HGABAC: Towards a Formal Model of Hierarchical Attribute-Based Access Control

Daniel Servos and Sylvia L. Osborn

Department of Computer Science
Western University
London, Ontario, Canada
`dservos5@uwo.ca` `sylvia@csd.uwo.ca`

Abstract. Attribute-based access control (ABAC) is a promising alternative to traditional models of access control (i.e. discretionary access control (DAC), mandatory access control (MAC) and role-based access control (RBAC)) that is drawing attention in both recent academic literature and industry application. However, formalization of a foundational model of ABAC and large scale adoption are still lacking. This paper seeks to aid in the transition by providing a formal model of hierarchical ABAC, called Hierarchical Group and Attribute-Based Access Control (or HGABAC), which includes attribute inheritance through user and object groups as well as environment, connection and administrative attributes. A formal specification and an attribute-based policy language are provided. Finally, several example configurations (which demonstrate the versatility of the model) are presented and evaluated.

1 Introduction

Until recently, access control research and real world access control implementations have largely fallen under one of the three traditional models of access control: discretionary access control (DAC)[11], mandatory access control (MAC)[1, 5] or role-based access control (RBAC)[6, 14]. In these models, access control decisions are largely based on the identity of the user. In DAC this often takes the form of an access control list (ACL) mapping users to permissions on an object, while MAC is based around a security lattice controlling the direction of information flow. In dynamic environments where information sharing between systems and users from different security domains is common, these identity-based access control models are inadequate. While RBAC provides a more generalized model than MAC or DAC [13], it also falls short in cases where users and their respective roles in the system are poorly defined beforehand. A secondary issue, common among these models, is the simplicity of the access control policies. In the case of RBAC, all access control policies must fit the form of "if a user is assigned a role X they are granted the set of permissions Y". However, this is insufficiently flexible for many real world scenarios. For example, a bank may only permit an employee with the role "teller" to access clients' accounts during set times of the day and week or limit their access to accounts based on a systemwide threat level. In both cases, the policy would be too complex to express in a traditional RBAC model.

To date, researchers have largely approached this problem by extending foundational RBAC models to compensate for inadequate flexibility required for their particular use case (e.g. [3, 10, 15, 4]). However, there has been a growing demand from both government and industry for a more general and dynamic model of access control, namely attribute-based access control (ABAC). Rather than basing access control decisions on a user's identity like the traditional methods, ABAC bases access control around the attributes of a user, the objects being accessed, the environment and a number of other attribute sources. Ideally, these are all properties of the elements already existing in the system and do not need to be manually entered by administration (e.g. many of the attributes about a document come from its existing metadata; author, title, etc.). Access policies can be created, limiting access to certain resources or objects, based on the result of a boolean statement comparing attributes, for example "user.age >= 18 OR object.owner == user.id". This allows for flexible enforcement of real world policies, while only requiring knowledge of some subset of attributes about a given user.

Despite the demand for and potential advantages of ABAC, little has been accomplished in the way of formalizing foundational models and large scale adoption is still in its early stages. The work detailed in this paper seeks to provide a formalized hierarchical model of ABAC, entitled Hierarchical Group and Attribute-Based Access Control (or HGABAC), which introduces a group based hierarchical representation of object and user attributes that is lacking in current models. HGABAC is intended to be a starting point that is detailed enough for real world use but generic enough to emulate traditional models of access control.

The rest of this paper is organized as follows. Section 2 reviews the existing work related to attribute-based access control models and current efforts towards standardization. Section 3 outlines our proposed model of ABAC, HGABAC, and provides a formal specification, as well as details of the policy language used and the group hierarchy. Section 4 gives an example use case and evaluates the solution HGABAC provides. Section 5 demonstrates how HGABAC can be configured to emulate DAC, MAC and RBAC access control policies. Finally, Section 6 details our conclusions and plans for future work.

2 Related Work

One of the most frequently referenced works in the ABAC literature is the eXtensible Access Control Markup Language (XACML) standard [7]. XACML is an XML-based access control policy language that is notable for its support of attribute-based policies and use in multiple access control products. While XACML supports attribute-base access control concepts and hierarchical resources, it intently lacks any kind of formal model (simply being a policy language) and instead relies on the implementing system to apply an underlying model of access control. Another related but distinct research area from ABAC is attribute-based encryption (ABE), where objects are encrypted based on attribute related access policies. In ciphertext-policy (CP-ABE) style ABE an attribute-based policy is used to encrypt an object and user's keys consist of a

Table 1. Comparison of notable models of attribute-based access control.

	Logic-based Framework for ABAC [18]	**ABACα** [8]	**ABAC for Web Services** [19]	**WS-ABAC** [17]	**ABMAC** [12]	**HGABAC**
Hierarchical	Hierarchical attributes, no user groups					✓
Object Attributes		✓	✓	✓	✓	✓
User Attributes	✓	✓	✓	✓	✓	✓
Environment Attributes			✓	✓	✓	✓
Connection Attributes					Shown in example but not model	✓
Administrative Attributes						✓
General Model	✓	✓	For web services	For web services	For grid computing	✓
Formal Model	Only models policies and evaluation	✓	Simplistic	Simplistic	✓	✓
Can Model DAC, MAC, and RBAC	Not demonstrated	✓	Not demonstrated	Not demonstrated	Not demonstrated	✓

set of attributes relating to that user[2, 16]. While ABE, much like XACML, lacks any kind of formal ABAC model and has rather simplified access policies, it does provide an interesting means of enforcing ABAC policies outside of the security domain in which they originate.

Various works have attempted to informally describe ABAC or have taken the first steps towards formalization. The most notable of these are summarized and compared to our model in Table 1. Yuan and Tong[19] describe the ABAC model in terms of authorization architecture and policy engineering and give an informal comparison between ABAC and traditional role-based models. Shen and Hong[17] present WS-ABAC, an ABAC model designed for web services based around XACML. However, the model presented in this work is limited and mostly describes an architecture to use XACML and attribute-based policies to provide authentication for web services. Lang et al.'s ABMAC model[12] aims to bring ABAC like access control to grid computing. While this model is based on attribute-based policy decisions, it has several key differences in the policy description and policy evaluation methods. This work is of note as it mentions ABAC's ability to represent the traditional models using a single policy language.

Wang et al.[18] propose a logic-based framework for ABAC based on logic programming where policies are specified as "stratified constraint flounder-free logic programs that admit primitive recursion". While their framework introduces hierarchical attributes (something lacking from other models), it is largely focussed on the representation, consistency and performance of attribute-based policies and their evaluation over providing a workable model of ABAC. Several critical components required for a usable model are absent, including object attributes (the only attributes considered are user attributes) and they omit any kind of formalization of ABAC aspects outside of policies and their evaluation (e.g. there is no mention of objects and only access control on services/operations is considered).

Lastly, and most promising is the work by Jin et al.[8] towards a generalized and formalized model of ABAC with constraints for the traditional models,

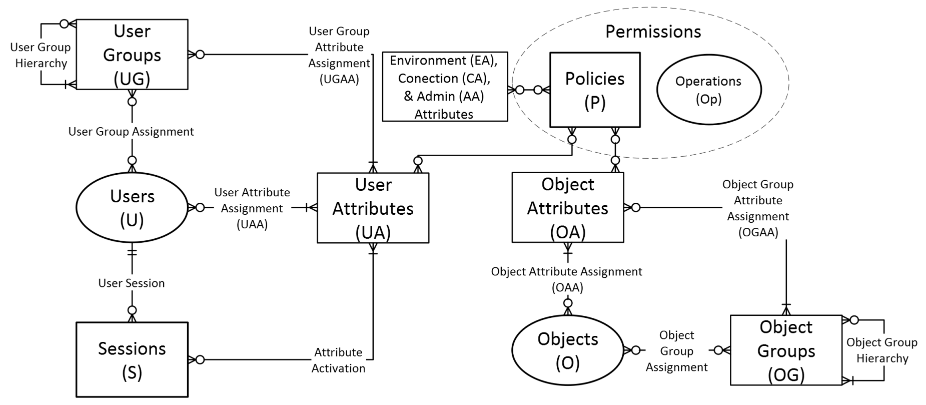


Fig. 1. HGABAC components and relations using Crow's Foot Notation to denote cardinality of relationships. Primitive components are shown in ovals.

which they call $ABAC_\alpha$. Their model provides a first step "to develop a formal ABAC model that is just sufficiently expressive to capture DAC, MAC and RBAC" which allows configuration of constraints on attributes at creation and modification time as well as policies. While this work provides a sufficient basis for new models of ABAC, it (intentionally) lacks components that would be necessary for a real world implementation, such as attribute and object hierarchies, a simplistic policy language and environment attributes.

Our model, HGABAC, is distinct from other models in several regards. Most notably, the graph based user group hierarchy provides several new interesting means of representing the traditional models in an ABAC framework (allowing the hierarchy to model MAC and RBAC in an intuitive way, rather than a partially ordered set as is done in $ABAC_\alpha$[8]). These hierarchical representations of MAC and RBAC are demonstrated in Section 5. The object group hierarchy allows for objects to be categorized into collections of similar types of objects (e.g. a collection of only health care record objects) and have common attributes applied to all members of the group. This reduces the amount of manual intervention required to tag similar objects with a common attribute and value pairs and reduces the number of object attribute assignments required (as evaluated in Section 4.2). Additionally, several efforts are made to create a model more suited to real world application without losing descriptive power or flexibility in terms of policies that may be enforced; a fully specified and intuitive policy language is presented (in Section 3.2) loosely based on C style boolean statements and a more rich selection of attribute sources is allowed. Attributes based on the user's current connection to the system and administrative attributes are supported which are lacking in other models. Finally, HGABAC uses a strongly typed system to represent attribute values (i.e. an attribute must have a predefined data type, e.g. integer, floating point, set, etc.). This helps enforce consistency in policies and attribute value assignments as well as helping to prevent any possible ambiguity in policies (e.g. preventing any type mismatches).

3 HGABAC Model

3.1 Formal Model

Basic Elements and Definitions: We define the base elements of the HGABAC model, as shown in Figure 1, as follows:

- **Users (U):** set of current human and non-human entities that may request access on system resources through sessions.
- **Objects (O):** finite set of system resources (files, database records, devices, etc.) for which access should be limited.
- **Operations (Op):** finite set of all operations provided by the system that may be applied to an object (e.g. read, write, create, delete, update etc.).
- **Policies (P):** set of all current policy strings following the format of our policy language defined in Section 3.2.
- **Sessions (S):** set of all user sessions, such that each element, s is a tuple of the form $s = (u \in U, a \subseteq \mathit{effective}(u \in U), \mathit{con\_atts})$ where u is the user who activated the session, $\mathit{con\_atts}$ is the set of connection attributes for the session such that $\forall c \in \mathit{con\_atts} : c = (\mathit{name}, \mathit{values})$ and a is the subset of the user's effective attributes they wish to activate for the given session ($\mathit{effective}(u)$ is the set of all attributes a user is assigned either directly through the UAA relation or indirectly through group membership). Policies are evaluated on the basis of the activated attributes in a user's session rather than the total set of the user's assigned and inherited attributes.
- **Permissions:** pairing of a policy string and an operation, such that $\mathit{perm} = (p \in P, \mathit{op} \in \mathit{Op})$. Access to perform an operation, op, on a given object is only allowed if there exists a permission that contains a policy, p, that is satisfied by a given set of attributes corresponding to the requesting user's session, object being accessed and the current state of the connection, environment and administrative attributes in the system. For example, a policy paired with a read operation, "$\mathit{user.id} = \mathit{object.author}$", would allow read access to all objects for which the user is also the author.

Attributes: HGABAC defines attributes to be (name, value, type) triples where the name is a unique identifier and value is an unordered set of atomic values of a given type or the *null* set. Type restricts the data type of the atomic values (e.g. string, integer, boolean, etc.) to a system defined data type. Attributes represent some descriptive characteristic of the entity to which they are assigned. For example, a user might have attributes describing their name, age, employee id, etc., while an object might have attributes describing its author, owner, file type, etc. The set of all attributes (TA) is divided into five subsets based on their origin and to which entity or object they may be applied:

- **User Attributes (UA):** the set of attribute name, type pairs that may be applied to users such that $\forall a \in \mathit{UA} : a = (\mathit{name}, \mathit{type})$ and each element of UA has a globally unique name (i.e. there cannot be two elements with the same name but different types). Note that value is left out of the definition of UA as user attributes are given a set of values when assigned to users directly or to groups (in the UAA and UGAA relations).

– **Object Attributes (OA):** the set of attribute name, type pairs that may be applied to objects such that $\forall a \in OA : a = (name, type)$ and each element of OA has a globally unique name (i.e. there cannot be two elements with the same name but different types). As with UA, value is left out of the definition of OA as object attributes are given a set of values when assigned to objects directly or to groups (in the OAA and OGAA relations).
– **Environment Attributes (EA):** the set of attribute (name, value, type) triples that represent the current state of the system's environment (e.g. the current time, number of active users, etc.) such that $\forall a \in EA : a = (name, value, type)$ and each element of EA has a globally unique name (i.e. there cannot be two elements with the same name but different types or values). What properties of a system's environment are available as environment attributes is left to the implementation.
– **Connection Attributes (CA):** the set of attribute name, type pairs that correspond to attributes derived from and available for each connection to the system such that $\forall a \in CA : a = (name, type)$ and each element of CA has a globally unique name (i.e. there cannot be two elements with the same name but different types). What properties of the connection are available as connection attributes is left as a implementation decision; however, at a minimum some kind of unique session id should be included.
– **Administrative Attributes (AA):** the set of attribute (name, value, type) triples that are defined by administrators (including automated administrative tasks and programs) that rarely change and apply to all policies which reference them such that $\forall a \in AA : a = (name, value, type)$ and each element of AA has a globally unique name. What administrative attributes are available will change at runtime based on both the implementation and actions of administrators.
– **Total Attributes (TA):** set of all attributes that exist in a given system such that $TA = UA \cup OA \cup CA \cup EA \cup AA$.

Groups: Groups and their hierarchies both simplify administration tasks, allowing attributes to be assigned to groups of users or objects at once rather than directly, and allow for more intuitive and expressive configuration possibilities than allowed in current ABAC models (including in the task of emulating the traditional models as shown in Section 5). Section 3.1 details the group hierarchy, while user and object group definitions and membership are defined below:

– **User Groups (UG):** set of all current user groups, where each element is comprised of a tuple, g, such that $g = (name, u \subseteq U, p \subseteq UG)$ where *name* is a globally unique identifier, u is the set of members of the group, and p is the set of the group's parents in the user group graph.
– **Object Groups (OG):** set of all current object groups, where each element is comprised of a tuple, g, such that $g = (name, o \subseteq O, p \subseteq OG)$ where *name* is a globally unique identifier, o is the set of members of the group, and p is the set of the group's parents in the object group graph.

Relations: We define the following relations between the base elements, groups and attributes:

- **Direct User Attribute Assignment (UAA):**
  user attribute assignment relation containing user, attribute name, value triples such that:
  $$\forall uaa \in UAA : uaa = (u \in U, att\_name, values)$$
  where $att\_name \in \{name \mid (name, type) \in UA\}$ and $values$ is some set of elements such that each element of $values$ is of the same data type ($type$) and $(att\_name, type) \in UA$. There may exist only one tuple in UAA for every user, att_name pair.

- **Direct Object Attribute Assignment (OAA):**
  object attribute assignment relation containing object, attribute name, value triples such that:
  $$\forall oaa \in OAA : oaa = (o \in O, att\_name, values)$$
  where $att\_name \in \{name \mid (name, type) \in OA\}$ and $values$ is some set of elements such that each element of $values$ is of the same data type ($type$) and $(att\_name, type) \in OA$. There may exist only one tuple in OAA for every object, att_name pair.

- **User Group Attribute Assignment (UGAA):**
  user group attribute assignment relation containing user group name, attribute name, value triples such that:
  $$\forall ugaa \in UGAA : ugaa = (group\_name, att\_name, values)$$
  where $group\_name \in \{name \mid (name, u, p) \in UG\}$ and $att\_name \in \{name \mid (name, type) \in UA\}$. $values$ is some set of elements such that each element of $values$ is of the same data type ($type$) and $(att\_name, type) \in UA$. There may exist only one tuple in UGAA for every group_name, att_name pair.

- **Object Group Attribute Assignment (OGAA):**
  object group attribute assignment relation containing object group name, attribute name, value triples such that:
  $$\forall ogaa \in OGAA : ogaa = (group\_name, att\_name, values)$$
  where $group\_name \in \{name \mid (name, u, p) \in OG\}$ and $att\_name \in \{name \mid (name, type) \in OA\}$. $values$ is some set of elements such that each element of $values$ is of the same data type ($type$) and $(att\_name, type) \in OA$. There may exist only one tuple in OGAA for every group_name, att_name pair.

Mappings: The following are the most important formal functions in the HGABAC model.

- **direct:** Mapping of a user, object, or group to the attribute name, value pairs directly assigned to it in the UAA, OAA, UGAA or OGAA relation (i.e. not including inherited attributes or attributes from group membership). *direct* is defined as:

$$direct(x) = \begin{cases} \{(n,v) \mid (x,n,v) \in UAA\}, & \text{if } x \in U \\ \{(n,v) \mid (x,n,v) \in OAA\}, & \text{if } x \in O \\ \{(n,v) \mid (name(x),n,v) \in UGAA\}, & \text{if } x \in UG \\ \{(n,v) \mid (name(x),n,v) \in OGAA\}, & \text{if } x \in OG \end{cases}$$

where $name(x)$ is the name of the given group, n is an attribute name, v is a set of valid values for that attribute and $x \in U \cup O \cup UG \cup OG$.

– **consolidate:** Mapping of a set of attribute name, value pairs which may contain multiple instances of the same name to a set of attribute name, value pairs where each name occurs only once. Value sets are unioned together for pairs with the same attribute name. *consolidate* is defined as:

$$consolidate(x) = \{(n, v_1 \cup v_2) \mid (n, v_1) \in x \wedge (n, v_2) \in x\}$$

where x is sets of attribute name, value pairs, n is an attribute name, v_1 and v_2 are sets of values.

– **member:** Mapping of a User or Object to the set of groups for which they are a member. *member* is defined as:

$$member(x) = \begin{cases} \{(n,u,p) \mid (n,u,p) \in UG \wedge x \in u\}, & \text{if } x \in U \\ \{(n,o,p) \mid (n,o,p) \in OG \wedge x \in o\}, & \text{if } x \in O \end{cases}$$

where n is the name of a group, u is a subset of U, o is a subset of O, p is a subset of UG or OG and $x \in U \cup O$.

– **inherited:** Mapping of a user, object or group to its set of inherited attributes (i.e. the set of attributes assigned indirectly through the group hierarchy or group membership). *inherited* is defined as:

$$inherited(x) = consolidate(\\ \begin{cases} \{(n,v) \mid g \in member(x) \wedge (n,v) \in consolidate(direct(g) \cup inherited(g))\}, & \text{if } x \in U \cup O \\ \{(n,v) \mid g \in parents(x) \wedge (n,v) \in consolidate(direct(g) \cup inherited(g))\}, & \text{if } x \in UG \cup OG \\ \emptyset, & \text{if } name(x) = min\_group \end{cases} \\ )$$

where $parents(x)$ is the set of parents for the given group, n is an attribute name, v is a set of valid values for n and $x \in O \cup U \cup UG \cup OG$.

– **effective:** Mapping of a user, object, or group to their effective attributes (i.e. all attributes inherited or directly assigned). *effective* is defined as:

$$effective(x) = consolidate(direct(x) \cup inherited(x))$$

where x is a user, object or group (i.e. $x \in U \cup O \cup OG \cup UG$).

– **name:** Mapping of a group or attribute to its assigned name. *name* is defined as:

$$name(x) = x_{(1)}$$

that is, the name is the first element of the tuple in both the case of groups and attributes, and $x \in OG \cup UG \cup TA$.

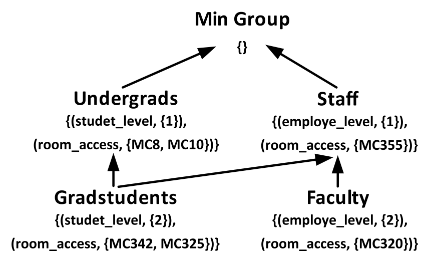


Fig. 2. Example user group hierarchy represented as a graph. The large bold text denotes the group's name, beneath which the set of directly assigned attributes is shown.

– **parents:** Mapping of a group to its set of parents. *parents* is defined as:

$$parents(x) = x_{(3)}$$

that is, the set of a groups where parents is the third element of the tuple in both the case of user and object groups, and $x \in OG \cup UG$.

– **authorized:** $P, S, O \rightarrow \{true, false, undef\}$
Function which determines if a user session passes the given policy given the current value of the environment and administrative attributes for a given object, where P is the set of all policies, S is the set of all sessions and O is the set of all objects. *true* and *false* are returned as expected based on the evaluation of the boolean policy rule; *undef* is returned if the policy cannot be evaluated (e.g. an object or user attribute referred to in the policy is not present in the attribute sets or incompatible types are compared).

Group Graph: HGABAC represents the group hierarchy as a directed acyclic graph with each group a vertex and each edge a parent/child relation between the groups such that the edge is directed to the parent. Additionally, all paths in the graph must eventually end at a special *min_group* that has no parents and no assigned attributes. A group, g, can only have *min_group* as a parent if it has one and only one parent such that effective(g) = direct(g) and inherited(g) = $\emptyset$. The parent/child relation between any two related groups is defined such that group c is a child of group p iff:

$$\forall (n, v_1) \in \text{effective(p)}:$$
$$\exists! a \in \text{effective(c)}: a = (n, v_2) \text{ and } v_1 \subseteq v_2$$

A child group must have one attribute for each effective attribute assigned to the parent group, such that the attribute has the same name and the parent's attribute's value is a subset of the child's attribute's value. Thus, the effective attributes for a group, g, are calculated as:

$$effective(g) = consolidate(direct(g) \cup inherited(g))$$

Users' and objects' effective attributes are calculated in a similar way, consolidating the values of directly assigned and inherited attributes.

An example user group hierarchy is shown in Figure 2. In this example the set of effective attributes of groups *Undergrad* and *Staff* are the same as their set of direct attributes as they both inherit from *min_group*. The group *Faculty* inherits the attributes (*employe_level*, {1}) and (*room_access*, {MC355}) from the group *Staff* such that the effective attributes of *Faculty* will be (*employe_level*, {2, 1}) and (*room_access*, {MC320, MC355}). Similarly, the group *Gradstudents* inherits attributes from both the groups *Undergrads* and *Staff* such that the set of effective attributes for *Gradstudents* is {(*employe_level*, {1}), (*student_level*, {1, 2}), (*room_access*, {MC325, MC342, MC8, MC10, MC355})}.

The object group hierarchy has the same properties as the user group hierarchy (being a directed acyclic graph, etc.), and is set up in a similar way with a *min_group* place holder being the ancestor of all object groups. In implementations, of HGABAC it is likely that both the user and object group graphs could be consolidated into a single graph with the same *min_group* and treated similarly (e.g. with the same functions/operations) so long as constraints are enforced so no object group may inherit from a user group and no object group may have a user attribute assigned (and vice versa).

3.2 Policy Language

In HGABAC access control decisions are based on a boolean rule based policy language comparing attributes and constants. The result of logical operations (AND, OR, NOT) on ternary values (TRUE, FALSE, UNDEF) are determined based on the AND, OR and NOT truth tables from Kleene K3 logic[9]. A policy evaluated to UNDEF is equivalent to FALSE in terms of access control decisions (i.e. access is denied). Comparison operations (<, >, etc.) result in TRUE or FALSE as expected when value types are comparable (e.g. $1 < 2$ results in TRUE) and UNDEF when incomparable (e.g. *"Pizza"* > 3.1415). The following definition of the policy language is given using ABNF syntax:

```
policy     =  exp [ bool_op policy ]
           /  ( policy )
exp        =  var op var
           /  [ "NOT" ] bool_var
           /  [ "NOT" ] "(" policy ")"
var        =  const / att_name
bool_var   =  boolean / att_name
op         =  ">" / "<" / "=" / ">=" / "<=" / "! =" / "IN" / "SUBSET"
bool_op    =  "AND" / "OR"
att_name   =  user_att_name / object_att_name / env_att_name / admin_att_name
           /  connect_att_name
user_att_name      =  "user." id
object_att_name    =  "object." id
env_att_name       =  "env." id
admin_att_name     =  "admin." id
connect_att_name   =  "connect." id
atomic     =  int / float / string / "NULL"
const      =  atomic / set
boolean    =  "TRUE" / "FALSE" / "UNDEF"
set        =  "{" "}" / "{" setval "}"
setval     =  atomic / atomic "," setval
id         =  +(ALPHA / DIGIT / "_")
int        =  ["-"] ( 1-9 ) *( DIGIT ) / "0"
float      =  int "." +( DIGIT )
string     =  DQUOTE *( %x20-21 / %x23-7E ) DQUOTE
```

user_att_name and *object_att_name* correspond to attribute names in *UA* and *OA* respectively, while *env_att_name*, *admin_att_name* and *connect_att_name* correspond to attribute names in EA, AA and CA respectively. *string* are c-style strings limited to printable characters. Otherwise, our policy language functions like c-style boolean statements where the only variables are attributes.

The following are example policy strings using the HGABAC policy language:

(a) *user.id* IN {5, 72, 4, 6, 4} OR *user.id* = *object.owner*
(b) *object.required_perms* SUBSET *user.perms* AND *user.age* >= 18
(c) *user.admin* OR (*user.role* = "doctor" AND *user.id* ! = *object.patient*)

Policy string *a* would only return true when processed by the *authorized* function, if the user attribute *id* is present and has at least one value matching an element in the given set {5, 72, 4, 6, 4} or has a value that is equal to a value in *object.owner*. Note that the value of the attribute *user.id* may be a set of multiple values, which would still pass the policy so long as $\exists e \in user.id : e \in \{5, 72, 4, 6, 4\}$ or $e = object.owner$. Policy string *b* would limit access to a user who is at least 18 and has the set of permissions such that the object's *required_perms* is a subset. Finally, policy string *c* demonstrates a possible use case, where you desire to give doctors access to any medical record but their own (as well as allow a user with the admin attribute to access any record).

4 Examples and Evaluation

4.1 Example: The Library

This section outlines how HGABAC may be used to provide access control for a hypothetical university library. In the following use cases it is assumed that access control is desired on four different kinds of resources provided by the library; books, course material (textbooks, lecture notes, etc.), periodicals, and archived records.

Case 1: Undergraduate students may check out any unrestricted book and any course materials for a course in which they are enrolled.
Case 2: Graduate students may check out any unrestricted book or periodical but may only check out course materials for courses in which they are a teaching assistant or enrolled.
Case 3: Faculty may check out any book, periodical or course material as well as any archived record from their department.
Case 4: Staff may access any resource between the hours of 8:00 and 17:00 on weekdays.
Case 5: Students enrolled in a computer science course may access periodicals from the university network.

Group Graphs: Figure 3 shows the the user and object group hierarchies that would be created by an administrator for the above example and cases.

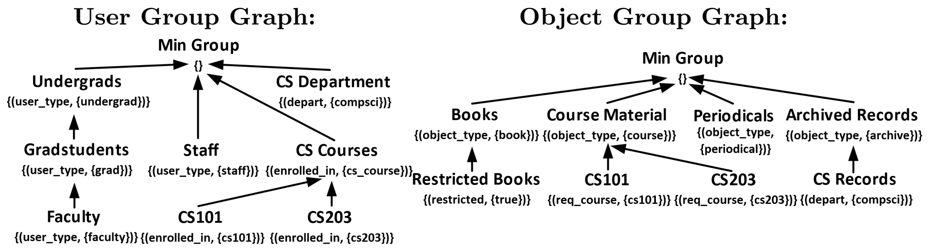


Fig. 3. User and object group hierarchies to support the cases given in Section 4.1.

Group Membership: Users are assigned to one of the four user_type groups (Undergrads, Gradstudents, Faculty or Staff) as expected (i.e. undergraduate students are members of the "Undergrads" group, graduate students to the "Gradstudents" group, etc.). Students are also assigned membership in a user group for each course they are enrolled in (e.g. if a student is enrolled in CS203 they would be a member of the user group "CS203"). Graduate students and faculty also belong to a department group (for the purposes of these cases, only a computer science department group, "CS Department", is considered). For example a computer science graduate student taking the course CS203 would be a member of the "Gradstudents","CS203" and "CS Department" groups and would have the effective attributes {{*user_type,* {*undergrad, grad*}}*,* {*enrolled_in,* {*cs_course, cs203*}}*,* {*depart,* {*compsci*}}.

Resources are assigned membership in one of the four object_type groups or one of their children as expected (i.e. books are assigned to the "Books" group or the "Restricted Books" group, course material to one of the course object groups (e.g. "CS101"), etc.). For example a textbook for the course CS101 would be assigned to the "CS101" object group and would have the following effective attributes {{*object_type,* {*course*}}*,* {*req_course,* {*cs101*}}}.

Case 1: One permission pair would be sufficient for meeting the requirements of the case: *PERMS* = {{*""undergrad" IN user.user_type AND ((object.object_type = "book" AND NOT object.restricted) OR (object.object_type = "course" AND user.enrolled_in IN object.req_course))", check_out_book*}} where "check_out_book" is the operation that allows a resource to be read/viewed.

Case 2: In this case each graduate student would be assigned an attribute "teaching" containing the set of courses the graduate student is assigned to as a TA. The following permission pair combined with the pair from Case 1 would be sufficient for meeting the requirements of the case: *PERMS* = {{*""grad" IN user.user_type AND (object.object_type = "periodical" OR (object.object_type = "course" AND object.req_course IN user.teaching))", check_out_book*}}. As the "Gradstudents" group is a child of the "Undergrads" group, graduate students are granted access to unrestricted books and course materials for courses they are enrolled in through the policy permission pair in Case 1 (as they have both the values "grad" and "undergrad" for their user_type attribute).

Case 3: As this case is less restrictive than the previous it can be met by a straightforward permission pair: *PERMS* = {{*""faculty" IN user.user_type AND (object.object_type IN* {*"book", "periodical", "course"*} *OR (object.object_type = "archive" AND object.depart IN user.depart))", check_out_book*}}.

Case 4: For this case at least two environment attributes are required. "time_of_day_hour", that represents the current hour (1 to 24) and, "day_of_week", that represents the current day of the week (1 to 7). Then the following permission pair would be sufficient for meeting the requirements for the case: *PERMS* = {{*""staff" IN user.user_type AND env.time_of_day_hour >= 8 AND env.time_of_day_hour <= 16 AND env.day_of_week IN* {*2, 3, 4, 5, 6*}*", check_out_book*}}.

Case 5: It is assumed that four connection attributes exist which represent the user's IP address; "ip_octet_1" represents the first digit of the user's IP address, "ip_octet_2", the second and so on up to "ip_octet_4". It is also assumed that all IP addresses matching the pattern "192.168.*.*" are internal addresses on the university's network. The following permission pair would then be sufficient for meeting the requirements of the case: *PERMS* = {{ *""cs_course" IN user.enrolled_in AND connect.ip_octet_1 = 192 AND connect.ip_octet_2 = 168 AND object.object_type = "periodical"", check_out_book*}}.

4.2 Evaluation

To evaluate whether the hierarchical user and object groups of the HGABAC model provides an advantage over more traditional non hierarchical models of ABAC in terms of simplifying administration and reducing complexity, we evaluate HGABAC based on the number of attribute and group assignments needed to fulfill the requirements of each use case given in Section 4.1. These results are compared to the number of attribute assignments that would be required in a non hierarchical model of ABAC such as ABAC$_\alpha$ [8] (if ABAC$_\alpha$ supported environment and connection attributes required to model cases 4 and 5).

Table 2 outlines the results of this comparison. The worst case (each user is enrolled in each course and each object is of an object_type such that it will have the most attributes) is assumed as well as a constant number of courses and departments (the same number shown in the group graphs in Figure 3). In cases 1, 2 and 3 where it is required that multiple attributes be assigned to each object and user, HGABAC has a noticeable advantage as hierarchical groups allow multiple attributes to be assigned with a single group membership assignment. This also has significant advantages for administration of ABAC systems, for example if an administrative tasks required adding an attribute to every student in a given course, only a single additional attribute assignment to the course's user group would be required in HGABAC, while a new attribute assignment for every user in the course would be required in traditional ABAC. Cases 4, and 5 take less advantage of HGABAC's group hierarchy, instead making use of connection and environment attributes, and as such results in HGABAC having a comparable performance to traditional ABAC but with a slight overhead due to the object and user groups.

Table 2. Number of attribute and group assignments required for each case in Section 4.1. U is the number of users and O is the number of objects.

	Case 1		**Case 2**		**Case 3**	
	HGABAC	ABAC	HGABAC	ABAC	HGABAC	ABAC
User Attribute Assignments	4	$4U$	$U+5$	$5U$	4	$2U$
Object Attribute Assignments	5	$2O$	6	$2O$	8	$2O$
User Group Assignments	$3U$	0	$3U$	0	$2U$	0
Object Group Assignments	O	0	O	0	O	0
Total Assignments	$3U+O+9$	$4U+2O$	$4U+O+11$	$5U+2O$	$2U+O+12$	$2U+2O$

	Case 4		**Case 5**			
	HGABAC	ABAC	HGABAC	ABAC		
User Attribute Assignments	1	U	1	U		
Object Attribute Assignments	0	0	1	O		
User Group Assignments	U	0	U	0		
Object Group Assignments	0	0	O	0		
Total Assignments	$U+1$	U	$U+O+2$	$U+O$		

5 Emulating Traditional Models

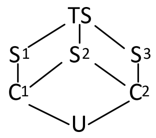
5.1 DAC Style Configuration

HGABAC can be configured to emulate DAC by assigning each user an "id" attribute with a single value equal to a unique identifier for that user and assigning each object an attribute for each access mode (e.g. "read" and "write") that contains the set of user ids corresponding to users who have access to that object for the given access mode. The set of permissions are then simply: *PERMS* = {("user.id IN object.read", read), ("user.id IN object.write", write)}. To model DAC style administration, an "owner" attribute maybe added to objects that contains a single user id corresponding to the owner of the object. The permission to grant access on administrative operations is then simply: ("user.id = object.owner", admin_operation).

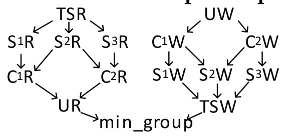
5.2 MAC Style Configuration

HGABAC's user groups allow configurations that emulate MAC style lattice based access control. For example given the following MAC lattice:

MAC Lattice:



User Group Graph:

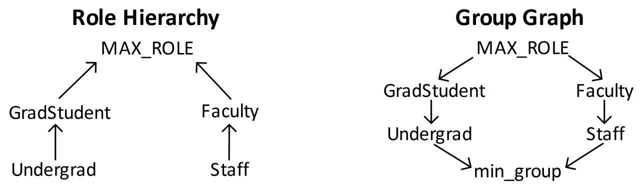


The user group graph may be configured as follows to enable MAC with a liberal *-property where each user is assigned only to a single read group and a single write group. This is similar to how RBAC is configured to emulate MAC in [13]. Each read group is assigned a single attribute named "read" and each write group is assigned a single attribute named "write" both with a single value equal to its clearance level (e.g. group UR is assigned the value {"UR"} for its "read" attribute). Each object is assigned a security level attribute named "level". The set of permissions are then simply: *PERMS* = {("object.level IN user.read", read), ("object.level IN user.write", write)}. Users are limited to only activating attributes inherited from groups of a single security level in any given session. The following table shows direct(g) and effective(g) for each group:

g	**direct(g)**	**effective(g)**
min_group	∅	∅
UR	"UR"	"UR"
C_1R	"C1R"	"UR", "C1R"
C_2R	"C2R"	"UR", "C2R"
S_1R	"S1R"	"UR", "C1R", "S1R"
S_2R	"S2R"	"UR", "C1R", "C2R", "S2R"
S_3R	"S3R"	"UR", "C2R", "S3R"
TSR	"TSR"	"UR", "C1R", "C2R", "S1R", "S2R", "S3R", "TSR"
TSW	"TSW"	"TSW"
S_1W	"S1W"	"TSW", "S1W"
S_2W	"S2W"	"TSW", "S2W"
S_3W	"S2W"	"TSW", "S3W"
C_1W	"C1W"	"TSW", "S1W", "S2W", "C1W"
C_2W	"C2W"	"TSW", "S2W", "S3W", "C2W"
UW	"UW"	"TSW", "S1W", "S2W", "S3W", "C1W", "C2W", "UW"

5.3 RBAC Style Configuration

HGABAC's user groups can also effectively enforce hierarchical RBAC style access control by having each user group represent a role and its assigned attributes, represent permissions. For example given the following role hierarchy, the user group graph on the right may be used:



Each group is assigned a single attribute named "perms" that contains the set of permissions that group grants. Objects are tagged with an attribute for each access mode whose value contains the set of permissions that grant permission to perform that access mode on the object. For example, an object may have a "read" attribute with values p_1 and p_4 and a "write" attribute with values p_2 and p_3. The set of permissions are then simply: $PERMS = \{($"user.perms IN object.read", read), ("user.perms IN object.write", write)$\}$ assuming the only access modes are read and write.

If the roles in the above example role hierarchy have the following directly assigned permissions, then the groups in the user group graph will have the following direct and effective values for the attribute "perms":

Role	**Direct Permissions**
Undergrad	P_1
Staff	P_2
GradStudent	P_3, P_4
Faculty	P_5, P_6
MAX_ROLE	$\emptyset$

g	**direct(g)**	**effective(g)**
min_group	$\emptyset$	$\emptyset$
Undergrad	P_1	P_1
Staff	P_2	P_2
GradStudent	P_3, P_4	P_1, P_3, P_4
Faculty	P_5, P_6	P_2, P_5, P_6
MAX_ROLE	$\emptyset$	P_1, P_2, P_3, P_4, P_5, P_6

While this enables HGABAC to emulate core and hierarchical RBAC (as defined in the NIST RBAC standard[6]), work towards emulating the separation of duty style constraints possible in NIST RBAC is left to future work.

6 Conclusions & Future Work

We have introduced a new model of ABAC, entitled HGABAC, that supports boolean rule based ABAC, hierarchical user and object groups, as well as environment, connection and administrative attributes. We show that adding user and object groups enables greater flexibility when modelling real world situations in addition to simplifying administration.

Future work in terms of formalizing a model of ABAC should largely consist of extending HGABAC to support features required for real world use of ABAC systems. Some potential additions include support for separation of duty, delegation, and access control for administrative functions. Expanding the policy language defined in Section 3.2 or alternatively exploring using XACML in it's place could lead to greater flexibility in supported policies. To achieve the full potential of ABAC, further automation is needed in terms of attribute assignment and group membership. The addition of conditional user and object group membership could also have interesting applications and implications that are worthy of future research.

References

[1] D. Bell and L. Padula. *Secure Computer Systems: Mathematical Foundations and Model*. Mitre, 1974.

[2] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 321–334. IEEE, 2007.

[3] S. M. Chandran and J. B. Joshi. LoT-RBAC: A location and time-based RBAC model. In *Web Information Systems Engineering–WISE 2005*, pages 361–375. Springer, 2005.

[4] L. Chen and J. Crampton. Risk-aware role-based access control. In *Proceedings of the 7th international conference on Security and Trust Management*, pages 140–156. Springer-Verlag, 2011.

[5] D. E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, May 1976.

[6] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.

[7] S. Godik, A. Anderson, B. Parducci, P. Humenn, and S. Vajjhala. OASIS extensible access control 2 markup language (XACML) 3. Technical report, OASIS, 2002.

[8] X. Jin, R. Krishnan, and R. Sandhu. A unified attribute-based access control model covering DAC, MAC and RBAC. In *Data and Applications Security and Privacy XXVI*, pages 41–55. Springer, 2012.

[9] S. C. Kleene. On notation for ordinal numbers. *The Journal of Symbolic Logic*, 3(4):150–155, 1938.

[10] D. R. Kuhn, E. J. Coyne, and T. R. Weil. Adding attributes to role-based access control. *IEEE Computer*, 43(6):79–81, 2010.

[11] B. W. Lampson. Protection. *ACM SIGOPS Operating Systems Review*, 8(1):18–24, 1974.

[12] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, and T. Freeman. A flexible attribute based access control method for grid computing. *Journal of Grid Computing*, 7(2):169–180, 2009.

[13] S. Osborn, R. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(2):85–106, 2000.

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

[15] D. Servos. A role and attribute based encryption approach to privacy and security in cloud based health services. Master's thesis, Lakehead University, 2012.

[16] D. Servos, S. Mohammed, J. Fiaidhi, and T.-H. Kim. Extensions to ciphertext–policy attribute–based encryption to support distributed environments. *International Journal of Computer Applications in Technology*, 47(2):215–226, 2013.

[17] H.-b. Shen and F. Hong. An attribute-based access control model for web services. In *Parallel and Distributed Computing, Applications and Technologies, 2006. PDCAT'06. Seventh International Conference on*, pages 74–79. IEEE, 2006.

[18] L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, pages 45–55. ACM, 2004.

[19] E. Yuan and J. Tong. Attributed based access control (ABAC) for web services. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*. IEEE, 2005.