Composing and Combining Policies under the Policy Machine

David F. Ferraiolo, Serban Gavrila*, Vincent Hu, D. Richard Kuhn
National Institute of Standards and Technology
Gaithersburg, MD 20899
+1 301-975-3046
dferraiolo@nist.gov, serban.gavrila@nist.gov, vhu@nist.gov, kuhn@nist.gov

ABSTRACT

As a major component of any host, or network operating system, access control mechanisms come in a wide variety of forms, each with their individual attributes, functions, methods for configuring policy, and a tight coupling to a class of policies. To afford generalized protection, NIST has initiated a project in pursuit of a standardized access control mechanism, referred to as the Policy Machine (PM) that requires changes only in its configuration in the enforcement of arbitrary and organization specific attribute-based access control policies. Included among the PM's enforceable policies are combinations of policy instances (e.g., Role-Based Access Control and Multi-Level Security). In our effort to devise a generic access control mechanism, we construct the PM in terms of what we believe to be abstractions, properties and functions that are fundamental to policy configuration and enforcement. In its protection of objects under one or more policy instances, the PM categorizes users and objects and their attributes into policy classes, and transparently enforces these policies through a series of fixed PM functions, that are invoked in response to user or subject (process) access requests.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection –*Access controls, Information flow controls.* C.2.0 [**Computer-communication Networks**]: General – *Security and protection.*

General Terms

Security, Standardization, Theory.

Keywords

Access control, role based access control, separation of duty, multi-level security.

1. INTRODUCTION

Access control is the administrative and automated process of defining and limiting which system users can perform which system operations on which system resources. Pertaining to each organization is a unique set of policies that dictate the circumstances and conditions under which specific users are permitted access to specific resources. Access control policies are enforced through a mechanism consisting of access control functions and access control data that together map a user's access request to a decision whether to grant or deny access. The ability of an organization to enforce its access policies directly impacts its ability to execute its mission – by determining the degree in which its volumes of data may be protected and shared among its user community. Whether in regard to the Government's war on terror or a company's formation of a strategic partnership, the focus on sharing and protecting information is becoming increasingly acute [1]. Unfortunately, when it comes to access control mechanisms one size does not fit all. Access control mechanisms come in a wide variety of forms, each with their individual (and often proprietary) attributes, functions, and methods for configuring policy, and a tight coupling to a class of policies.

* contracted from VDG, Inc. Chevy Chase, MD

For instance, in response to the need to protect classified information, there are mechanisms to enforce Multi-level Security (MLS) policies and mechanisms to enforce need-to-know policies, and in recognition of the needs of industry, Role-based Access Control (RBAC) mechanisms enforce commercial policies. While these and other mechanisms may meet broad policy requirements within their respective user domains, there is also a need to address specific and often ad hoc organization requirements. These requirements may, for example, pertain to controlling access based on: a user's membership within an organizational entity; the inclusion of a resource within a geographical region, or facility; the relative importance of data (e.g., ordinary, important, critical); or even something as esoteric as a user's affiliation to a political party. In addition, organizational policies can and often do pertain to combinations of two or more policies. For example, to gain access to a classified medical record may require the enforcement of an MLS policy (to prevent direct and indirect compromise of classified data), the enforcement of an RBAC policy (to ensure the user is qualified), and the enforcement of an Identity-based Access Control (IBAC) policy (to protect patient privacy).

Towards affording generalized protection, NIST, under the request and support of the Department of Homeland Security (DHS), has initiated a project in pursuit of a standardized attribute-based access control mechanism (excluding for the time being environmental factors such as time and location), referred to as the Policy Machine (PM). Core features, and the subject of this paper, is the PM's ability to configure and enforce arbitrary attribute-based access control policies and its ability to protect resources under multiple instances of these policies. (This paper assumes that readers have a basic familiarity with common access control models and policies). Other PM features not included in this paper pertain to the configuration and enforcement of safety invariants, static separation of duty, and multi-state policies (also referred to as history-based policies). Although in some respects the PM is similar to existing models and mechanisms, and the techniques deployed could afford improvements to those technologies, in our effort to devise a generic access control mechanism, we have resisted the temptation of extending any model or mechanism. At the same time it is not our intent to devise a completely new model, but instead, we attempt to redefine access control in terms of what we believe to be elements, abstractions, properties and functions that are fundamental to the configuration and enforcement of attribute-based policies in general. In building from these primitives, we believe that the PM is generic in its support of well-documented models, while also accommodating the need for ad hoc attribute-based policy requirements. Considered among these requirements are policy combinations. In its protection of objects (e.g., files) under one or more policy instance, the PM includes a capability to categorize user and object attributes into their respective classes of policies, and appropriately enforcing subsets of these policies in response to each user and subject (process) access request.

We define the PM as a mechanism rather then a model, by making its functions and functional interfaces explicit. At the same time the PM is also abstract and general. It is abstract because properties not relevant to access control are not included; and it is general because many architectural and implementation choices could be valid interpretations of the prescribed mechanism.

The remainder of this paper is organized as follows. In the following section we describe access control's basic and salient elements, relationships, properties, and functions that are conformed to in the formulation the PM's relations and functions as described in subsequent sections. Section 3 describes the PM method for instantiating arbitrary user and object attributes. In section 4 we begin to describe the PM's basic functions in terms of the abstractions and relations presented in prior sections. Section 5 demonstrates by example the PM's ability to configure three different classes of policies. Section 6 extends the PM's functionality to include the dynamic and constrained activation of subject attributes, and section 7 describes and demonstrates the PM's ability to specify and enforce policy combinations over user and subject access requests. Section 8 contrasts the PM to related models, and mechanisms, and section 9 concludes the paper.

2. ACCESS CONTROL BASICS

Classically access control models and mechanisms are defined in terms of authorized users (U), subjects (S), system operations (Op), and named objects (O). Authorized users are humans, each with a one-to-one mapping (as a consequence of authentication) to a unique user identifier (typically established at user account creation time). An authorized user (hereafter, simply referred to as a user) is unable to execute access requests directly, but instead must submit a request through a subject—a system process (with memory) that operates on behalf of the user. Objects are names (with global meaning) given to system entities that must be protected, and perhaps shared, under one or more policies. The set of objects may pertain to processes, files, or exhaustible system resources such as printers. The selection of entities included in this set is a matter of choice determined by the protection requirements of the system. Essential properties of subjects are that they have exclusive access to their own memory and none to any others, they have potentially different access to objects than other subjects, and that they are semi-autonomous. In addition to issuing user requests, a subject may maliciously (through a Trojan horse), or by system error, issue requests that are independent of, and perhaps unknown to its user.

In associating subjects and users, we denote by *subj_user*(s) the user, $u \in U$, associated with subject $s \in S$. In distinguishing their requests we denote by $<ops, o>_u$ a legal user access request, and by $<op, o>_s$ a subject access request, where $u \in U$, $op \in Op$, $ops \subseteq Op$, $o \in O$, and $s \in S$. Note that a user's request may include more then one operation. For example, a request to "open" a document is often interpreted (in many operating systems) to mean "open for read and write." On the other hand, subject requests are issued serially. In response to an "open" request a subject may read a document and present an image of the document to its user. After modifying the image, the user may issue a request to "save" the document (interpreted as write) to his subject.

Included in the access control data of a mechanism is a set of permissions P and included among its functions is a reference mediation function [2]. In a very primitive form permissions may be represented as a list of triples of the form (u, op, o) indicating that a user $u \in U$ is potentially able to perform operation $op \in Op$ on the contents of object $o \in O$ [3]. Associated (perhaps by virtue of a search routine) with each user is a set of capabilities $\{(op, o) \in Op \times O\}$, and with each object is a set of access control entries $\{(u, op) \in U \times Op\}$[2]. Although permissions are defined in terms of users, the reference mediation function controls access in terms of subjects. Ultimately, a subject's request, $<op, o>_s$ is granted by the reference mediation function, if there exists a permission (u, op, o)$\in P$, where u is the subject's user, and (op, o) is a capability of u, otherwise the request is denied.

Regarding practical mechanisms, permissions are not individually managed, but instead permissions are organized in terms of, and derived from, a set of policy specific user and object attributes, providing a strategy for organizing, managing and reviewing permission data, and controlling the access requests of subjects. The attributes of users and objects are established through administrative or system assignments. Regarding their mappings to permissions, user attributes in one form or another are associated (by rules or administrative assignments) with a defined set of capabilities. Thereby assigning a user to an attribute indirectly associates the user with the capabilities of the attribute.

In addition to an access control mechanism's reference mediation function are two other basic functions—a function to create subjects and associate these subjects with their users, and a function to associate a subject with a subset of attributes that are assigned to its user. Regardless of its implementation, and the type of attributes that are deployed, reference mediation effectively constrains the successful execution of subject and user requests to the capabilities that are associated with a subject's attributes. Although a number of access control mechanisms associate a subject with each and every attribute of its user, in order for an access control mechanism to support the principle of least privilege [4], as well as the properties of a variety of access control models (e.g., one-directional information flow to defeat Trojan horse attacks), constraints must be placed on the attributes that may be associated with a subject. While in the absence of these constraints, reference mediation limits the execution of a subject request to the space of capabilities associated with its user, subject attribute constraints further reduce this space of permissible requests to a policy preserving and potentially small subset of the capabilities of its user.

3. ABSTRACTING USER AND OBJECT ATTRIBUTES

A principal benefit of the PM is its ability to abstract any attribute under the observation that user attributes are effectively mappings of defined sets of users to defined sets of capabilities and object attributes are effectively mappings of defined sets of objects to defined sets of access control entries. Also it is often the case that the users/objects that are associated with an attribute$_1$ are (inherit) a subset of the users/objects that are associated with some attribute$_2$, while the capabilities/access control entries of the attribute$_2$ are (inherit) a subset of the capabilities/access control entries of attribute$_1$.

Under the standard RBAC model [5], roles are associated with capabilities through administrative assignments. For instance, Nurses may be assigned the capabilities to append new entries to a patient's history of treatments. Although RBAC does not formally recognize object attributes and hierarchies, their existence offers dual benefits as those of user-role assignments. For instance, the role Doctor might be assigned the capabilities to read all objects currently assigned to the Medical Records attribute. Assigning a new object *o* to Medical Records potentially enables a large number of users (in the Doctor role, and potentially other roles) access to object *o*. Under the Bell & LaPadula model [6], users and objects are associated with attributes by administrative or system assignment and capabilities/access control entries are associated with attributes by virtue of a set of rules. In consideration of these rules, for example, the Secret user attribute is associated with the capabilities to *read* the set of Unclassified, Confidential and Secret objects and *write* to the set of Secret and Top Secret objects. Associated with each object attribute are access control entries. For instance, the set of objects that are classified at the Secret level can be *read* by the set of users that are cleared to the Secret or Top Secret level.

In consideration of these observations, we define the sets of user attributes (*UA*), and object attributes (*OA*), with mappings to permissions expressed in terms of users, objects, operations, and assignment relations denoted by "$\rightarrow$". Note that we consider an object (name) as an object attribute, i.e., $O \subseteq OA$.

We define the *user-to-attribute assignment* as a binary relation on $U \times UA$, denoted by "$\rightarrow$". Intuitively, for a user $u \in U$ and an attribute $ua \in UA$, $u \rightarrow ua$ would mean that user *u* has the properties denoted by the attribute *ua*. On the set of *UA* we define the function $users{:}UA \rightarrow 2^U$ that associates each attribute with the set of users possessing that attribute.

In addition, we define *attribute assignment* as a binary relation on the set *UA*, also denoted by "$\rightarrow$"($\rightarrow \subseteq UA \times UA$). Note that we overload the symbol "$\rightarrow$" without any danger of confusion; its meaning will be clear from the type of its operands.

The user-to-attribute assignment, the function *users*, and the attribute assignment must satisfy the following properties:

1. $\forall u \in U, \forall ua \in UA, u \rightarrow ua \Rightarrow u \in users(ua)$. Note that the reverse implication is not necessarily true.
2. $\forall ua \in UA, \neg(ua \rightarrow^+ ua)$: the attribute assignment relation has no cycles.
3. $\forall ua_1 \in UA, \forall ua_2 \in UA, ua_1 \rightarrow ua_2 \Rightarrow users(ua_1) \subseteq users(ua_2)$.
4. $\forall u \in U, \forall ua \in UA, u \in users(ua) \Rightarrow \exists ua_0 \in UA: u \rightarrow ua_0 \rightarrow^* ua$.

(We used $^+$, respectively * to denote the transitive, respectively transitive and reflexive closure of a binary relation).

On the set *OA* we define a function $objects{:}OA \rightarrow 2^O$. Intuitively, *objects*(*oa*) is the set of objects possessing the attribute *oa*. We also define the *attribute assignment* as a binary relation on *OA* denoted by "$\rightarrow$" ($\rightarrow \subseteq OA \times OA$). The function *objects* and the attribute assignment must satisfy the following properties:

1. $\forall o \in O \subseteq OA, objects(o) = \{o\}$.
2. $\forall oa \in OA, \neg(oa \rightarrow^+ oa)$.
3. $\forall oa_1 \in OA, \forall oa_2 \in OA, oa_1 \rightarrow oa_2 \Rightarrow objects(oa_1) \subseteq objects(oa_2)$.
4. $\forall o \in O, \forall oa \in OA, o \in objects(oa) \Rightarrow o \rightarrow^* oa$.

We define the assignment between user attributes and operation sets as a binary relation on $UA \times 2^{Op}$, denoted by "$\rightarrow$". Similarly, we define the assignment between operation sets and object attributes as a binary relation on $2^{Op} \times OA$, also denoted by "$\rightarrow$". As we will see below, these two assignment relations indirectly specify the set of access control entries (*u*, *op*) for an object attribute *oa* assigned to an operation set *ops* such that $op \in ops$; and the set of capabilities (*op*, *o*) for a user attribute *ua* assigned to an operation set *ops* such that $op \in ops$.

Let us define the *access control entries* of an object attribute *oa* as the access control entries derived from operation sets assigned to that object attribute. Formally:

$$aces(oa) = \{(u, op) \mid \exists\, ua \in UA, \exists\, ops \in Ops: u \in users(ua) \wedge op \in ops \wedge ua \rightarrow ops \wedge ops \rightarrow oa\}$$

Let us define the *capabilities* of a user attribute *ua* as the capabilities derived from that user attribute's assignments to operation sets. Formally,

$$caps(ua) = \{(op, o) \mid \exists\, ops \in Ops, \exists\, oa \in OA: op \in ops \wedge o \in objects(oa) \wedge ua \rightarrow ops \wedge ops \rightarrow oa\}$$

Depending on one's perspective, the set of permissions *P* that exist in the PM system may be represented and reviewed either in terms of assignments between user attributes and capability sets $ua \rightarrow caps(ua)$ forming *capability lists*, or in terms of assignments between object attributes and access control entry sets $oa \rightarrow aces(oa)$ forming access control lists, as the following relation shows:

$P = \{(u, op, o) \mid u \in U, op \in Op, o \in O, \exists ua \in UA, \exists ua_1 \in UA: (op, o) \in caps(ua) \wedge u \rightarrow ua_1 \wedge ua_1 \rightarrow^* ua\}$

$= \{(u, op, o) \mid u \in U, op \in Op, o \in O, \exists oa \in OA, (u, op) \in aces(oa) \wedge o \rightarrow^{*} oa\}$.

4. PM BASIC FUNCTIONS

With respect to our uniform representation of attributes we begin to define PM access control functions that are consistent with those described in section 2.

Let us first define the function $attrs{:}S \rightarrow 2^{UA}$ that associates a subject with its set of attributes, with the property that any subject attribute is an attribute of the subject's user. Formally,

$$\forall s \in S,\ attrs(s) \subseteq \{ua \in UA \mid subj\_user(s) \in users(ua)\}$$

Reference mediation: The PM applies subject attributes in restricting the space of permissible subject access requests. The reference mediation function grants the subject *s* the permission to execute a request $<op, o>_s$ if and only if the pair (op, o) is a capability of an attribute *ua* of subject *s*. Formally,

$$\forall s \in S,\ \forall op \in Op,\ \forall o \in O:$$

$$reference\_mediation(<op, o>_s) = grant \Leftrightarrow \exists ua \in attrs(\mathrm{s}){:}\ (op, o) \in caps(ua)$$

Although a user may ultimately be authorized for a wide range of capabilities, we impose subject attribute constraints to further limit the capabilities that can be exercised by a subject to a policy preserving subset of the user's capabilities.

Subject attribute constraints: A subject attribute constraint is a list of disjoint user attribute sets: $sac = (uas_1, \ldots, uas_n)$, where $n \in \mathbb{N}_1 \wedge (\forall i \in 1..n,\ uas_i \subseteq UA) \wedge (\forall i, j \in 1..n,\ \mathrm{i} \neq \mathrm{j} \Rightarrow uas_i \cap uas_j = \varnothing)$.

We denote by *SAC* the set of subject attribute constraints. For example, *SAC* = {({Doctor, Intern}, {Consultant}), ({L}, {M}, {H})} contains two constraints; the first constraint is a list of two user attribute sets {Doctor, Intern} and {Consultant}, the second constraint is a list of three attribute sets, {L}, {M}, {H}. Intuitively, if a subject $s \in S$, has an attribute $ua_1 \in attrs(s)$ belonging to one user attribute set of a subject attribute constraint relation, then subject *s* cannot have a second attribute $ua_2 \in attrs(s)$ belonging to any other user attribute set of the same subject attribute constraint relation.

For an arbitrary attribute set $uas \subseteq UA$, let *attribute_set_valid*(*uas*) be the predicate specifying that *uas* does not include two or more attributes from different attribute sets of any subject attribute constraint:

$$attribute\_set\_valid(uas) \overset{def}{\Longleftrightarrow} \forall sac \in SAC,\ sac = (uas_1, \ldots, uas_n),$$
$$\forall i, j \in 1..n,\ i \neq j,\ uas \cap uas_i \neq \varnothing \Rightarrow uas \cap uas_j = \varnothing.$$

Now we can define the global predicate *subject_attributes_valid* as follows:

$$subject\_attributes\_valid \overset{def}{\Longleftrightarrow} \forall s \in S,\ attribute\_set\_valid(attrs(s)).$$

For example, if ({L}, {M}, {H})$\in SAC$, and $\mathrm{M} \in attrs(s)$, then for *subject_attribute_valid* to be true $\mathrm{L} \notin attrs(s)$, $\mathrm{H} \notin attrs(s)$.

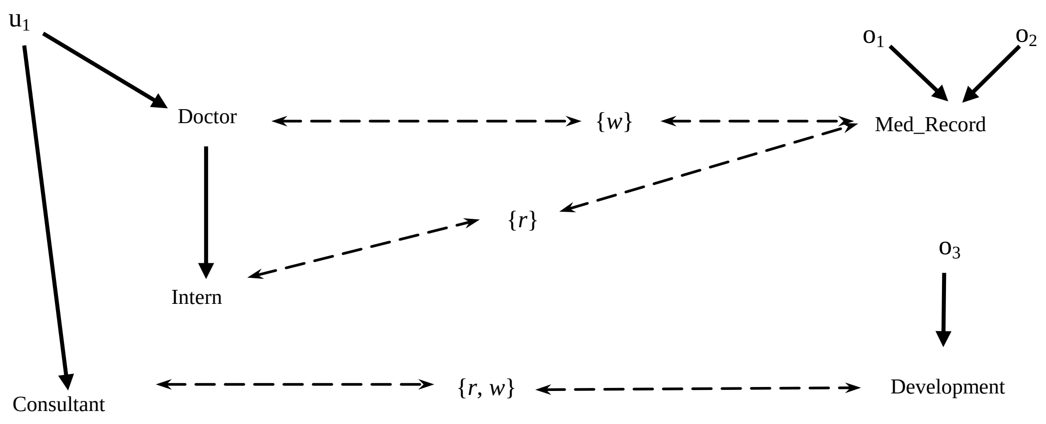
5. EXAMPLE PM POLICY EMBODIMENTS

Assuming the PM implements the reference mediation function and subjects adhere to subject attribute constraints, as defined in the preceding section, we are able to emulate a number of different access control models, each affording its own class of policies. In this section, we demonstrate by example three different classes of policies (RBAC, MLS, and IBAC); each achieved through different configurations of assignment and subject constraint relations. In demonstrating the PM's ability to abstract attributes consider the assignment relations depicted in figure 1, where the assignments between users and user attributes, objects and object attributes, and attributes and attributes of the same kind, are represented by the darker arrows, and assignment relations between user attributes and operation sets, and operation sets and object attributes are represented with lighter-dashed arrows.

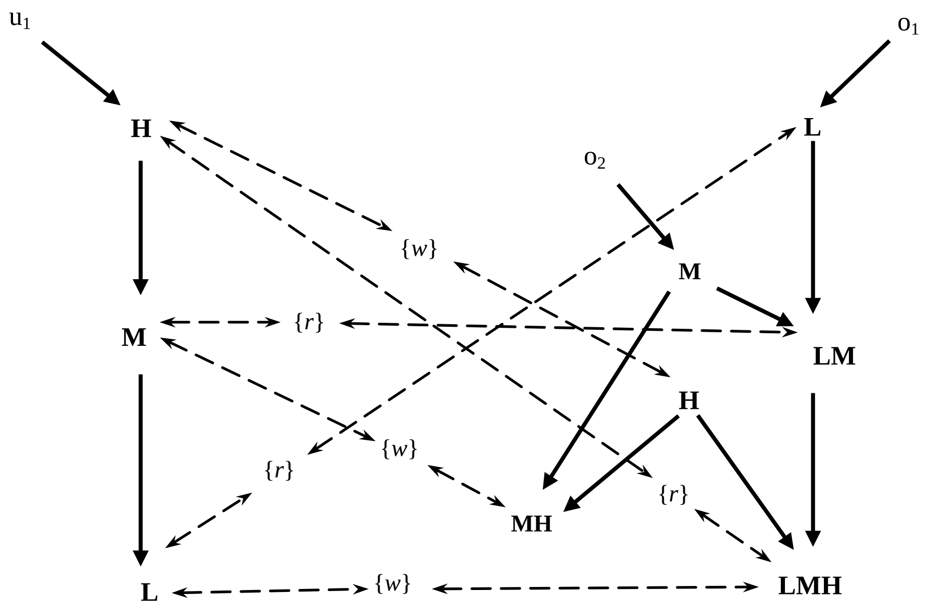
Figure 1(a) depicts an RBAC policy where Doctor and Intern are user attributes and o_1, o_2, o_3, Med_Record, and Development are object attributes. The *caps*(Doctor) = $\{(w, o_1), (w, o_2)\}$ which are derived from the Doctor$\rightarrow\{w\}$, $\{w\}\rightarrow$Med_Record assignment relations. The *caps*(Intern) = $\{(r, o_1), (r, o_2)\}$ which are derived from the Intern$\rightarrow\{r\}$, $\{r\}\rightarrow$Med_Record assignment relations. The *caps*(Consultant) = $\{(r, o_3), (w, o_3)\}$ which are derived from the Consultant$\rightarrow\{r, w\}$, $\{r, w\}\rightarrow$Development assignment relations.

In addition to these assignment relations assume the following subject attribute constraint relation ({Doctor, Intern}, {Consultant})$\in SAC$, in support of a least privilege policy. Under this constraint the *reference_mediation* function would grant access requests issued by the subjects of user u_1 that are contained in either *caps*(Doctor) $\cup$ *caps*(Intern), or *caps*(Consultant).

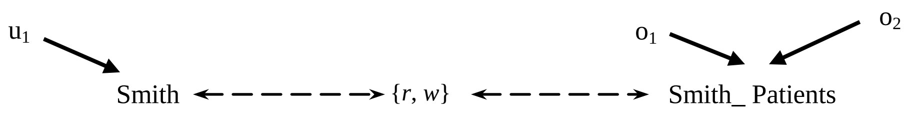
Figure 1(b) depicts attributes in support of a Multi-level Security Policy. Under the MLS policy [6], security levels are assigned to users and objects, and associated with subjects. The security levels are partially ordered under a dominance relation, often written as "≥". For example H ≥ M ≥ L. Figure 1(b) infers users cleared to the levels of high (H), medium (M) and low (L) are respectively assigned to H, M and L user attributes and objects that are classified at the H, M and L levels are respectively assigned to H, M, and L object attributes. With respect to the security level of a subject and the security level of an object, access control decisions are made according to two properties: *Simple Security Property* – A subject is permitted read access to an object if the subject's security level dominates the security level of the object and **-Property* – A subject is permitted write access to an object if the object's security level dominates the security level of the subject. Notice that the capabilities of each user attribute preserve both the simple security property and *-property. The MLS model dictates that a subject is to be limited to a single attribute that is dominated by its user's clearance. By Imposing the following subject attribute constraint limits a subject to a single attribute that is dominated by its user's clearance and ensures the adherence to the simple security and star properties: ({H}, {M}, {L})∈SAC.



(a) Example RBAC policy attributes and assignments



(b) Example MLS policy attributes and assignments



(c) Example IBAC policy attributes and assignments

Figure 1: Three example policy attribute and assignment configurations

User u_1 is assigned to H and as such the subjects of u_1 are capable of activating attributes H, or M, or L. Table 1 summarizes permissible subject access request with respect to the object attributes and assignment relations of figure 2(b) and subject attributes imposed under the before mentioned subject attribute constraint relations.

The graph of figure 1(c) illustrates user and object attributes pertaining to a simple identity-based access control policy. User u_1 is assigned to Smith. Recall that a user is not considered an attribute, but an identifier such as the user's name is. The Smith attribute permits (assuming the *reference_mediation* function) subjects to read and write objects with the Smith_Patients attribute.

Table 1: Permitted accesses for subjects with attributes H, or M, or L, and objects assigned to H, or M, or L object attributes.

attrs(ua)\oa	H	M	L
subject(H)	r, w	r	r
subject(M)	w	r, w	r
subject(L)	w	w	r, w

6. ATTRIBUTE ACTIVATION

Before a user or subject is able to access information, the user's subject must first activate one or more attributes. In the face of subject attribute constraints, a decision must be made as to which legal attribute(s) are to be activated by a subject. For instance, in support of RBAC, the SE Linux operating system asks a user to select a role to be activated, or in its support of its MLS policy to select a security level [7]. In the RBAC/Web implementation [8], a user is presented with the largest subsets of roles that do not violate a constraint relation, and asked to choose. Regarding these systems, the attribute activation decision is made by the user.

The PM takes a dramatically different approach. Through the PM's capability and access control entry review functions (i.e., *caps*(*ua*) and *aces*(*oa*)), the PM is able to calculate and automatically activate a minimum subject attribute set that fits the optimal need of the user's access request. Subsequent user access requests issued through the same subject may activate additional attributes as long as the current set of attributes does not violate a subject attribute constraint. This approach has the advantage of access system transparency, strong support for least privilege (attributes are incrementally activated to fit the task at hand), and takes into consideration subject memory that may have accrued under prior subject attribute states (the attribute set is only augmented).

The PM's dialog with a user (previously authenticated) begins with the creation of a subject with an initial empty set of attributes. The *create_subject* procedure/operation creates a subject *s* for a user *u*:

$$\text{create\_subject}(u) = \text{success} \Leftrightarrow s \notin S,\ S' = S \cup \{s\},\ subj\_user(s) = u,\ attrs(s) = \varnothing.$$

Note that if the predicate *subject_attributes_valid* was true before the subject creation, it remains true after subject creation.

Once a user creates a subject the user may issue an access request to the subject, automatically activating one or more appropriate attributes.

Subject attribute activation: The *subject_attr_activation* procedure/operation activates a set of attributes *uas* for a subject *s* such that the subject is capable of performing the maximum number of operations in *ops* of a request $<ops, o>_u$ of the user *u* associated with that subject, where $ops \subseteq Op$ and $o \in O$.

In our formalism, the newly activated attribute set will appear as the range of a partial function that (a) maps an operation *op* in *ops* to a user attribute *ua* that has the capability (*op*, *o*); (b) has a maximal domain; and (c) the subject's attribute set augmented with the function's range satisfies the PM's subject attribute constrains.

We first define the set of "valid" attribute selection functions for subject *s* and request $<ops, o>_u$, denoted by $ASF(<ops, o>_u , s)$, as follows:

$$asf \in ASF(<ops, o>_u , s) \overset{def}{\Leftrightarrow}$$

$$asf : ops \xrightarrow{p} UA \wedge$$

$$(subj\_user(s) = u) \wedge$$

$$(\forall op \in \text{dom } asf,\ \forall ua \in UA,\ ua = asf(op) \Rightarrow u \in users(ua) \wedge (op, o) \in caps(ua)) \wedge$$

$$attribute\_set\_valid(attrs(s) \cup \text{ran } asf) \wedge$$

$$\text{dom } asf \neq \varnothing.$$

We now formally define the operation $subject\_attr\_activation(<ops, o>_u , s)$:

$$subject\_attr\_activation(<ops, o>_u , s) = success \Leftrightarrow$$

$$\exists\ asf_0 \in ASF(<ops, o>_u , s) :$$

$$(\forall asf \in ASF(<ops, o>_u , s), \text{dom } asf \subseteq \text{dom } asf_0) \wedge$$

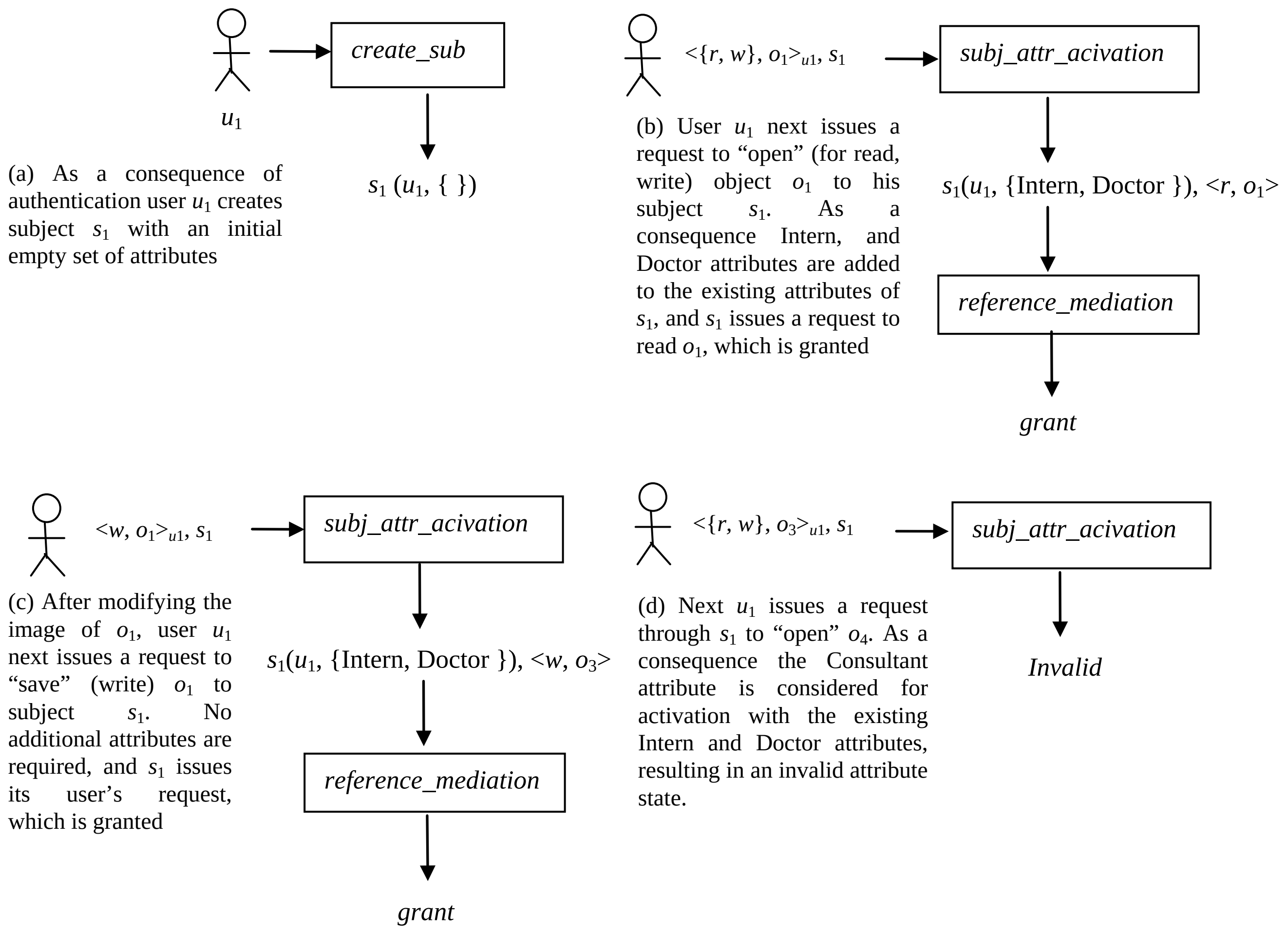
$$(attrs'(s) = attrs(s) \cup \text{ran } asf_0).$$

Obviously, in the new state the predicate *subject_attribute_valid* is true. Note that how the PM selects the attributes *ua* is implementation-dependent and is not specified here. There are also a number of implementation choices when *subject_attributes_valid* evaluates to false. Included among these choices is a message sent to the user or the creation of a new subject with initial attribute(s) equal to *attr*'(*s*). Figure 2 illustrates the application of the three PM functions for a sequence of user requests with respect to the RBAC policy described in section 5 and illustrated in figure 1(a). Recall the existence of the subject attribute constraint ({Doctor, Intern}, {Consultants})∈*SAC*, pertaining to the RBAC policy.

As another example, assume a user u_3 that is assigned to Intern, along with u_3's newly created subject *s* in the context of figure 1a, where u_3 issues a request ($<\{r, w\}, o_2>_{u3}$) to *s*. As a consequence of this request *attrs*(*s*)={*Intern*}, but u_3 would only be able to open o_2 for reading.

7. COMBINING POLICIES

In section 5 we demonstrated the PM's ability to configure instances of RBAC, MLS and IBAC policies. Other instances of these policies, as well as other entirely new classes of policy instances (e.g., Biba integrity [9]) could be configured. In this section we describe and demonstrate the PM's ability to configure policy combinations of different access control policies and its ability to enforce these policy combinations in response to user and subject access requests.



(a) As a consequence of authentication user u_1 creates subject s_1 with an initial empty set of attributes

(b) User u_1 next issues a request to "open" (for read, write) object o_1 to his subject s_1. As a consequence Intern, and Doctor attributes are added to the existing attributes of s_1, and s_1 issues a request to read o_1, which is granted

(c) After modifying the image of o_1, user u_1 next issues a request to "save" (write) o_1 to subject s_1. No additional attributes are required, and s_1 issues its user's request, which is granted

(d) Next u_1 issues a request through s_1 to "open" o_4. As a consequence the Consultant attribute is considered for activation with the existing Intern and Doctor attributes, resulting in an invalid attribute state.

Figure 2: Sequence of user requests per figure 2(a)

Consider the combination of the three access control policies presented in section 5. Under these individual policies only doctors can perform read and write operations on medical records, only users that are cleared to the M level may read and write objects classified at the M level, and only Smith may read and write Smith Patients objects. Under combinations of these policies only users that are Doctors, and are cleared to the M level, and are Smith may perform read and write operations on Med_Records that are classified at the M level, and are Smith_Patients. In combining policies it is important to note that in general not all classes of policies protect all objects and not all user and subject requests are controlled under all policy classes, nor are all user and object attributes relevant to all classes of policies.

To provide a mapping of users, user attributes, objects and object attributes to their relevant policy classes we introduce the notion of policy classes. Let *PC* be the set of policy classes. We define an assignment relation between user attributes and policy classes, denoted by the symbol $\to$ ($\to \subseteq UA \times PC$) with the property that every user attribute is "eventually assigned" to a policy class through a chain of attribute assignments:

$$\forall ua \in UA, \exists ua_1, \exists pc \in PC: ua \to^* ua_1 \to pc.$$

Similarly, between object attributes and policy classes we define an assignment relation, $\to \subseteq OA \times PC$, with the property that every object attribute is "eventually assigned" to a policy class through a chain of attribute assignments:

$$\forall oa \in OA, \exists oa_1 \in OA, \exists pc \in PC: oa \to^* oa_1 \to pc.$$

For convenience, for $ua \in UA$ and $pc \in PC$, let's denote by $ua \to^+ pc$ the fact that $\exists ua_1 \in UA$, such that $ua \to^* ua_1 \to pc$. Similarly, for $oa \in OA$ and $pc \in PC$, let's denote by $oa \to^+ pc$ the fact that $\exists oa_1 \in OA$, such that $oa \to^* oa_1 \to pc$.

For $u \in U$, ua, ua_1, $ua_2 \in UA$, oa_1, $oa_2 \in OA$, and $pc \in PC$, we use the notation $\underset{pc}{\to}$ to denote attribute assignments within the policy class *pc*:

$$u \underset{pc}{\rightarrow} ua \overset{def}{\Leftrightarrow} u \rightarrow ua \text{ and } ua \rightarrow^{+} pc$$

$$ua_1 \underset{pc}{\rightarrow} ua_2 \overset{def}{\Leftrightarrow} ua_1 \rightarrow ua_2 \text{ and } ua_2 \rightarrow^{+} pc$$

$$oa_1 \underset{pc}{\rightarrow} oa_2 \overset{def}{\Leftrightarrow} oa_1 \rightarrow oa_2 \text{ and } oa_2 \rightarrow^{+} pc$$

$$oa_1 \underset{pc}{\rightarrow}{*} oa_2 \overset{def}{\Leftrightarrow} oa_1 \rightarrow{*} oa_2 \text{ and } oa_2 \rightarrow^{+} pc$$

Figure 3 illustrates the combination of three policy classes—RBAC, MLS and IBAC that were each individually presented in section 5. The dotted arrows depict policy class assignments. Note that u_1, o_1, and o_2 are included in RBAC, MLS, and IBAC policies (i.e., $u_1 \underset{RBAC}{\rightarrow}$ Doctor, $u_1 \underset{RBAC}{\rightarrow}$ Conslt, $u_1 \underset{MLS}{\rightarrow}$ H, $u_1 \underset{IBAC}{\rightarrow}$ Smith, $o_2 \underset{RBAC}{\rightarrow}$ Med_Recds, $o_2 \underset{MLS}{\rightarrow}$ M, $o_2 \underset{IBAC}{\rightarrow}$ Smith_Pats, $o_1 \underset{RBAC}{\rightarrow}$ Med_Recds, $o_1 \underset{MLS}{\rightarrow}$ L, $o_1 \underset{IBAC}{\rightarrow}$ Smith_Pats) while o_3 is only included in the RBAC policy (i.e., $o_3 \underset{RBAC}{\rightarrow}$ Devlmt).

Now we can define the attribute set of a subject s within a given policy class pc as follows:

$$attr_{pc}(s) = attrs(s) \cap \{ua \in UA \mid ua \rightarrow^{+} pc\},$$

i.e., we consider only the attributes of subject s that are "eventually assigned" to policy class pc.

Also, we define the set of capabilities of a user attribute ua within a given policy class pc as follows:

$$caps_{pc}(ua) = \{(op, o) \mid ua \rightarrow^{+} pc \wedge \exists ops \subseteq Op, \exists oa \in OA: op \in ops, o \underset{pc}{\rightarrow}{}^{*} oa, ua \rightarrow ops \rightarrow oa\}.$$

We modify the predicate *subject_attributes_valid* for multiple policy classes by requiring the attribute set of each subject s within each policy class pc to satisfy the condition of the former *subject_attributes_valid* predicate. Formally:

$$subject\_attributes\_valid \overset{def}{\Leftrightarrow} \forall s \in S, \forall pc \in PC, attribute\_set\_valid(attrs_{pc}(s)).$$

The subject attribute activation operation for multiple policy classes: Because an object may be protected under multiple policies, a subject's requesting access to the object must satisfy all policies that protect the object. For a given user request $<ops, o>_u$ and a subject s executing on behalf of u, the subject attribute activation operation activates, for each policy class pc, an attribute set $uas \subseteq UA$, such that subject s is capable of performing the maximum number of operations in ops of the request $<ops, o>_u$.

In our formalism, for each policy class containing object o, the newly activated attribute set will appear as the range of a partial function that (a) maps an operation op in ops to a user attribute ua that has the capability (op, o) in that policy class; (b) has a maximal domain – in order to allow the subject to perform as many operations of the request as possible; and (c) the subject's attribute set augmented with the function's range satisfies PM's subject attribute constraints with each policy class.

We first define the set of "valid" attribute selection functions for subject s and request $<ops, o>_u$ within policy class pc, denoted by $ASF_{pc}(<ops, o>_u , s)$, as follows:

$$asf \in ASF_{pc}(<ops, o>_u , s) \overset{def}{\Leftrightarrow}$$

$$asf : ops \xrightarrow{p} UA \wedge$$
$$(subj\_user(s) = u) \wedge$$
$$(\forall op \in \text{dom } asf, \forall ua \in UA, ua = asf(op) \Rightarrow u \in users(ua) \wedge (op, o) \in caps_{pc}(ua)) \wedge$$
$$attribute\_set\_valid(attrs_{pc}(s) \cup \text{ran } asf) \wedge$$
$$\text{dom } asf \neq \varnothing.$$

We can now formally define the operation *subject_attr_activation*($<ops, o>_u , s$):

$$subject\_attr\_activation(<ops, o>_u , s) = success \Leftrightarrow$$

$$\forall pc \in PC, \exists\, asf_{0,pc} \in ASF_{pc}(<ops, o>_u , s) :$$
$$(\forall asf \in ASF_{pc}(<ops, o>_u , s), \text{dom } asf \subseteq \text{dom } asf_{0,pc}) \wedge$$
$$(attrs_{pc}'(s) = attrs_{pc}(s) \cup \text{ran } asf_{0,pc}).$$

Obviously, the predicate *subject_attributes_valid* will be satisfied in the new state.

For example consider the request $<\{r, w\}, o_2>_{u1}$ issued to a newly created subject s, in the context of figure 3. Because o_2 is included in three policies (RBAC, MLS and IBAC) the subject attribute activation operation (as originally defined in section 6, and augmented above), would activate three attribute sets—$attrs_{RBAC}(s)$={Intern, Doctor}, $attrs_{MLS}(s)$={M}, $attrs_{IBAC}(s)$={Smith} where all attribute sets adhere to attribute constraint relations (defined in section 5).

Reference Mediation Under Multiple Policies: The reference mediation function grants the subject s the permission to execute a request $<op, o>_s$ if and only if the pair (op, o) is a capability of an attribute ua of subject s within each policy class that contains the object o. Formally,

$$\forall s \in S, \forall op \in Op, \forall o \in O:$$

$$reference\_mediation(<op, o>_s) = grant \Leftrightarrow (op, o) \in \bigcap_{\substack{pc \in PC \\ o \rightarrow^{+} pc}} \bigcup_{ua \in attr_{pc}(s)} caps_{pc}(ua) .$$

Assuming the attribute sets of subject s that were activated in the proceeding example by the user request $<\{r, w\}, o_2>_{u1}$, the following subject requests $<r, o_2>_s$, $<w, o_2>_s$, and $<r, o_1>_s$ would each be granted, while subject requests $<w, o_1>_s$, $<r, o_3>_s$, $<w, o_3>_s$, would each be denied.

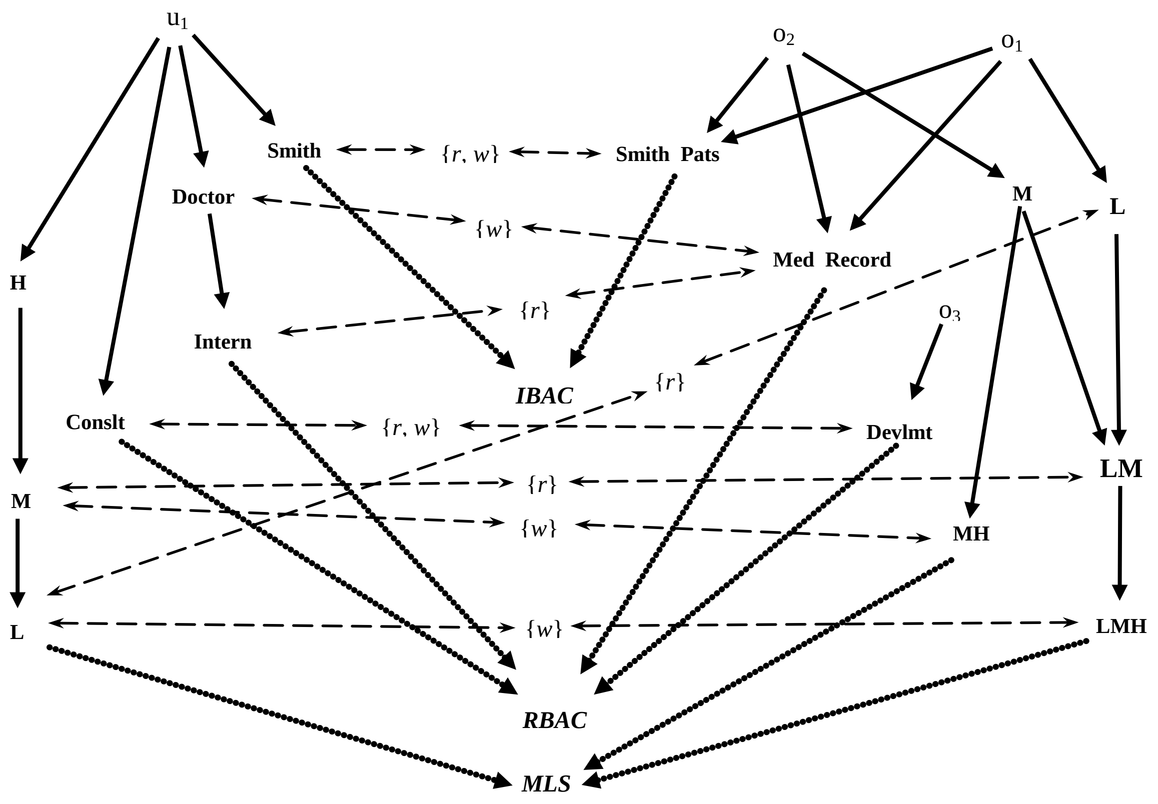


Figure 3: Example combination of RBAC, MLS, and IBAC Policies

8. RELATED WORK

The basic objective of our research is to defining a standardized access control mechanism (that we refer to as the Policy Machine) that requires changes only in its configuration in affording arbitrary and organization specific attribute-based access control policy. The PM is not unique in this pursuit of policy flexibility. The RBAC model of Sandhu, et al., often referred to as RBAC96 [12] is considered policy neutral in its ability to specify a large variety of policies. Although there are many similarities between RBAC and the PM, important differences exist.

In [10], Saunders formally analyzes and characterizes RBAC in terms of the Access Control Matrix (ACM) and its closely related derivatives (access control lists and capabilities) and not surprisingly concludes that RBAC fundamentally shares the administrative benefits and costs of a capability model. To apply similar analysis we would expect the PM to be characterized as both a capability and an access control list model (and would enjoy the benefits of both, but none of the costs). In this characterization the PM is similar to domain-type enforcement mechanisms [11]. The PM is different from domain-type enforcement and RBAC in the following ways. PM offers partial ordering and inheritance between both user and object attributes that can be achieved through its direct configuration. RBAC only offers configurable inheritance among user attributes. Domain-type enforcement offers neither of these two features without adding to its implementation.

While there are clear structural differences between RBAC and the PM as noted above, RBAC nonetheless has been demonstrated to be flexible in its ability to configure multiple policy classes. An important question is how efficient and how natural is RBAC and the PM in configuring policy? To analyze these differences we take a closer look at the Osborn, Sandhu, and Munawer configuration of RBAC96 [12] in the emulation of an MLS policy [13], and compare it to that of the PM's configuration of the same policy as presented in section 5.

In the construction of this policy Osborn et al., assumes the existence of a lattice of security labels *SC*, $\{L_1 \ldots L_n\}$ with a partially ordered dominance relation ≥ and a least upper bound operator (e.g., $H \geq M \geq L$), two role hierarchies one for read, consisting of roles $\{L_1R \ldots L_nR\}$and the other for write, consisting of roles $\{L_1W \ldots L_nW\}$ where the second hierarchy has a partial order which is the inverse of the first. With respect to the roles in these hierarchies there are a series of obligation constraints (that are permitted but not specified in RBAC96) that are defined to appropriately create capabilities (or permissions per RBAC96) and capability-role assignments, to create user-role assignments, and to activate subject attributes (session roles per RBAC96). For each user in the policy the user is assigned to two roles, xR and LW where x is the label (clearance) assigned to the user and LW is the write role corresponding to the lowermost security level according to ≥. For each object *o* of label x (classification), there are two capabilities created (*o*, *r*) and (*o*, *w*), that are assigned to xR and xW roles respectively. In consideration of the above constructions, a user establishes a logon session at level y where roles yR and yW are activated, such that x ≥ y, and x is the users clearance level and consequently permits user accesses that conform to the simple security and *-property of [6].

In our configuration of the MLS policy each user was assigned to just one user attribute (at the user's clearance level), each object was assigned to just one object attribute (at the objects classification level), capabilities were not needed to be dynamically created, and the user's subject activated a single attribute that was dominated by his/her clearance. This suggests that the PM's MLS configuration is more efficient (requires fewer assignments) then that of RBAC's configuration. We also feel that the PM is more natural in its embodiment of the MLS policy. Although RBAC96 includes provisions for a variety of constraints, a standard implementation of RBAC96 would not necessarily include the obligation constraints that were applied in the construction of the MLS policy and if it were these obligation constraints would not generally apply to other policies.

The Flexible Authorization Framework [14] strategy for policy flexibility exploits the hierarchical structures in which fixed system components (objects, users, groups, and roles) are organized. Groups define a grouping of people and roles define groupings of capabilities, but only roles are allowed to be activated. The PM's user attributes abstracts away the difference between groups and role, since either can be instantiated as a user attribute instance, and as such both may be activated, perhaps in support of different policies.

In its protection of objects under a multitude of policies, the PM includes a capability to categorize users, objects and attributes into their respective classes of policies, and appropriately enforces these policies in response to a user's access request. Although products that protect objects under an MLS policy traditionally also protect these same objects under a need-to-know policy, such products afford these policy combinations through the deployment of two separate mechanisms, one in support of the MLS policy and the other in support of the need-to-know policy. The PM is different in this regard in its ability to enforce multiple arbitrary policies through the application of a single mechanism.

9. CONCLUSIONS

In this paper we presented core features of the Policy Machine that are capable of configuring, combining and enforcing arbitrary attribute-based policies. The PM is not an extension of any other access control model, but instead we have attempted to specify the PM in terms of access control abstractions, functions and properties basic to access control in general. These features include the ability to generically represent arbitrary user and object attributes, which are associated with subjects and applied in mediating a subject's request to access objects. In its protection of objects under a multitude of policies, the PM includes a capability to categorize users, objects and attributes into their respective classes of policies, and appropriately enforcing these policies in response to a user's access request.

10. REFERENCES

1. National Commission on Terrorist Attacks Upon the United States. The 9/11 Commission Report, 2004.
2. Anderson, J.P., Computer Security Technology Planning Study, Tech Report ESD-TR-73-51, US Air Force Electronic Systems Div., Hanscom AFB, 1972.
3. B. Lampson. Protection. *ACM Operating Sys. Reviews*, 8, 1 (1974), 18-24.
4. Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proc. IEEE*, 63, 9 (September 1975), 1278–1308.
5. ANSI INCITS 359-2004, Role-Based Access Control.
6. D. Bell and La Padula. Secure computer systems: unified exposition and MULTICS. Report ESD-TR-75-306, The MITRE Corporation, Bedford, Massachusetts, March 1976.
7. Peter A. Loscocco, and Stephen P. Smalley. Meeting Critical Security Objectives with Security Enhanced Linux, *Proc. 2001 Ottowa Linux Symposium*, 2001.
8. D.F. Ferraiolo, J. Barkley, D.R. Kuhn, A Role Based Access Control Model and Reference Implementation within a Corporate Intranet, *ACM Transactions on Information Systems Security*, 1, 2 (February 1999), 34-64.
9. K. J. Biba. *Integrity Considerations for Secure Computer Systems*. Technical Report ESD-TR-76-372, USAF Electronic Systems Division, Hanscom Air Force Base, Bedford, Massachusetts, (April 1977).
10. G. Saunders. Role-Based Access Control and the Access Control Matrix. *ACM SIGOPS Operating System and Review*, 35, 4 (2001), 6-20.
11. L. Badger, et al. A Domain and Type Enforcement Prototype. *Computing Systems*, 9, 1 (1996), 47-83.
12. R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-Based Access Control Models, *IEEE Computer,* 29, 2 (Feb. 1996), 38-47.
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 Systems Security*, 3, 2 (May 2002), 85-106.
14. S. Jajodia, S. Pierangela, M. L. Sapino, V. S. Sabrahmanian. Flexible Support for Multiple Access Control Policies, *ACM Transactions on Database Systems,* 26, 2 (June 2001), 214-260.