical model.

4 ABAC_α Components

Based on the above analysis, we present a unified ABAC_α model informally in this section followed by its formalization in the next section. The structure of ABAC_α model is shown in Figure 1. The core components of this model are: users (U), subjects (S), objects (O), user attributes (UA), subject attributes (SA), object attributes (OA), permissions (P), authorization policies, and constraint checking policies for creating and modifying subject and object attributes.

An **attribute** is a function which takes an entity such as a user and returns a specific value from its range. An attribute range is given by a finite set of atomic values. An atomic valued attribute will return one value from the range, while a set valued attribute will return a subset of the range. Each **user** is associated with a finite set of **user attribute** functions whose values are assigned by security administrators (outside the scope of the model). These attributes represent the user properties, such as name, clearance, roles and gender. **Subjects** are created by users to perform some actions in the system. For the purpose of this paper, subjects can only be created by a user and are not allowed to create other subjects. The creating user is the only one who can terminate a subject. Each subject is associated with a finite set of **subject attribute** functions which require an initial value at creation time. Subject attributes are set by the creating user and are constrained by policies established by security architects (discussed later). For example, a subject attribute value may be inherited from a corresponding user attribute. This is shown in Figure 1 as an arrow from user attributes to subject attributes. **Objects** are resources that need to be protected. Objects are associated with a finite set of **object attribute** functions. Objects may be created by a subject on behalf of its user. At creation, the object's attribute values may be set by the user via the subject. The values may be constrained by the corresponding subject's attributes. For example, the new object may inherit values from corresponding subject attributes. In Figure 1, the arrow from subject attributes to object attributes indicates this relationship.

Constraints are functions which return true when conditions are satisfied and false otherwise. Security architects configure constraints via policy languages. Constraints can apply at subject and object creation time, and subsequently at subject and object attribute modification time.

Permissions are privileges that a user can hold on objects and exercise via a subject. Permissions enable access of a subject to an object in a particular mode, such as read or write. Permissions definition is dependent on specific systems built using this model.

Authorization policy. Authorization policies are two-valued boolean functions which are evaluated for each access decision. An authorization policy for a specific permission takes a subject, an object and returns true or false based on attribute values. More generally, access decision may be three-valued, possibly returning "don't know" in addition to true and false. This is appropriate in multi-policy systems. It suffices for our purpose to consider just two values. Security architects are able to specify different authorization policies using the language offered in this model.

Table 2. *Basic sets and functions of ABAC$_\alpha$*

U, S and O represent finite sets of *existing* users, subjects and objects respectively.

UA, SA and OA represent finite sets of user, subject and object attribute functions respectively. (Henceforth referred to as simply attributes.)

P represents a finite set of permissions.

For each *att* in UA $\cup$ SA $\cup$ OA, Range(*att*) represents the attribute's range, a finite set of *atomic* values.

SubCreator: $S \rightarrow U$. For each subject SubCreator gives its creator.

attType: UA $\cup$ SA $\cup$ OA $\rightarrow$ {set, atomic}. Specifies attributes as set or atomic valued.

Each attribute function maps elements in U, S and O to atomic or set values.

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

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

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

5 Formal ABAC$_\alpha$ Model

The basic sets and functions in ABAC$_\alpha$ are given in Table 2. U is the set of existing users and UA is a set of attribute function names for the users in U. Each attribute function in UA maps a user in U to a specific value. This could be atomic or set valued as determined by the type of the attribute function (attType). We specify similar sets and functions for subjects and objects. SubCreator is a distinguished attribute that maps each subject to the user who creates it (an alternate would be to treat this attribute as a function in SA). Finally, P is a set of permissions.

Policy Configuration Points. We define four policy configuration points as shown in Table 3. The first is for authorization policies (item 1 in table 3). The security architect specifies one authorization policy for each permission. The authorization function returns true or false based on attributes of the involved subject and object. The second configuration point is constraints for subject attribute assignment (item 2 in table 3). The third is constraints for object attributes assignment at the time of object creation (item 3 in table 3). The fourth is constraints for object attribute modification after the object has been created (item 4 in table 3). Note that we have not provided a configuration point for subject

Table 3. *Policy configuration points and languages of $ABAC_\alpha$*

1. Authorization policies.
For each p ∈ P, $\text{Authorization}_\text{p}$(s:S,o:O) returns true or false.
Language LAuthorization is used to define the above functions (one per permission), where s and o are formal parameters.

2. Subject attribute assignment constraints.
Language LConstrSub is used to specify ConstrSub(u:U,s:S,saset:SASET), where u, s and saset are formal parameters. The variable saset represents proposed attribute name and value pairs for each subject attribute. Thus SASET is a set defined as follows:

$$\text{SASET} = \bigcup_{\forall sa \in SA} \text{OneElement}(\text{SASET}_{sa})$$

$$\text{For each } sa \text{ in SA, } SASET_{sa} = \begin{cases} \{sa\} \times \text{Range(sa)} \text{ if attType}(sa) = \text{atomic} \\ \{sa\} \times 2^{\text{Range(sa)}} \text{ if attType}(sa) = \text{set} \end{cases}$$

We define OneElement to return a singleton subset from its input set.

3. Object attribute assignment constraints at object creation time.
Language LConstrObj is used to specify ConstrObj(s:S,o:O,oaset:OASET), where s, o and oaset are formal parameters. The variable oaset represents proposed attribute name and value pairs for each object attribute. Thus OASET is a set defined as follows:

$$\text{OASET} = \bigcup_{\forall oa \in OA} \text{OneElement}(OASET_{oa})$$

$$\text{For each } oa \text{ in OA}, OASET_{oa} = \begin{cases} \{oa\} \times \text{Range(oa)} \text{ if attType}(oa) = \text{atomic} \\ \{oa\} \times 2^{\text{Range(oa)}} \text{ if attType}(oa) = \text{set} \end{cases}$$

4. Object attribute modification constraints.
Language LConstrObjMod is used to specify ConstrObjMod(s:S,o:O,oaset:OASET), where s, o and oaset are formal parameters.

attribute modification after it has been created. For the stated purposes in this paper, the function SubCreator captures necessary information.

Policy Configuration Languages. Each policy configuration point is expressed using a specific language. The languages specify what information is available for the functions that configure the four points discussed above. For example, in LConstrSub function, only attributes from the user who wants to create the subject as well as the proposed subject attribute values are allowed. Since all specification languages share the same format of logical structure while differing only in the values they can use for comparison, we define a template called Common Policy Language (CPL). CPL is not a complete language unless terminals

Table 4. *Definition of* CPL

$\varphi ::= \varphi \wedge \varphi \mid \varphi \vee \varphi \mid (\varphi) \mid \neg \varphi \mid \exists$ x $\in$ set.$\varphi \mid \forall$ x $\in$ set. $\varphi \mid$ set setcompare set $\mid$ atomic $\in$ set $\mid$ atomic atomiccompare atomic
setcompare ::= $\subset \mid \subseteq \mid \nsubseteq$
atomiccompare ::= $< \mid = \mid \leq$

set and *atomic* are specified. It can be instantiated for specifying each configuration point. CPL is defined in table 4.

LAuthorization is a CPL instantiation for specifying authorization policies in which *set* and *atomic* are specified as follows:

set::= setsa(s) | setoa(o)
atomic::= atomicsa(s) | atomicoa(o)
setsa $\in$ {sa | sa $\in$ SA $\wedge$ attType(sa) = set }
setoa $\in$ {oa | oa $\in$ OA $\wedge$ attType(oa) = set }
atomicoa $\in$ {oa | oa $\in$ OA $\wedge$ attType(oa) = atomic }
atomicsa $\in$ {sa | sa $\in$ SA $\wedge$ attType(sa) = atomic }

LAuthorization allows one to specify policies based only on the value of involved subject and object. Parameters such as s and o in this and following languages are formal parameters as introduced in table 3.

LConstrSub is a CPL instantiation for specifying ConstrSub where:

set::= setua(u) | value
atomic::= atomicua(u) | value
setua $\in$ {ua | ua $\in$ UA $\wedge$ attType(ua) = set }
atomicua $\in$ {ua | ua $\in$ UA $\wedge$ attType(ua)= atomic }
value $\in$ {val | (sa, val) $\in$ saset $\wedge$ sa $\in$ SA}

This instance is different from above because in the constraint function for subject attributes, only the attribute of user who wants to create the subject and the proposed values for subject attributes are allowed.

LConstrObj is a CPL instantiation for specifying ConstrObj where:

set::= setsa(s) | value
atomic::= atomicsa(s) | value
setsa $\in$ {sa | sa $\in$ SA $\wedge$ attType(sa) = set }
atomicsa $\in$ {sa | sa $\in$ SA $\wedge$ attType(sa)= atomic }
value $\in$ {val | (oa, val) $\in$ oaset $\wedge$ oa $\in$ OA}

Here we use subject attributes instead of user attributes.

LConstrObjMod, used to specify ConstrObjMod, is the same as above except: set::= setsa(s) | setoa(o) | value and atomic::= atomicsa(s) | atomicoa(o) | value. Note that this language allows one to compare proposed

new attribute values with current attribute values of an object unlike LConstrObj.

Table 5. *Functional specification.*

Functions	**Conditions**	**Updates**
Administrative functions: Creation and maintenance of user and their attributes. UASET is a set containing name and value pairs for each user attribute. UASET $= \bigcup_{\forall ua \in UA}$ OneElement(UASET$_{ua}$) $\forall ua \in$ UA. $UASET_{ua} = \begin{cases} \{ua\} \times \text{Range(ua)} \text{ if attType}(ua) = \text{atomic} \\ \{ua\} \times 2^{\text{Range(ua)}} \text{ if attType}(ua) = \text{set} \end{cases}$		
AddUser (u:NAME,$uaset$:UASET)	$u \notin$U	U′=U∪{u} **forall** (ua,va)∈$uaset$ **do** ua(u)=va
DeleteUser(u:NAME) /*delete all u's subjects*/	$u \in$U	S′=S\{s|SubCreator(s)=u} U′=U\{u}
ModifyUserAtt (u:NAME,$uaset$:UASET) /*delete all u's subjects*/	$u \in$U	**forall** (ua,va)∈$uaset$ **do** ua(u)=va S′=S\{s|SubCreator(s)=u}
System functions: User level operations.		
CreateSubject (u, s:NAME,$saset$:SASET)	$u \in$U ∧ $s \notin$S∧ ConstrSub($u, s, saset$)	S′=S∪{s};SubCreator(s)=u **forall** (sa,va)∈$saset$ **do** sa(s)=va
DeleteSubject (u, s:NAME)	$s \in$S ∧ $u \in$U ∧ SubCreator(s)=u	S′=S\{s}
ModifySubjectAtt (u, s:NAME,$saset$:SASET)	$s \in$S ∧ $u \in$U ∧ SubCreator(s)=u ∧ ConstrSub($u, s, saset$)	**forall** (sa,va)∈$saset$ **do** sa(s)=va
CreateObject (s, o:NAME,$oaset$:OASET)	$s \in$S ∧ $o \notin$O ∧ ConstrObj($s, o, oaset$)	O′=O∪{o} **forall** (oa,va)∈oaset **do** oa(o)=va
ModifyObjectAtt (s, o:NAME,$oaset$:OASET)	$s \in$S ∧ $o \in$O ∧ ConstrObjMod($s, o, oaset$)	**forall** (oa,va)∈oaset **do** oa(o)=va
∀ $p \in$ **P. Authorization**$_p$; **ConstrSub; ConstrObj; ConstrObjMod**	/*Left to be specified by security architects*/	

Functional Specifications. The ABAC$_\alpha$ functional specification, as shown in Table 5, outlines the semantics of various functions that are required for creation and maintenance of the ABAC$_\alpha$ model components. Our intention here is to only provide a sample set of key functions due to space limitations. The first column lists all the function names as well as required parameters. The second column represents the conditions which

need to be satisfied before the updates, which are listed in the third column, can be executed. $NAME$ refers to set of all names for various entities in the system.

The first kind of functions are administrative in nature which are designed to be invoked only by security administrators. We do not specify the authorization conditions for administrative functions which are outside the scope of ABAC_α. They mainly deal with user and user attribute management. One important issue with the user management is that the subjects created by a user are forced to be terminated whenever user attributes are modified or the user is deleted. We understand there are various options here (discussion on this question is out of scope due to lack of space). The second kind of functions are system functions which can be invoked by subjects and users. By default, the first function parameter is the invoker of each function. For example, CreateSubject is invoked by user u and ModifyObjectAtt is invoked by subject s. The third kind of functions are authorization policies and subject and object attribute constraint functions which are left to be configured by security architects.

6 ABAC_α: CONFIGURING DAC, MAC AND RBAC

In this section, we show the capability of ABAC_α in configuring DAC, MAC and RBAC. For this illustration, we set P=$\{read, write\}$.

DAC (Table 6) Each object is associated with the same number of set-valued attributes as that of permissions and there is a one to one semantic mapping between them. An object attribute returns the list of users that hold the permission indicated by the object attribute name. Object attribute *createdby* is set to be the owner of this object.

MAC (Table 7) Each user is associated with an atomic-valued attribute *uclearance*. Each subject is also associated with an atomic-valued attribute *sclearance*. Each object is associated with an atomic-valued attribute *sensitivity*. Similar to MAC, the user and subject attributes represent their clearance in the system. The *sensitivity* attribute of the object represents the object's classification in MAC. The 3 attributes share the same range which is represented by a system maintained lattice L.

RBAC (Table 8) Each user and subject is associated with set-valued attributes *urole* and *srole* respectively. Each object is associated with the same number of set-valued attributes as that of permissions and there is a one to one semantic mapping between them. Each attribute returns the role that is assigned the permission on this specific object. For example,

Table 6. DAC (Owner-controlled access control lists) configuration

Basic sets and functions
UA={}, SA={}, OA={*reader*, *writer*, *createdby*}
P={*read*, *write*}
Range(*reader*)=Range(*writer*)=Range(*createdby*)=U
attType(*reader*)=attType(*writer*)=set
attType(*createdby*)=atomic
Thus, reader: O $\rightarrow 2^{U}$,writer: O $\rightarrow 2^{U}$, createdby: O $\rightarrow$ U
The function SubCreator is defined in Table 2.
Configuration points
1. Authorization policy
Authorization$_{read}$(*s*:S, *o*:O)$\equiv$SubCreator(*s*)$\in$*reader*(*o*)
Authorization$_{write}$(*s*:S, *o*:O)$\equiv$SubCreator(*s*)$\in$*writer*(*o*)
2. Constraint for subject attribute is not required
Note that SubCreator is implicitly captured in function CreateSubject in Table 5. Function ConstrSub(*u*:U, *s*:S, {}:SASET) is defined to return true.
3. Constraint for object attribute at creation time
ConstrObj(*s*:S, *o*:O, {(*reader*,val1), (*writer*,val2), (*createdby*,val3)}:OASET)$\equiv$ val3=SubCreator(*s*)
4. Constraint for object attribute at modification time
ConstrObjMod(*s*:S, *o*:O, {(*reader*,val1), (*writer*,val2), (*createdby*,val3)}:OASET)$\equiv$ *createdby*(*o*)=SubCreator(*s*)

rrole of object *obj* returns the role which is assigned the permission of reading *obj*. The ranges of all attributes are the same as that of a system defined set of role names R which are unordered for RBAC_0 and partially ordered for RBAC_1. Note that subjects model sessions in RBAC.

7 CONCLUSION AND FUTURE WORK

In this paper, we proposed a unified ABAC_α model and showed that it can be used to naturally configure the three classical models. We believe the insights gained in this paper will assist understanding the connections between desired ABAC model and widely-deployed classical models. In addition, we hope this work will inspire further research in formally designing foundational ABAC models.

Some extensions of classical models can also be accommodated. In MAC, it is useful to categorize subjects into different types as read_only and read_write for both security and availability. The rule governing their actions can be different in that read_only subjects are allowed to read all levels of objects. While read_write subjects' action is strictly regulated. Another example is in RBAC, certain level of automatic permission-role assignment can be achieved by interpreting permissions as accessing a

Table 7. MAC configuration

Basic sets and functions
UA={*uclearance*}, SA={*sclearance*}, OA={*sensitivity*}
P={*read*, *write*}
Range(*uclearance*)=Range(*sclearance*)=Range(*sensitivity*)=L
L is a lattice defined by system.
attType(*uclearance*)=attType(*sclearance*)=attType(*sensitivity*)= atomic
Thus, uclearance: U $\rightarrow$ L, sclearance: S $\rightarrow$ L, sensitivity: O $\rightarrow$ L.
Configuration points
1. Authorization policies
Authorization$_{read}$(s:S, o:O)$\equiv$*sensitivity*(o)$\leq$*sclearance*(s)
Liberal Star: Authorization$_{write}$(s:S, o:O)$\equiv$*sclearance*(s)$\leq$*sensitivity*(o)
Strict Star: Authorization$_{write}$(s:S, o:O)$\equiv$*sclearance*(s)=*sensitivity*(o)
2. ConstrSub(u:U, s:S, {(*sclearance*,value)}:SASET)$\equiv$value$\leq$*uclearance*(u)
3. ConstrObj(s:S, o:O, {(*sensitivity*, value)}:OASET)$\equiv$*sclearance*(s)$\leq$value
4. ConstrObjMod(s:S, o:O, {(*sensitivity*, value)}:OASET) returns false.

Table 8. RBAC configurations

RBAC$_0$ configuration
Basic sets and functions
UA={*urole*}, SA={*srole*}, OA={*rrole*, *wrole*}
P={*read*, *write*}
Range(*urole*)=Range(*srole*)=Range(*rrole*)=Range(*wrole*)=R
R is a set of atomic roles define by the system.
attType(*urole*)=attType(*srole*)=attType(*rrole*)=attType(*wrole*)=set
Thus, urole: U $\rightarrow$ 2^{R}, srole: S $\rightarrow$ 2^{R}, rrole: O $\rightarrow$ 2^{R}, wrole: O $\rightarrow$ 2^{R}
Configuration points
1. Authorization policy
Authorization$_{read}$(s:S, o:O)$\equiv\exists$r$\in$*srole*(s).$r\in$*rrole*(o)
Authorization$_{write}$(s:S, o:O)$\equiv\exists$r$\in$*srole*(s).$r\in$*wrole*(o) (same as above)
2. ConstrSub(u:U, s:S, {(*srole*,val1)}:SASET)$\equiv$val1$\subseteq$*urole*(u)
3. ConstrObj(s:S, o:O, {(rrole,val1),(wrole,val2)}:OASET) returns false.
4. ConstrObjMod(s:S, o:O, {(rrole,val1),(wrole,val2)}:OASET) returns false.

RBAC$_1$ configuration
Basic sets and functions
The basic sets and functions are the same as RBAC$_0$ except:
R is a partially ordered set defined by the system.
Configuration points
1. Authorization policy
Authorization$_{read}$(s:S, o:O)$\equiv\exists$r1$\in$*srole*(s). $\exists$r2$\in$*rrole*(o).r2$\leq$r1
Authorization$_{write}$(s:S, o:O)$\equiv\exists$r1$\in$*srole*(s). $\exists$r2$\in$*wrole*(o).r2$\leq$r1 (same as above)
2. ConstrSub(u:U, s:S, {(*srole*,val1)}:SASET)$\equiv\forall$r1$\in$val1.$\exists$r2$\in$*urole*(u).r1$\leq$r2
3. ConstrObj(s:S, o:O, {(rrole,val1),(wrole,val2)}:OASET) returns false.
4. ConstrObjMod(s:S, o:O, {(rrole,val1),(wrole,val2)}:OASET) returns false.

group of objects with the same attribute expression. Organization based access control model (OrBAC)[17] is another example of abstracting activities, objects and so on.

The first aspect of future work is to extend and consolidate the proposed model. Examples are to accommodate static/dynamic separation of duty in RBAC and subjects carrying additional attributes other than the corresponding users to reflect contextual information. Security properties and expressive power of this model are important questions for further theoretical analysis. On the other hand, useful instances of this model with various relationships between user, subject and object attributes can be developed for specific groups of application. For example, usable ABAC_α instance in organizations offer better guidance than general ABAC_α. In future work, we plan to develop XACML profiles for ABAC models as we develop them. By design XACML does not recognize user-subject mapping but assumes that subject attributes are correctly produced from user attributes prior to making access decisions. Modeling this process will therefore require extensions to XACML.

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. OASIS, Security assertion markup language (SAML), v2.0 (2005).
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. Mohammad A. Al-Kahtani and Ravi S. Sandhu. A model for attribute-based user-role assignment. In *ACSAC*, 2002.
5. Elisa Bertino, Barbara Catania, Elena Ferrari, and Paolo Perlasca. A logical framework for reasoning about access control models. SACMAT, 2001.
6. Piero A. Bonatti and P. Samarati. Regulating service access and information release on the web. In *ACM CCS*, 2000.
7. Piero A. Bonatti and P. Samarati. A uniform framework for regulating service access and information release on the web. *J. Comp. Secur.*, 2002.
8. David W. Chadwick, Alexander Otenko, and Edward Ball. Role-based access control with X.509 attribute certificates. *IEEE Internet Computing*, 2003.
9. E. Damiani, S.D.C. di Vimercati, and P. Samarati. New paradigms for access control in open environments. *Int. Sym on Sig. Proc. and Info Tech*, 2005.
10. Mark Evered. Supporting parameterised roles with object-based access control. In *HICSS*, 2003.
11. David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 2001.
12. Jeffrey Fischer, Daniel Marino, Rupak Majumdar, and Todd D. Millstein. Fine-grained access control with object-sensitive roles. In *ECOOP*, 2009.

13. L. Fuchs, G. Pernul, and R. Sandhu. Roles in information security: A survey and classification of the research area. *Comp. and Secur.*, 2011.
14. Mei Ge and Sylvia L. Osborn. A design for parameterized roles. In *DBSec*, 2004.
15. Luigi Giuri and Pietro Iglio. Role templates for content-based access control. In *ACM Workshop on RBAC*, 1997.
16. Sushil Jajodia, P. Samarati, Maria Luisa Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Trans. Database Syst.*, 2001.
17. Anas Abou El Kalam, Salem Benferhat, Alexandre Miège, Rania El Baida, Frédéric Cuppens, Claire Saurel, Philippe Balbiani, Yves Deswarte, and Gilles Trouessin. Organization based access control. In *POLICY*, 2003.
18. S. Kandala, R. Sandhu, and V. Bhamidipati. An attribute based framework for risk-adaptive access control models. In *ARES*, 2011.
19. Bo Lang, Ian T. Foster, Frank Siebenlist, Rachana Ananthakrishnan, and Timothy Freeman. A flexible attribute based access control method for grid computing. *J. Grid Comput.*, 2009.
20. Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust management framework. In *2002 IEEE S&P*.
21. Jaehong Park and Ravi Sandhu. The UCONabc usage control model. *ACM Trans. Inf. Syst. Secur.*, 2004.
22. Ravi S. Sandhu. Lattice-based access control models. *IEEE Computer*, 1993.
23. Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 1996.
24. Ravi S. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Com. Mag.*, 1994.
25. Christian Schläger, Manuel Sojer, Björn Muschall, and Günther Pernul. Attribute-based authentication and authorisation infrastructures for e-commerce providers. In *EC-Web*, 2006.
26. Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia. A logic-based framework for attribute based access control. In *In 2nd ACM Workshop on FMSE*, 2004.
27. Jianming Yong, Elisa Bertino, Mark Toleman, and Dave Roberts. Extended RBAC with role attributes. In *10th Pacific Asia Conf. on Info. Sys.*, 2006.
28. Ting Yu, Xiaosong Ma, and Marianne Winslett. Prunes: an efficient and complete strategy for automated trust negotiation over the internet. In *ACM CCS*, 2000.
29. Ting Yu, Marianne Winslett, and Kent E. Seamons. Interoperable strategies in automated trust negotiation. In *ACM CCS*, 2001.
30. Ting Yu, Marianne Winslett, and Kent E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Trans. Inf. Syst. Secur.*, 2003.
31. Eric Yuan and Jin Tong. Attributed based access control (ABAC) for web services. In *Intl. ICWS*, 2005.