ABAC with Group Attributes and Attribute Hierarchies Utilizing the Policy Machine

Smriti Bhatt
Institute for Cyber Security
Dept. of Computer Science
Univ. of Texas at San Antonio
San Antonio, TX 78249
Smriti.Bhatt@my.utsa.edu

Farhan Patwa
Institute for Cyber Security
Dept. of Computer Science
Univ. of Texas at San Antonio
San Antonio, TX 78249
Farhan.Patwa@utsa.edu

Ravi Sandhu
Institute for Cyber Security
Dept. of Computer Science
Univ. of Texas at San Antonio
San Antonio, TX 78249
Ravi.Sandhu@utsa.edu

ABSTRACT

Attribute-Based Access Control (ABAC) has received significant attention in recent years, although the concept has been around for over two decades now. Many ABAC models, with different variations, have been proposed and formalized. Besides basic ABAC models, there are models designed with additional capabilities such as group attributes, group and attribute hierarchies and so on. Hierarchical relationship among groups and attributes enhances access control flexibility and facilitates attribute management and administration. However, implementation and demonstration of ABAC models in real-world applications is still lacking. In this paper, we present a restricted HGABAC (*rHGABAC*) model with user and object groups and group hierarchy. We then introduce attribute hierarchies in this model. We also present an authorization architecture for implementing *rHGABAC* utilizing the NIST Policy Machine (PM). PM allows to define attribute-based access control policies, however, the attributes in PM are different in nature than attributes in typical ABAC models as name-value pairs. We identify a policy configuration mechanism for our proposed model employing PM capabilities, and demonstrate use cases and their configuration and implementation in PM using our authorization architecture.

Keywords

Attribute-Based Access Control; Group Attributes; Group Hierarchy; Attribute Hierarchy; Policy Machine;

1. INTRODUCTION

Attribute-Based Access Control (ABAC) [1, 2] has been around for over two decades. Numerous ABAC models [3–10] have been proposed. Despite the existence of these different ABAC models, there is no consensus on a specific standard ABAC model. However, a well-accepted simplest form of attribute-based access control includes users, user attributes, objects, object attributes, actions, and permissions or operations allowed for users on specific objects, based on attributes of users and objects. This typical foundation provides great freedom to researchers to incrementally enhance this basic ABAC architecture based on customized needs and requirements in different scenarios. Any additional component that uses or is compatible with the basic ABAC components can be incorporated with them in order to get a more powerful and flexible ABAC model.

The move towards ABAC models is mainly inspired by limitations of traditional access control models. Three most significant and widely adopted access control models are Discretionary Access Control (DAC) [11], Mandatory Access Control (MAC) [11], and Role-Based Access Control (RBAC) [12–14]. Among these, RBAC has been the most successful since its introduction in early 1990s. RBAC is policy neutral and is capable of expressing DAC and MAC. However, role-based access control has its own set of limitations such as role explosion and role-permission explosion [15]. It is also restrictive in nature since accesses are based only on roles and it is difficult to include other characteristics of users, and contextual or environmental factors (e.g. time, location, etc.) in access control policies.

On the other hand, attribute-based access control policies can easily incorporate these factors with environmental and contextual attributes [3, 4], as well as define access control policies based on different characteristics of users and objects, besides roles. Although ABAC research has received significant attention in academia, it is not so common to find implementations of these models in the industry. There are a few existing tools such as XAMCL [16] and Policy Machine (PM) [17–19] that are capable of expressing different types of attribute-based access control policies. However, wide adoption of these tools remains a challenge.

In this paper, we present a restricted HGABAC [4] model known as *rHGABAC* including user and object groups, group attributes, and group hierarchy. We then introduce attribute hierarchies [5, 20] to *rHGABAC* model. Unlike most ABAC models formalized as logical-formula authorization policy (LAP) [21], we formalize *rHGABAC* as a single-value enumerated policy. In a single-value enumerated policy, a policy tuple comprises of a single value of user attribute and a single value of object attribute. Due to this property, *rHGABAC* model is restricted in nature and is not capable of expressing some kinds of policies which can be represented in HGABAC or other types of enumerated policies. An example of policy that can't be expressed in *rHGABAC* is where a user who is both a *Manager* and an *Admin* can access objects of type

ABAC'17, March 24 2017, Scottsdale, AZ, USA
© 2017 ACM. ISBN 978-1-4503-4910-9/17/03…$15.00
DOI: http://dx.doi.org/10.1145/3041048.3041053

Private. A different policy where a user who is a *Manager* or an *Admin* can access objects of type *Private* can be easily expressed in *rHGABAC*. In other words, *rHGABAC* cannot express conjunctive policies.

PM is also a single-value enumerated authorization policy which makes it a feasible choice for implementing *rHGABAC* model. Although PM hasn't been specifically designed for group attributes and hierarchical relationships, we can realize these features with its containment property. The containment property exists among PM attributes through its *assignment* relation. It implies that if there exist any two elements x and y such that x is assigned to (contained in) y by one or more assignment relations, then x acquires or gets all the properties and capabilities of y in addition to its own directly conferred properties [18].

Previously, we have proposed a role-centric [22] ABAC extension for OpenStack [23] and enforced it in OpenStack using the PM and our authorization engine (AE) [10]. Whereas, here instead of an application dependent AE, we present a more general authorization architecture independent of applications using it. A hierarchical ABAC model with group attributes, hierarchical relations, and set-valued attributes allows to define more fine-grained access control policies and also simplifies the management and administration of attributes and policy. A use case incorporating both aspects of *rHGABAC* model—group attributes and hierarchy, and attribute hierarchy—is configured and implemented in the PM using this authorization architecture. These added capabilities facilitate attribute assignment for users and objects thereby reducing administrative tasks.

The rest of the paper is organized as follows. Section 2 presents relevant background on group attributes and attribute hierarchies, and the Policy Machine (PM). Section 3 presents *rHGABAC* model and an extended version of the model with attribute hierarchies, along with formal definitions. Section 4 discusses the details of our implementation and authorization architecture. In Section 5, we demonstrate a use case and its configuration and implementation. We evaluate policy decision time for different types of access control policies in the PM and Authorization Engine in Section 6, and finally conclude the paper in Section 7.

2. BACKGROUND

In this section, we briefly describe relevant background on group attributes and attribute hierarchies and their significance in access control. We also discuss the Policy Machine (PM) and its architecture. Generally, any ABAC model can be expressed based on logical-formula authorization policy (LAP) or enumerated authorization policy (EAP) [21]. However, most of the proposed ABAC models [3,4,7–9,24, 25] in literature are based on LAP, with very few EAP-ABAC models [5,18,21,26].

A logical-formula based authorization policy comprises of a boolean expression or a number of sub-expressions connected using logical operators such as and($\wedge$), or($\vee$), etc. These expressions consist of user and object attribute values or constants, specified in an authorization policy, which are matched with actual attribute values of a requesting user and a requested object for an access request and returns either true (access granted) or false (access denied). One of the possible problems in this method of policy specification is policy update and policy review. It would be a NP-complete or even undecidable problem if policies are specified respectively in propositional logic and first-order logic. Whereas, the other method of defining ABAC policies by enumeration uses simple structure which makes policy update and review a polynomial time problem [5] (at the cost of exponential increase in size). In an enumerated authorization policy, a policy is a set of policy tuples defined for a specific operation (read, write, etc.) which includes a user attribute value and an object attribute value. An example tuple, for read operation, is informally defined as $Policy_{read} = (Manager, Private)$. It implies that a user with title as *Manager* can read any object of type *Private*. Here, *title* is a user attribute and *type* is an object attribute.

Examples of above discussed methods of specifying ABAC policies in industry standards are XACML (an OASIS standard and a LAP) and NGAC (Next Generation Access Control) [27] (a NIST standard and an EAP). PM has been the foundation for the NIST NGAC standard. Both standards have similar goals and objectives and are inherently capable of expressing and defining different types of ABAC policies in different environments. While XACML is a well established standard and has also been realized in some of the real-world ABAC systems [28], NGAC remains to be studied in detail with different types of ABAC models for its adoption in real-world applications. A comparison of these two standards based on five criteria—(i) separation of access control functionality from proprietary operating environments, (ii) operational efficiency, (iii) attribute and policy management, (iv) scope and type of policy support, and (v) support for administrative review and resource discovery—along with their basics is presented and discussed in [27,29].

NGAC, or generally PM, provides almost complete access control functionality with its own policy administration point (PAP), policy decision point (PDP), and policy enforcement point (PEP), whereas XACML doesn't provide a PEP and is in turn dependent on the operating environment. A PM deployment could also include a different PEP accessible through an application programming interface (API) which is capable of recognizing operating-environment specific operations. We utilize this form of architecture where PEP is application dependent, hence these applications become PM-agnostic and need not be aware of PM relations and routines.

While NGAC provides better support for attribute and policy management, administrative review and resource discovery, XACML is capable of expressing complex and rich set of access control rules including negative and integer attribute values which is difficult to represent in NGAC without using *prohibition* and *obligation* relations [18,27,29].

2.1 Group Attributes and Hierarchies

Groups are collection of entities such as users and objects. Group attributes are attributes assigned to groups of users and objects. These groups based on their nature (user or object) get relevant attributes assigned to them which represents the group characteristics. This is especially beneficial when there exist many users in a system having common characteristics. Therefore, instead of assigning same attribute values to each user, we can group the users into specific groups and assign appropriate attributes and their values to these groups. Similar, is the case for objects where objects having same properties and characteristics can be grouped into one object group. In addition to directly assigned attribute-value pairs, these groups also have hierar-

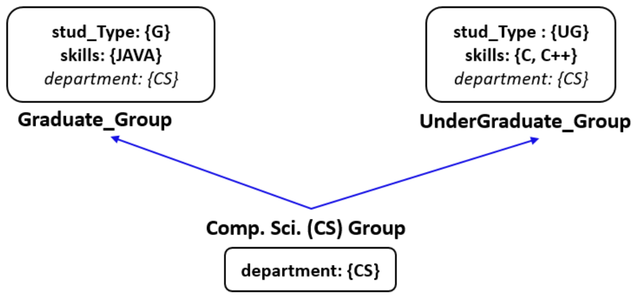
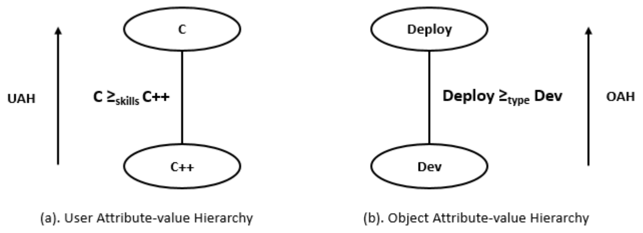


Figure 1: An Example of User Group Hierarchy Adapted from [30]



UAH
C
C ≥skills C++
C++
Deploy
Deploy ≥type Dev
Dev
OAH
(a). User Attribute-value Hierarchy
(b). Object Attribute-value Hierarchy

Figure 2: An Example of Attribute Hierarchy

chy among them which results in attribute inheritance. A group hierarchy is a partial order relation written as $\succeq_g$ where senior groups acquire all attribute values assigned to the groups junior to them, along with their own directly assigned attribute values.

A simple user group hierarchy example is shown in Figure 1. There are three groups *Computer Science Group, Graduate_Group* and *Undergraduate_Group*. Among these graduate and undergraduate are senior groups and are represented higher up whereas computer science is a junior group and is one level below the senior groups. An attribute and its values are written as *att_name* : $\{Val_1, Val_2, ..., Val_n\}$. A directly assigned attribute-value is shown in bold font and an inherited attribute-value is shown as italicized in normal font. In this example the senior groups, *Graduate_Group* and *Undergraduate_Group*, inherit attribute *department* and its value *CS* from junior group *Computer Science Group*, and also have directly assigned attributes *skills* and *stud_Type*.

Servos and Osborn [4] introduced group attributes and hierarchies in ABAC and presented a formal hierarchical group and attribute-based access control model, known as HGABAC. The main advantage of this model is that it facilitates the administrative tasks of assigning attributes and values to user and object groups rather than assignning them to each user or object. An administrative model for HGABAC has been developed by Gupta and Sandhu [30]. Note that HGABAC requires attributes to be set-valued so inheritance can be seamlessly achieved. Atomic-valued attributes, which can have only one value, would require some kind of conflict resolution in the presence of inheritance. Therefore, in this paper we follow the HGABAC approach.

2.2 Attribute Hierarchies

The concept of attribute hierarchy has been explored in literature in context of attribute-based encryption by Li et al [20]. They presented a hierarchical attribute-based encryption (HABE) mechanism where attributes can be classified in a tree structure based on their access control relationship in a system. An abstract idea of the encryption mechanism is that attributes being the nodes in a hierarchical tree, an ancestral node can derive its descendant's key, but converse is not allowed. However, our concept of attribute hierarchy in ABAC considers hierarchy among attribute values rather than attributes themselves. A similar concept has been introduced in [5] by Biswas et al where an ABAC model (LaBAC$_H$), with one user attribute and one object attribute, have hierarchical relationship among attribute values. The hierarchical relationship can be written as $\succeq_{att}$ and implies if a senior attribute value is assigned to a user or an object then all the junior attribute values are automatically acquired by that user or object through attribute-value inheritance.

An example of attribute hierarchy is shown in Figure 2. User attribute hierarchy is depicted in Figure 2 (a) where C and $C{+}{+}$ belongs to the range of a user attribute named*skills*. There is a partial order relation between C and $C{+}{+}$, represented as $C \succeq_{skills} C{+}{+}$ which implies that value C is senior to $C{+}{+}$. Therefore, if a user attribute value C is assigned to a user then it also acquires user attribute value $C++$ and all its associated access privileges through attribute-value inheritance. Similarly in Figure 2 (b), an object attribute hierarchy is shown where *type* is an object attribute and there is a hierarchical relation between two of its values *Deploy* and *Dev* represented as $Deploy \succeq_{type} Dev$. It implies that if an object is assigned object attribute value *Deploy* of object attribute *type*, then that object automatically gets its junior attribute value, *Dev*, assigned to it. The hierarchy also implies that the objects assigned to senior object attribute value, say *Deploy*, gets associated with access rights imposed on this attribute value as well as access rights imposed on all of its junior object attribute values such as *Dev* here. Thus, attribute hierarchy facilitates policy management and attribute management. It also simplifies administrative task of assigning attributes and their values to users and objects, and specifying access control policies in an ABAC model.

2.3 Policy Machine

Policy Machine (PM) [18, 19] is an access control framework where access control mechanisms are redefined in terms of a standardized and generic set of relations and functions. Its main objective is to provide a general and unified framework to support different types of attribute-based policies or policy combinations through a single mechanism requiring changes only in its data configuration points. Overall, PM has eight core elements: *users, objects, user attributes, object attributes, operations, processes, access rights* and *policy classes*. The policy classes, user attributes and object attributes are containers for policies, users and objects respectively. In addition to core elements, it has four types of relations: *assignment, association, prohibition* and *obligation*, and two sets of functions: *access control decisions* and *policy enforcement*. The policies in the PM can be configured in its *Administration Tool*, via a fixed set of administrative relations for defining wide range of access control

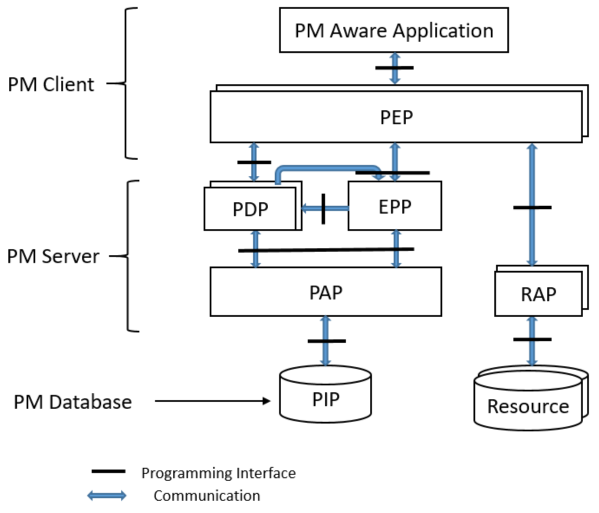


Figure 3: Architectural Components of the PM Adapted from [18]

policies and their combinations. PM functions helps in making access control decisions, and enforcing these decisions.

PM's standard architecture comprises one or more PM servers, one or more PM clients, a PM database, and one or more resource servers. The architectural components of the PM are shown in Figure 3. PM server includes a policy decision point (PDP), a policy administration point (PAP) and an event processing point (EPP). PM clients comprise of a policy enforcement point (PEP) and PM aware applications through which users requests access to protected resources. For a general application to be PM compliant, the application needs to be modified to incorporate and be able to communicate to the PEP in the PM which talks to the PDP for access control decisions. A PDP determines whether an access request made by PEP should be granted or denied as per the policy defined in PAP. All the information regarding access control data and relations is stored in a PM database, a policy information point (PIP).

In PM version 1.5, Active Directory (AD) has been used as the PM database where all the information about user, user attributes, object attributes, references to objects, access control relations and policies are stored. EPP module, in the PM server, is triggered only if any event occurs as specified in the *obligation* relation. We do not consider *obligation* and *prohibition* relations in this paper. After getting the access control decision from PDP, PEP enforces the decision on the applications using a resource access point (RAP) to get the actual location of the resources in order to perform a requested operation [18].

Here, we utilize a minimized version of the PM, with only two types of relations: *assignment*—for specifying policies, users, user attributes, user groups, objects, object attributes, object groups, and hierarchical relationships among them, and *association*— for making association between user attributes and object attributes or objects through operations defined in the PM. Association are the relations used for defining policies in the PM. PM also allows use of multiple policy classes in order to express multi-attributes access control policies or conjunctive policies. Expressing such policies require combination of multiple policy classes with same objects and users which would increase the administrative overhead. Therefore, in context of this paper, we utilize a minimized version of the PM with a single policy class to specify and enforce different types of access control policies. Using this minimized version of PM enables us to simplify the policy administration, policy update, and policy review.

From an architectural perspective, we basically include PM server (excluding EPP) and a PM database (Active Directory). Applications encapsulates the PEP, RAP, resources and resource repositories. The details of applications and how they implement these components are outside the scope of this paper.

3. A RESTRICTED HGABAC MODEL

Servos and Osborn [4] proposed a hierarchical ABAC model known as hierarchical group and attribute-based access control (HGABAC) model. In addition to basic ABAC components, this model includes user and object groups and introduces attribute inheritance through hierarchy among these groups. It also includes environment, connection, and administrative attributes in access control policies. A group-based hierarchical assignment of user and object attributes is a novel part of this model.

As discussed earlier, HGABAC is a logical-formula based authorization policy model. However, here we present a restricted HGABAC model, named *rHGABAC*, and formalize this model as a single-value enumerated authorization policy. The main motivation behind this model is to include features such as groups, groups attributes, hierarchies in an ABAC model and show its implementation as enumerated policy utilizing an enumerated policy tool, the PM. In process of developing an administrative model for HGABAC, named $\mathrm{GURA_G}$, Gupta and Sandhu presented a conceptual model of HGABAC with an alternate formalization for it based on ABAC_α [30]. Our *rHGABAC* model has been adapted from their model, however, formalized as a single-value enumerated authorization policy. The model is shown in Figure 4 with its formalization presented in Table 1.

The major components of this model are *users, user groups, user attributes, objects, object groups, object attributes, operations, policy,* and *authorization.* There is hierarchical relationship (i.e., a partial order) among groups—user group hierarchy *(UGH)* and object group hierarchy *(OGH)* through which attribute inheritance is achieved. For simplicity we have removed subjects from the model and thus consider that users and subjects are equivalent. We have also modified the *operations (OP)* component of the conceptual model with *policy*—a collection of policies defined for each operation in *OP*. In context of this paper, we ignore all the *assignment* relations from the conceptual model of HGABAC, since we focus only on the operational model rather than the administrative part of the model.

Users (U) is a set of individuals or automated entities (a system or a process) which make requests to access objects, where *objects (O)* is a set of resources such as files, directories, applications, etc. Users and objects are associated with specific set of *attributes (Att)* respectively. These attributes reflects the properties and characteristics of users and objects. Each attribute is a function that takes an entity—a user, an object, a user group, or an object group, and returns one or more values from its range. The range of an attribute consists of a finite set of atomic values.

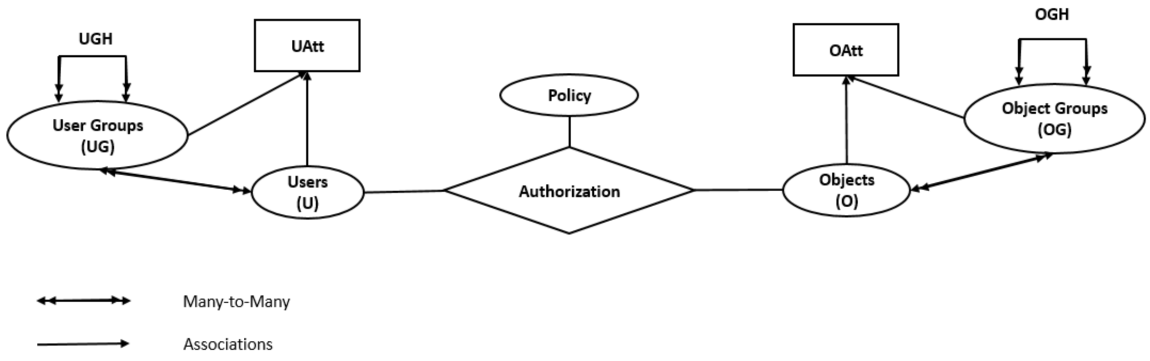


Figure 4: *rHGABAC* Model Adapted from [4], [30]

Table 1: *rHGABAC* Model with single-value EAP

I. Core Components
- *U, O* and *OP* are finite sets of users, objects, and operations respectively
- *UG* and *OG* are finite set of user and object groups respectively
- *UAtt* and *OAtt* are finite set of user attributes and object attributes functions respectively
- For each att in $UAtt \cup OAtt$, $Range(att)$ is a finite set of atomic values, where $Range(att_i) \cap Range(att_j) = \phi$ for $i \neq j$
- For each $uatt$ in $UAtt$, $uatt : U \cup UG \rightarrow 2^{Range(uatt)}$, mapping each user and user group to a set of values in $Range(uatt)$
- For each $oatt$ in $OAtt$, $oatt : O \cup OG \rightarrow 2^{Range(oatt)}$, mapping each object and object group to a set of values in $Range(oatt)$
- $directUg : U \rightarrow 2^{UG}$, mapping each user to a set of user groups
- $directOg : O \rightarrow 2^{OG}$, mapping each object to a set of object groups
- $UGH \subseteq UG \times UG$, a partial order relation $\succeq_{ug}$ on UG
- $OGH \subseteq OG \times OG$, a partial order relation $\succeq_{og}$ on OG
II. Derived Components (Effective Attributes)
- For each $uatt$ in $UAtt$,
i) $effectiveUG_{uatt} : UG \rightarrow 2^{Range(uatt)}$, defined as
$effectiveUG_{uatt}(ug_i) = uatt(ug_i) \cup (\bigcup_{\forall g \in \{ug_j \mid ug_i \succeq_{ug} ug_j\}} effectiveUG_{uatt}(g))$
ii) $effective_{uatt} : U \rightarrow 2^{Range(uatt)}$, defined as
$effective_{uatt}(u) = uatt(u) \cup (\bigcup_{\forall g \in directUg(u)} effectiveUG_{uatt}(g))$
- For each $oatt$ in $OAtt$,
i) $effectiveOG_{oatt} : OG \rightarrow 2^{Range(oatt)}$, defined as
$effectiveOG_{oatt}(og_i) = oatt(og_i) \cup (\bigcup_{\forall g \in \{og_j \mid og_i \succeq_{og} og_j\}} effectiveOG_{oatt}(g))$
ii) $effective_{oatt} : O \rightarrow 2^{Range(oatt)}$, defined as
$effective_{oatt}(o) = oatt(o) \cup (\bigcup_{\forall g \in directOg(o)} effectiveOG_{oatt}(g))$
III. Policy Components
- $Policy_{op} \subseteq \{(ua_i, oa_j) \mid ua_i \in Range(uatt_k), oa_j \in Range(oatt_l)\}$, for $uatt_k \in UAtt$, $oatt_l \in OAtt$, and $op \in OP$
- $Policy = \{Policy_{op} \mid op \in OP\}$
IV. Authorization Function
- $is\_authorized(u : U, op : OP, o : O) = (\exists v_u \in effective_{uatt_i}(u) \mid uatt_i \in UAtt)$ and $(\exists v_o \in effective_{oatt_j}(o) \mid oatt_j \in OAtt)$, $[(v_u, v_o) \in Policy_{op}]$

Table 2: *rHGABAC* with Attribute Hierarchy (AH) as single-value EAP

I. Additional Core Components
- $UAH_i \subseteq Range(uatt_i) \times Range(uatt_i)$, a partial order relation $\succeq_{uatt_i}$ on $Range(uatt_i) \mid uatt_i \in UAtt$
- $OAH_j \subseteq Range(oatt_j) \times Range(oatt_j)$, a partial order relation $\succeq_{oatt_j}$ on $Range(oatt_j) \mid oatt_j \in OAtt$
II. Modified Authorization Function
- $is\_authorized(u : U, op : OP, o : O) = (\exists v_u \in effective_{uatt_i}(u) \mid uatt_i \in UAtt)$ and $(\exists v_o \in effective_{oatt_j}(o) \mid oatt_j \in OAtt)$, and $(\exists v_u \succeq_{uatt_i} v_u')$ and $(\exists v_o \succeq_{oatt_j} v_o')$,$[(v_u', v_o') \in Policy_{op}]$

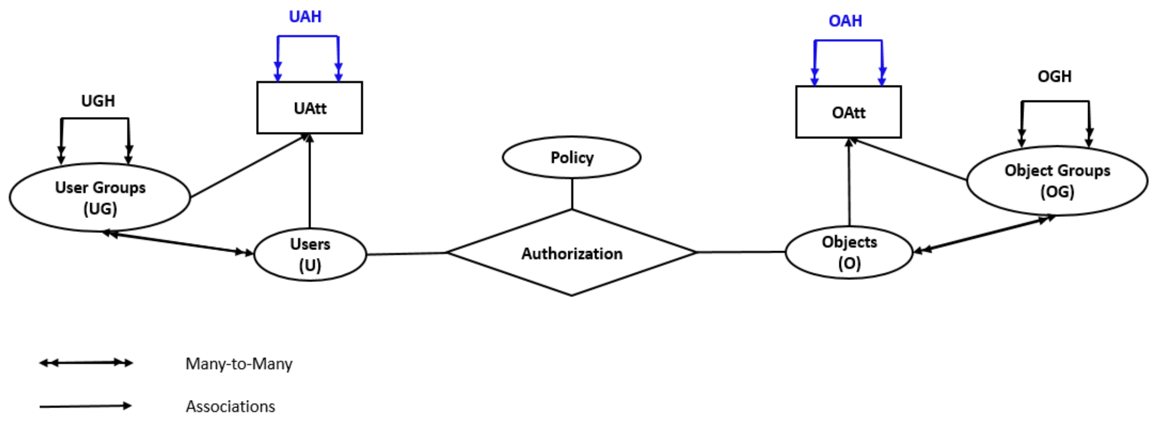


Figure 5: *rHGABAC* Model with Attribute Hierarchy

There are two types of attributes—*atomic valued* where only one value can be assigned to a attribute from its range, and *set valued* where a subset of values is assigned from the range of an attribute. All the attributes are set valued in this model. *User attributes (UAtt)* is a set of user attributes for users and user groups. Similarly, *object attributes (OAtt)* is a set of object attributes for object and object groups. *Operations (OP)* is a set of access rights such as *read*, *write*, etc. that can be performed on objects by users.

The set of user groups is UG and the set of object groups is OG. These groups have a many to many hierarchical relationship (a partial order relation $\succeq_g$) among them, represented as UGH and OGH respectively. Thus, any senior group inherits all the attributes and values from the groups junior to it. For example, $g_1 \succeq g_2$ implies that g_1 is senior and inherits all the attributes and values assigned to junior group g_2, in addition to its own directly assigned attributes and values. Users and objects can be assigned to zero or more user groups and object groups respectively. The function *directUg* takes a user and returns the set of user groups to which the user has been assigned, and *directOg* takes an object and returns the set of object groups to which the object belongs. A user assigned to a user group gets all the attributes and values assigned to the group as well as inherited attributes and their values from junior groups. The effective attribute values of a user group ug is defined as $effectiveUG_{uatt}(ug)$ and comprises of directly assigned attribute values to the user group *uatt(ug)* and inherited attribute values from all the groups junior to it, i.e. effective attribute values of all the groups which are junior to ug. Hence, effective attribute values of a user u, defined as $effective_{uatt}(u)$, consists of attribute values directly assigned to the user *uatt(u)* and effective attribute values of all the groups directly assigned to the user. Similarly, effective values of object and object group attributes can be obtained [30]. These derived components and their formulas are shown in section II of Table 1.

In section III of Table 1, *Policy* is set of policies where each policy is defined for a specific operation and is represented as $Policy_{read}$. A policy for a specific operation is set of policy tuples defined for a single-value of user attribute and a single-value of object attribute. The function, $is\_authorized(u, op, o)$, authorizes a user u to perform an operation op on object o, if there exists a policy tuple in $Policy_{read}$, which includes a user attribute value that belongs to the effective attribute values of user u and an object attribute value that belongs to the effective attribute values of object o.

3.1 Extending HGABAC with Attribute Hierarchies

Now, we extend *rHGABAC* model by adding attribute hierarchies as discussed earlier in Section 2. An attribute hierarchy, represented as ($\succeq_{att}$), is a partial order relation among attribute values, i.e., range of an attribute $Range(att)$. For example, if there exists a partial order relation $v_i \succeq_{uatt} v_j$ for a user attribute $uatt$ in $UAtt$, then a user or a user group having v_i value assigned directly will also get v_j value through attribute-value hierarchy.

The HGABAC model extended with partial order hierarchies among user attribute-values and object attribute-values is shown in Figure 5. The addition of attribute hierarchies requires modification of the authorization function.

The additional core components and modified authorization function are defined in Table 2. The authorization function is modified as it considers the effective attribute values of a user requesting access and a protected object through direct or group assignment along with attribute hierarchies, if any, among attribute-values of that user and object.

4. IMPLEMENTATION

In this section, we discuss the implementation details of our authorization framework utilizing the PM. It allows to configure and implement different types of enumerated and logical-formula (with some restrictions) authorization policies. First we present a generalized authorization architecture independent of any application or system, and second we discuss how two variations of *rHGABAC* model can be defined and implemented in the Policy Machine (PM).

The goal is to develop a general authorization framework which can be readily used by any application or system supporting RESTful service that are interested in implementing attribute-based access control policies. The architecture includes a PM Server as policy administration point (PAP) and policy decision point (PDP), and a PM Database used

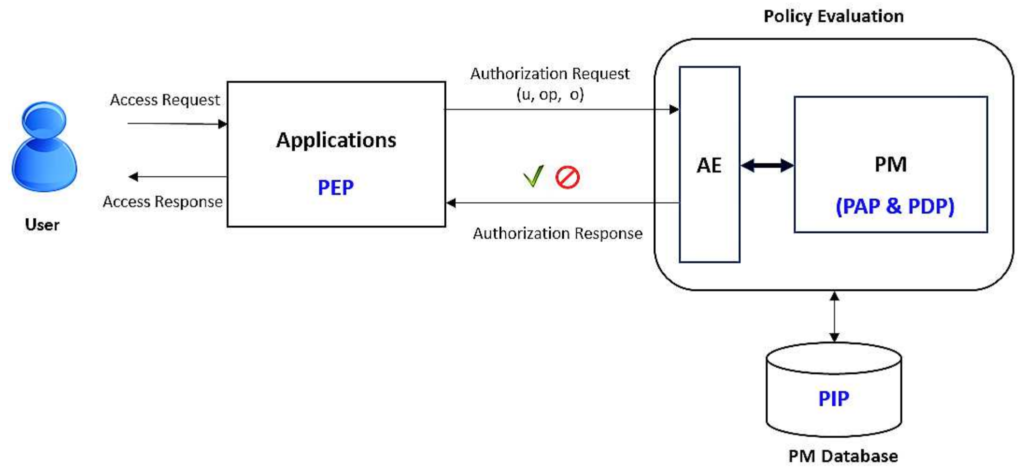
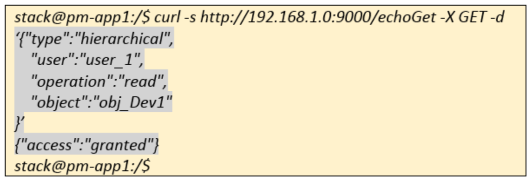


Figure 6: Authorization Architecture Utilizing PM and AE



```
stack@pm-app1:/$ curl -s http://192.168.1.0:9000/echoGet -X GET -d
'{"type":"hierarchical",
   "user":"user_1",
   "operation":"read",
   "object":"obj_Dev1"
}'
{"access":"granted"}
stack@pm-app1:/$
```

Figure 7: Example Authorization Request and Response

as policy information point (PIP) where all the access control data related to users, objects, their attributes, relations, etc. is stored.

4.1 Authorization Architecture

A generalized authorization architecture is shown in Figure 6. As compare to Figure 3, our authorization framework consists of a PM Server including PAP and PDP. Since we ignored *obligation* and *prohibition* relations for now, the architecture does not include an EPP. The PM Client is our authorization engine (AE), which actually communicates with PM Server on one side and acts as a HTTP server for the applications on the other. Therefore, it also can be thought of as an interface between PM Server and applications. Applications do not need to be modified to make them PM aware, and instead can readily use PM for their security policy requirements. However, the applications need to support RESTful service to make HTTP requests to AE. These applications are responsible for policy enforcement once the authorization decisions are made by the PM. The PEP in the application would enforce the decisions on specific resources residing in the resource repository of these applications. In this architecture, we assume both PM and applications use the same identity management system such as Active Directory (AD) here, to store information about users, user groups, their attributes, etc.

Applications act as a PEP and are responsible for enforcing the authorization decisions returned by PM through AE. They manage their objects and resources themselves including PAP and resources repository components as shown in PM architecture. PM Server stores references to these objects in the PM database and policies are defined based on these objects and their attributes. When a user requests access to resources in an application, the application issues a HTTP request to AE for evaluating the policy and gets the authorization decision from PM. The HTTP request would originate from a machine where the application is hosted.

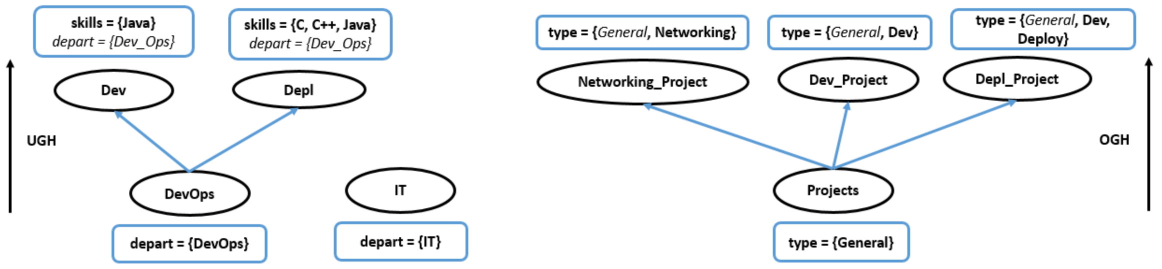
An authorization request comprises of a *user* requesting access, an *operation* (e.g. read, write, execute), and the requested *object*. The request is implemented as a HTTP GET. An example of the authorization request and its response is shown in Figure 7. The request includes the authorization data in JSON format and gets a HTTP response with authorization decision returned in JSON format as well.

4.2 Policy Configuration and Setup

Policy Machine (PM) provides a powerful and near complete access control framework with PDP, PAP, PIP, and PEP components included in one tool. Existing access control models such as DAC, MAC, and RBAC can be configured and implemented in PM [18,19]. PM_{mini}, a bare minimum version of PM with all the core elements and only two PM relations—*assignment*, and *association*—has been shown to be capable of representing attribute-based access control policies such as $LaBAC_H$ [5]. In $LaBAC_H$ model, there is only one user and one object attribute and this scenario maps exactly with PM's inherent way of configuring access control policies.

The attributes in PM are containers for users and objects, whereas the attributes in ABAC policies are attribute-value pairs and the values of these attributes are used in policy specification which determines allowed accesses for a user on an object. Using rich set of PM's capabilities and freedom to express access control policies in different ways, we identified an intuitive way of configuring attribute-based access control policies in the PM.

PM uses its user attributes and object attributes in policy definitions. Different aspects of $rHGABAC$ model have to be mapped within this territory to configure and implement it in the PM. Therefore, we represent user groups, user attributes, and user attribute-values of $rHGABAC$ model as PM's user attributes and object groups, object attributes, and object attribute-values as PM's object attributes. To incorporate values and hierarchical relations, we use *assign-*



(a). User Groups and their assigned user attributes and values

(b). Object Groups and their assigned object attribute and values

Figure 8: User and Object Groups with Associated Attributes

ment relation of the PM, taking advantage of the containment property of the attributes in PM. The policies for each operation are defined using *association* relation as defined for PM_{mini} [5]. We present use cases in following section to provide a better understanding of this mechanism.

5. USE CASES

In this section we discuss two variations of a use case, one with groups and group hierarchy, and the other with additional attribute-value hierarchies which is a modified version of the former. We believe our use cases closely resembles a real enterprise scenario and are capable of representing different features of our *rHGABAC* model. We have configured and implemented these use cases in the PM using our authorization architecture.

Enumerated and logical-formula policies are, in general, equally expressive. However, it is difficult to include negative attribute values in PM, such as *not a Manager* in a policy. For example defining a policy such as—a user with title *IT_Manager* and not with a title *CTO* is allowed to read objects with type *Networking*—is a complicated task in PM and requires use of *prohibition* relations, constraints, and combination of policies. However, a restricted HGABAC policy can be easily defined in PM using core elements and *assignment* and *association* relations.

5.1 Group Attribute and Group Hierarchy

This use case includes user groups, object groups, and hierarchy among groups besides other ABAC components. A set of user groups, their attributes and hierarchy among these groups is shown in Figure 8(a). Similarly, a set objects groups, their attributes and hierarchy among object groups is shown in Figure 8(b). There are four user groups *DevOps, IT, Development* and *Deployment* where there is group hierarchy among *DevOps*, *Development* and *Deployment*. *Development* and *Deployment* are senior to *DevOps* and inherit attributes and their values from it such as *depart* and its value. They also have their own directly assigned attribute, *skills*, with specific values. *IT* group does not have hierarchical relationship to any other group and has only one directly assigned attribute, *depart*, with value *IT*.

A similar object group hierarchy is shown in Figure 8(b) between four object groups, *Projects, Networking_Project, Dev_Project* and *Depl_Project*. *Projects* is junior to all the other groups and has one object attribute *type* with one value *General* assigned to it. The senior groups inherits this object attribute and its value along with other values directly assigned to them. For example, *Networking_Project* inherits object attribute *type* and its value *General*, and also has another value assigned to it as *Networking*. Thus, the effective attribute values of *Networking_Project* for attribute *type* is a set of all the values, directly assigned and inherited through the hierarchy, written as {*General, Networking*}.

In addition to two user attributes, namely *depart* and *skills*, there is one more user attribute, *title*, with range {*CTO, IT_Manager, DevOps_Manager*}. We consider this user attribute to be directly assigned to the users since titles are individually assigned to user rather than being assigned to a group of users. There is only one object attribute, *type* assigned to the object groups. Any object in an object group automatically gets all the attributes and its values assigned to it through that group. Similarly, users gets user attributes and values assigned directly or through user groups.

A graph similar to a PM graph [18] is shown in Figure 9 for this use case. A policy named GHP (Group Hierarchy Policy) is shown at level 0, user attributes and object attributes are shown at level 1. User attributes, their values, and user groups are on the left-hand side of the policy and object attributes, their values, and object groups on the right-hand side of the policy. User and object attribute values are placed at level 2 and theses entities are assigned to each other using *assignment* relation. *Assignments* in the graph are represented as directed edge, for example *x ASSIGN y* is represented as a directed edge from x to y.

User and Object groups are present at level 3 in this use case. If there is hierarchy among groups then senior groups are shown one level above their junior groups, with consecutive increments in level for each level of hierarchy among groups. *DevOps_Group* is a junior group and is shown one level below the senior groups, *Dev_Group* and *Depl_Group*. The same applies on the object side where *Projects_Group* is shown one level below other senior object groups. The hierarchical relations among groups are achieved through the *containment* property of PM attributes.

User and objects are the leaves of the graph and could appear at any level in the graph based on their assignment to user groups, user attribute values and object groups, object attribute values respectively. A policy in the PM is defined using *associations* which comprise of a user attribute value, an operation and an object attribute value written as (ua_i, op, oa_j). To keep the graph simple in Figure 9, we show only one association between *IT_Manager* (a user attribute value for *title*) and *Networking* (an object attribute value for *type*) labeled as *read* using a dashed line. This association allows *user_IT1* to read *obj_Net1*. Therefore, all

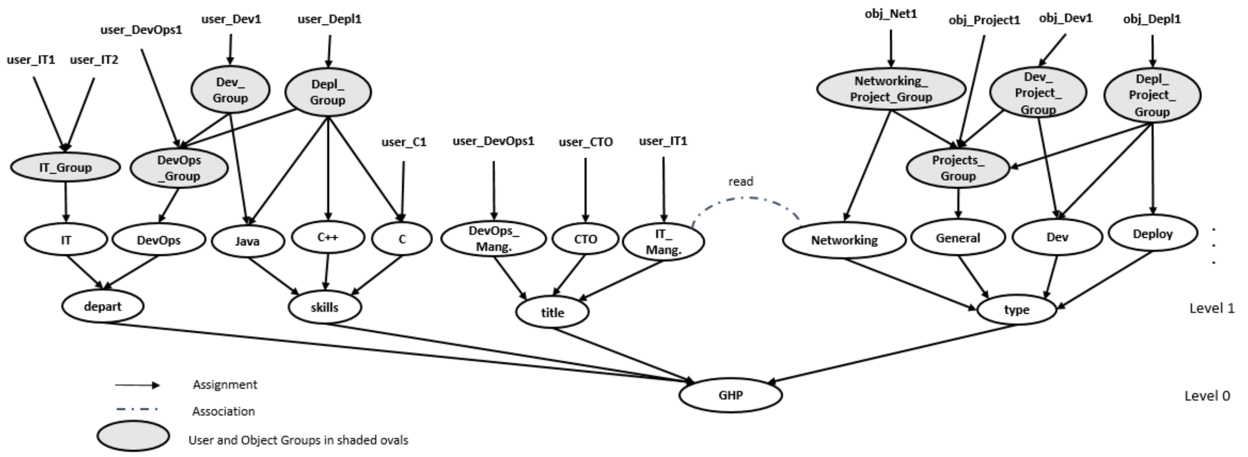


Figure 9: Group Hierarchy Policy Graph (Based on PM Graph Structure)

Table 3: Policy for *Read* Operation with Group Hierarchy

$Policy_{read}$	
User Attribute Values	Object Attribute Values
IT_Manager	Networking
IT	Networking
DevOps_Manager	Dev
Java	Dev
DevOps_Manager	Deploy
Java	Deploy
C	Deploy
C++	Deploy
CTO	General

the users having title *IT_Manager* can read all the objects of type *Networking*.

In this use case, we define an access control policy for *read* operation. Policies are defined based only on user attribute values and object attribute values. For each policy tuple in $Policy_{read}$, only a single value of user attribute and a single value of object attribute is allowed to be used in a single-value EAP. A policy for read operation, based on Figure 9, is specified as follows.

i. A user who is an IT_Manager or works in the IT department can read objects of type Networking.

ii. A user who is a DevOps_Manager or has Java skill can read objects of type Dev.

iii. A user who is a DevOps_Manager or has Java or C or C++ skill can read objects of type Deploy.

iv. A user who is a CTO can read objects of type General.

The fourth policy statement incorporates powerful policy, where a CTO can actually read all the objects in the above scenario due to the attribute inheritance achieved through object group hierarchy. All the groups have type General assigned to them through object group hierarchy and attribute inheritance, thus any object assigned to any one of these groups can be read by CTO. Therefore, many implied policies can be derived via one policy tuple.

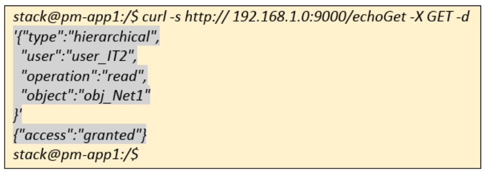

```
stack@pm-app1:/$ curl -s http:// 192.168.1.0:9000/echoGet -X GET -d
'{"type":"hierarchical",
  "user":"user_IT2",
  "operation":"read",
  "object":"obj_Net1"
}'
{"access":"granted"}
stack@pm-app1:/$
```

Figure 10: Sample Authorization Request and Response

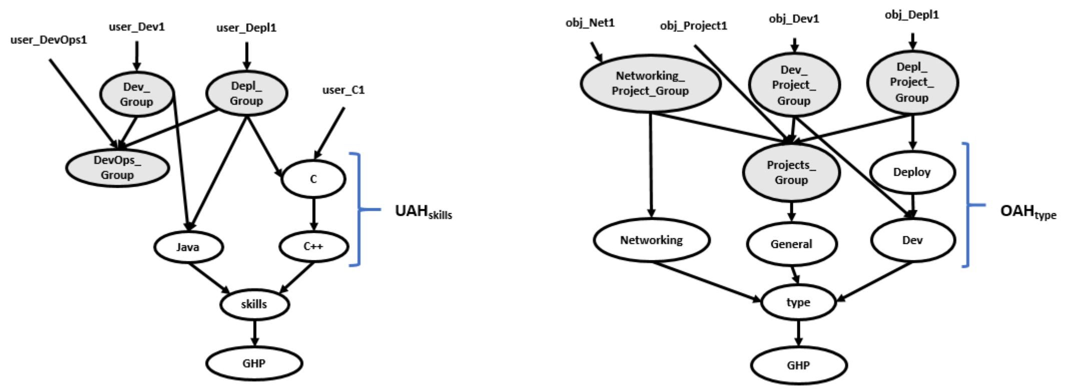
A set of policy tuples for $Policy_{read}$ as per above specifications is presented in Table 3. Each row of the table represents an association defined in the PM. An authorization request involving user *user_IT2*, operation *read* and object *obj_Net1*, along with its response is shown in Figure 10. This request is granted based on second row of Table 3.

The importance and benefits of group attributes and hierarchy can be realized when there are numerous users and objects in the system and attributes need to be assigned to each one of them. In this use case, for simplicity, we presented a limited number of users and objects, however in a real-world scenario there are hundreds, thousands, or even millions of users and objects. In such a scenario, *rHGABAC* could be an effective solution for access control requirements with better policy administration and attribute management.

5.2 Attribute Hierarchies

In enumerated policies, an inherent problem is the exponential size of a complex policy and its associated space requirements. The number of policy tuples in a policy, for an operation, would increase exponentially with large ranges of user attributes and object attributes. A policy consists of policies defined for different operations (*read, write, execute, etc.*) in a system which further worsens the problem. An interesting solution to this problem could be to realize hierarchical relations among the values of user attributes and object attributes.

With attribute hierarchy, a single policy tuple defined in the policy can imply many policy tuples through partial order relations among user and object attribute values. In this subsection, we introduce attribute hierarchies among ranges of one user attribute (*skills*) and one object attribute (*type*)



(a). Subgraph Showing Attribute Hierarchy in *skills* Attribute (b). Subgraph Showing Attribute Hierarchy in *type* Attribute

Figure 11: Attribute Hierarchy

Table 4: Policy for *Read* Operation with Group and Attribute Hierarchy

$Policy_{read}$	
User Attribute Values	Object Attribute Values
IT_Manager	Networking
IT	Networking
DevOps_Manager	Dev
Java	Dev
C++	Deploy
CTO	General

from above use case. The subgraph for each *skills* and *type* are shown in Figure 11(a) and 11(b) respectively. In the use case graph of Figure 9, *skills* has all three values assigned to it at the same level. However, in subgraph of Figure 11(a) C is assigned to $C{+}{+}$ and $C \succeq_{skills} C++$. This means that attribute value C is senior to $C{+}{+}$ and gets all the properties and capabilities of $C{+}{+}$. Therefore, any accesses allowed for $C{+}{+}$ would be allowed for C as well.

In Figure 11(b), the *type* attribute is shown with a partial order hierarchy between its values *Deploy* and *Dev*, written as $Deploy \succeq_{type} Dev$ where *Deploy* is senior to *Dev*. Any *association* involving junior attribute value will also be implied on senior attribute value. In other words, *Deploy* will inherit all the associations applied on *Dev*.

Based on attribute hierarchies introduced in *skills* and *type*, the *associations* defined in PM are changed. The updated policy tuples for$Policy_{read}$ are shown in Table 4. Three policy tuples were removed since they were implied through attribute hierarchy. Two tuples *(DevOps_Manager, Deploy)* and *(Java, Deploy)* were removed since they were implied through hierarchy between *Deploy* and *Dev*. Similarly, *(C, Deploy)* is removed as its implied through hierarchy between *C* and *C++*. This clearly shows how hierarchical attributes can be exploited in reducing the size of enumerated policies. It also contributes in facilitating administrative tasks by reducing the number of direct policy tuples to be defined in a policy.

An authorization request for user *user_C1* (assigned to *C*), operation *read* and object *obj_Depl1* (assigned to *Deploy*), along with its response is shown in Figure 12. Though, there is no direct policy tuple defined between *C* and *Deploy* attribute values, this request is granted through an implied policy tuple based on effective user attribute values of the user and effective object attribute values of the object.

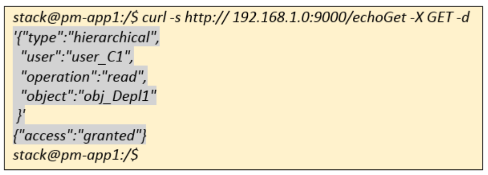

```
stack@pm-app1:/$ curl -s http:// 192.168.1.0:9000/echoGet -X GET -d
'{"type":"hierarchical",
 "user":"user_C1",
 "operation":"read",
 "object":"obj_Depl1"
}'
{"access":"granted"}
stack@pm-app1:/$
```

Figure 12: Sample Authorization Request and Response

The partial order relations among groups and attributes depend on how they are realized while defining security policies. Some hierarchical ordering are intuitive, whereas others may not be so evident. It is for security architects and administrators to design these hierarchies to define access control policies in most efficient way.

6. POLICY EVALUATIONS IN PM

In this section, we compare the policy evaluation times for different types of attribute-based access control policies in AE and PM. When a user requests access to an object in an application, an authorization request is received by AE which in turn communicates to the PM to get authorization decisions. For each type of policy, the time taken to communicate to the PM and evaluate authorization decisions was measured. The average policy evaluation time for a *Role-Centric ABAC*, and two variations of *rHGABAC* without and with attribute hierarchy is presented in Table 5. The average policy evaluation time using AE and the PM is shown in second column, while the type of policies are shown in the first column.

For the experiments, we first setup each type of policy in the PM and then execute multiple authorization requests for each one of them to obtain averages of their policy evaluation times. The average policy evaluation time for above policies are very close to each other with a maximum difference of 1 ms. The results were in compliance with our initial hypothesis, since we expected the average policy evaluation

Table 5: Average Policy Evaluation Time for ABAC Policies

Policy	Avg. Time (ms)
Role-Centric ABAC	26.04
rHGABAC	27.04
rHGABAC with AH	26.57

times to be similar, mainly due to the way ABAC policies are implemented in the PM.

The average policy evaluation time for *rHGABAC with AH* is slightly less than *rHGABAC*. In *rHGABAC with AH*, we can represent complex and large policies in a concise way realizing hierarchies among attribute values. It is evident in our use case discussed in previous section as well. A concise policy in the PM implies a compact PM policy graph with less associations between entities or elements. Both *rHGABAC* and *rHGABAC with AH* policy defined in the PM has same set of users, user groups, user attributes, objects, object groups, object attributes, however the number of association relations (policy tuples) varies which results in a disperse policy tree for *rHGABAC* compared to a concise or small policy tree for *rHGABAC with AH*. We believe this is one of the possible reasons for faster policy evaluation time in *rHGABAC with AH* policy, since small number of association relations defined in the PM would involve less internal evaluations to get a policy decision.

A role-centric ABAC is a special type of attribute-based access control policy which includes roles and attributes in the policy specifications and policy decisions. Although, this policy had the lowest average of policy evaluation time in our experiment, the time would increase exponentially if there are numerous roles that a user can be assigned. It is a combination of role-based and attribute-based access control policy and would require a more detailed investigation with similar types of combinational access control models. However, our main focus in this paper are ABAC policies.

Overall, our initial investigation shows that the average policy evaluation times are comparable across different types of ABAC models, and *rHGABAC* model can be easily implemented utilizing the PM and can be applied in real-world applications and systems. However, the enforcement framework in the applications utilizing the PM need to be designed accordingly in order to optimize policy evaluation and policy enforcement times.

7. CONCLUSION

In this paper, we presented and formalized a restricted HGABAC model as single-value EAP, named as *rHGABAC*, with group attributes and group hierarchies. We then extended this model with attribute hierarchies, i.e. partial order relations among values of user and object attributes. We implemented use cases of these models utilizing the PM, and presented an initial investigation of the policy evaluation times for different types of attribute-based access control policies in the PM using AE. We also presented a general authorization architecture that can be conveniently adapted by any application or system in order to implement different types of ABAC policies utilizing the PM.

Our approach of employing containment property of PM attributes via *assignment* relations gracefully incorporates groups attributes and group and attribute hierarchies in *rHGABAC* model. Further exploration of newer version of PM tool and its interfaces might be beneficial to find more efficient ways to represent ABAC models for its wide adoption in the industry.

8. ACKNOWLEDGMENTS

The authors would like to thank NIST and the Policy Machine team, especially Dr.David Ferraiolo, Serban I. Gavrila, and Gopi Katwala for their guidance and support. This research is partially supported by NSF Grants CNS-1111925 and CNS-1423481, and DoD ARL Grant W911NF-15-1-0518.

9. REFERENCES

[1] V. C. Hu, D. R. Kuhn, and D. F. Ferraiolo, "Attribute-based access control." *IEEE Computer*, vol. 48, no. 2, pp. 85–88, 2015.

[2] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, "Guide to attribute based access control (ABAC) definition and considerations," *NIST Special Publication 800-162*, 2014.

[3] X. Jin, R. Krishnan, and R. Sandhu, "A unified attribute-based access control model covering DAC, MAC and RBAC," in *IFIP Annual Conference on Data and Applications Security and Privacy.* Springer, 2012, pp. 41–55.

[4] D. Servos and S. L. Osborn, "HGABAC: Towards a formal model of hierarchical attribute-based access control," in *International Symposium on Foundations and Practice of Security.* Springer, 2014, pp. 187–204.

[5] P. Biswas, R. Sandhu, and R. Krishnan, "Label-based access control: An ABAC model with enumerated authorization policy," in *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control.* ACM, 2016, pp. 1–12.

[6] J. Hur and D. K. Noh, "Attribute-based access control with efficient revocation in data outsourcing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 7, pp. 1214–1221, 2011.

[7] T. Priebe, W. Dobmeier, and N. Kamprath, "Supporting attribute-based access control with ontologies," in *First International Conference on Availability, Reliability and Security (ARES'06).* IEEE, 2006, pp. 8–pp.

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

[9] E. Yuan and J. Tong, "Attributed based access control (ABAC) for web services," in *IEEE International Conference on Web Services (ICWS'05).* IEEE, 2005.

[10] S. Bhatt, F. Patwa, and R. Sandhu, "An attribute-based access control extension for openstack and its enforcement utilizing the policy machine," in *IEEE 2nd International Conference on Collaboration and Internet Computing (CIC).* IEEE, 2016, pp. 37–45.

[11] R. S. Sandhu and P. Samarati, "Access control: principle and practice," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 40–48, 1994.

[12] R. Sandhu, "Role-based access control," *Advances in Computers*, vol. 46, pp. 237–286, 1998.

[13] 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)*, vol. 4, no. 3, pp. 224–274, 2001.

[14] R. Sandhu, E. J. Coyne, H. Feinstein, and C. Youman, “Role-based access control models,” *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[15] Q. M. Rajpoot, C. D. Jensen, and R. Krishnan, “Integrating attributes into role-based access control,” in *IFIP Annual Conference on Data and Applications Security and Privacy.* Springer, 2015, pp. 242–249.

[16] “XACML.” [Online]. Available: https://en.wikipedia.org/wiki/XACML

[17] “Policy Machine.” [Online]. Available: http://csrc.nist.gov/pm/

[18] D. Ferraiolo, S. Gavrila, and W. Jansen, “Policy Machine: Features, architecture, and specification,” *National Institute of Standards and Technology Internal Report 7987*, 2014.

[19] D. Ferraiolo, V. Atluri, and S. Gavrila, “The Policy Machine: A novel architecture and framework for access control policy specification and enforcement,” *J. of Sys. Architecture*, vol. 57, no. 4, pp. 412–424, 2011.

[20] J. Li, Q. Wang, C. Wang, and K. Ren, “Enhancing attribute-based encryption with attribute hierarchy,” *Mobile Networks and Applications*, vol. 16, no. 5, pp. 553–561, 2011.

[21] P. Biswas, R. Sandhu, and R. Krishnan, “A comparison of logical-formula and enumerated authorization policy abac models,” in *IFIP Annual Conference on Data and Applications Security and Privacy.* Springer, 2016, pp. 122–129.

[22] D. R. Kuhn, E. J. Coyne, and T. R. Weil, “Adding attributes to role-based access control,” *IEEE Computer*, vol. 43, no. 6, pp. 79–81, 2010.

[23] O. Sefraoui, M. Aissaoui, and M. Eleuldj, “OpenStack: toward an open-source solution for cloud computing,” *International Journal of Computer Applications*, vol. 55, no. 3, 2012.

[24] 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*, vol. 7, no. 2, pp. 169–180, 2009.

[25] 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.* ACM, 2004, pp. 45–55.

[26] W. Kuijper and V. Ermolaev, “Sorting out role based access control,” in *Proceedings of the 19th ACM Symposium on Access Control Models and Technologies.* ACM, 2014, pp. 63–74.

[27] D. Ferraiolo, R. Chandramouli, V. Hu, and R. Kuhn, “A comparison of attribute based access control (ABAC) standards for data service applications,” *NIST Special Publication 800-178*, 2016.

[28] “Axiomatics - XACML.” [Online]. Available: https://www.axiomatics.com/attribute-based-access-control.html

[29] “Exploring the next generation of access control methodologies.” [Online]. Available: http://ws680.nist.gov/publication/get_pdf.cfm?pub_id=922332

[30] M. Gupta and R. Sandhu, “The GURA_G administrative model for user and group attribute assignment,” in *International Conference on Network and System Security.* Springer, 2016, pp. 318–332.