Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC)

David Ferraiolo, Ramaswamy Chandramouli, Rick Kuhn and Vincent Hu

National Institute of Standards and Technology
Gaithersburg, Maryland 20899
{dferraiolo, mouli, Kuhn, vhu}@nist.gov

ABSTRACT
Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC) are very different attribute based access control standards with similar goals and objectives. An objective of both is to provide a standardized way for expressing and enforcing vastly diverse access control policies in support of various types of data services. The two standards differ with respect to the manner in which access control policies and attributes are specified and managed, and decisions are computed and enforced. This paper is presented as a consolidation and refinement of public draft NIST SP 800-178 [21], describing, and comparing these two standards.

Keywords
ABAC; XACML; NGAC; Policy Machine; Access Control

1. INTRODUCTION
Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC) offer different approaches to attribute based access control (ABAC). XACML, available since 2003, is an Extensible Markup Language (XML) based language standard designed to express security policies, as well as the access requests and responses needed for querying the policy system and reaching an authorization decision [17]. XACML was developed as collaboration among vendors with a goal to separate policy expression and decision-making from proprietary operating environments in support of the access control needs of applications. NGAC is an emerging, relations and architecture-based standard designed to express and enforce access control policies, through configuration of relations [2], [20]. NGAC stems from and is in alignment with the Policy Machine, a research effort to develop a general-purpose ABAC framework [6], [7], [8], [9].

What are the similarities and differences between these two standards? What are their comparative advantages and disadvantages? These questions are particularly relevant because XACML and NGAC provide different means of achieving a common access control goal—to allow vastly different access policies to be expressed and enforced in data services using the features of the same underlying mechanism in diverse ways. These are also important questions, given the prevalence of data services in computing. Data services include computational capabilities that allow the consumption, alteration, management, and sharing of data resources. Data services can take on many forms, to include applications such as time and attendance reporting, payroll processing, and health benefits management, but also including system level utilities such as file management.

This paper describes XACML and NGAC and compares them with respect to five criteria. The first criterion is the relative degree to which the access control logic of a data service can be separated from a proprietary operational environment. The other four criteria are derived from ABAC issues or considerations identified by NIST Special Publication (SP) 800-162 [13]: operational efficiency, attribute and policy management, scope and type of policy support, and support for administrative review and resource discovery.

2. BACKGROUND
Controlling and managing access to sensitive data has been an ongoing challenge for decades. ABAC represents the latest milestone in the evolution of logical access control methods. It provides an attribute-based approach to accommodate a wide breadth of access control policies and simplify access control management.

Most other access control approaches are based on the identity of a user requesting execution of a capability to perform an operation on a data resource (e.g., read a file), either directly via the user's identity, or indirectly through predefined attribute types such as roles or groups assigned to the user. Practitioners have noted that these forms of access control are often cumbersome to set up and manage, given their need to, and the difficulty of, associating capabilities directly to users or their attributes. Furthermore, the identity, group, and role qualifiers of a requesting user are often insufficient for expressing real-world access control policies. An alternative is to grant or deny user requests based on arbitrary attributes of users and arbitrary attributes of data resources, and optionally environmental attributes that may be globally recognized and tailored to the policies at hand. This approach to access control is commonly referred to as attribute-based access control (ABAC) and is an inherent feature of both XACML and NGAC.

The XACML and NGAC standards also enable decoupling of data service access control logic from proprietary operating environments (e.g., operating system, middleware, application).

This paper is authored by an employee(s) of the United States Government and is in the public domain. Non-exclusive copying or redistribution is allowed, provided that the article citation is given and the authors and agency are clearly identified as its source.
ABAC'16, March 11 2016, New Orleans, LA, USA
ACM 978-1-4503-4079-3/16/03
DOI: http://dx.doi.org/10.1145/2875491.2875496

More precisely, a data service is normally comprised of an application layer and an operating environment layer that can be delineated by their functionality and interfaces. The application layer provides a user interface and methods for presentation, manipulation, management and sharing of data. The application layer does not carry out operations that consume data, alter the state of data, organize data, or alter the access state to data, but instead issue requests to the operating environment layer to perform those operations. An operating environment implements operational routines (e.g., read, write/save) to carry out application access requests as well as access control routines to ensure executions of user processes involving operational routines are policy preserving.

Access control routines comprise several components that work together to bring about policy-preserving data resource access. These components include access control data for expressing access control policies and representing attributes, and a set of functions for trapping access requests, and computing and enforcing access decisions over those requests. Most operating environments implement access control in different ways, each with a different scope of control (e.g., users, resources), and each with respect to different operation types (e.g., read, send, approve, select) and data resource types (e.g., files, messages, work items, records).

This heterogeneity introduces a number of administrative and policy enforcement challenges. Administrators are forced to contend with a multitude of security domains when managing access policies and attributes. Even if properly coordinated across operating environments, global controls are hard to visualize and implement in a piecemeal fashion. Furthermore, because operating environments implement access control in different ways, it is difficult to exchange and share access control information across operating environments. XACML and NGAC seek to alleviate these problems by creating a common and centralized way of expressing all access control data (policies and attributes) and computing and enforcing decisions, over the access requests from applications.

3. XACML

For purposes of brevity and readability, the XACML specification is presented as a summary that is intended to highlight XACML's salient features and should not be considered complete. In some instances, actual XACML terms are substituted with equivalent terms to accommodate a simpler and more consolidated presentation.

3.1 Attributes and Policies

An XACML access request consists of subject attributes (typically for the user who issued the request), resource attributes (the resource for which access is sought), action attributes (the operations to be performed on the resource), and environment attributes.

XACML attributes are specified as name-value pairs, where attribute values can be of different types (e.g., integer, string). An attribute name/ID denotes the property or characteristic associated with a subject, resource, action, or environment. For example, in a medical setting, the attribute name Role associated with a subject may have doctor, intern, and admissions nurse values, all of type string. Subject and resource instances are specified using a set of name-value pairs for their respective attributes. For example, the subject attributes used in a Medical Policy may include: Role = "doctor", Ward = "pediatrics"; an environmental attribute: Time = 12:11; and resource attributes: Resource-id = "medical-records", WardLocation = "pediatrics", Patient = "johnson".

Subject and resource attributes are stored in repositories and are retrieved through the Policy Information Point (PIP) at the time of an access request and prior to or during the computation of the decision. XACML formally defines an action as a component of a request with attribute values that specify operations such as read, write, submit, and approve.

Environmental attributes, which depend on the availability of system sensors that can detect and report values, are somewhat different from subject and resource attributes, which are administratively created. These environmental characteristics are subject and resource independent, and may include the current time, day of the week, or threat level.

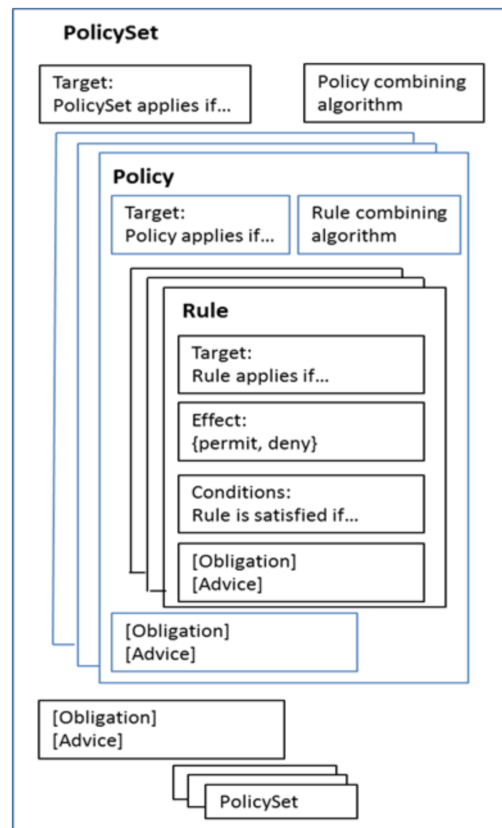

Figure 1. XACML Policy Constructs

As shown by Figure 1, XACML access policies are structured as PolicySets that are composed of Policies and optionally other PolicySets, and Policies that are composed of Rules. Policies and PolicySets are stored in a Policy Retrieval Point (PRP). Because not all Rules, Policies, or PolicySets are relevant to a given request, XACML includes the notion of a Target. A Target defines a simple Boolean condition that, if satisfied (evaluates to True) by the attributes, establishes the need for subsequent evaluation by a Policy Decision Point (PDP). If no Target matches the request, the decision computed by the PDP is NotApplicable.
In addition to a Target, a rule includes a series of boolean conditions that if evaluated True have an effect of either Permit or Deny. If the target condition evaluates to True for a Rule and the Rule's condition fails to evaluate for any reason, the effect of the Rule is Indeterminate. In comparison to the (matching) condition

of a Target, the conditions of a Rule or Policy are typically more complex and may include functions (e.g., “greater-than-equal”, “less-than”, “string-equal”) for the comparison of attribute values. Conditions can be used to express access control relations (e.g., a doctor can only view a medical record of a patient assigned to the doctor’s ward) or computations on attribute values (e.g., sum(x, y) less-than-equal:250).

3.2 Combining Algorithms

Because a Policy may contain multiple Rules, and a PolicySet may contain multiple Policies or PolicySets, each Rule, Policy, or PolicySet may evaluate to different decisions (Permit, Deny, NotApplicable, or Indeterminate). XACML provides a way of reconciling the decisions each makes. This reconciliation is achieved through a collection of combining algorithms. Each algorithm represents a different way of combining multiple local decisions into a single global decision. There are several combining algorithms, to include the following:

- Deny-overrides: if any decision evaluates to Deny, or no decision evaluates to Permit, then the result is Deny. If all decisions evaluate to Permit, the result is Permit.
- Permit-overrides: if any decision evaluates to Permit, then the result is Permit, otherwise the result is Deny.

Combining algorithms are applied to rules in a Policy and Policies within a PolicySet in arriving at an ultimate decision of the PDP. Combining algorithms can be used to build up increasingly complex policies. For example, given that a subject request is Permitted (by the PDP) only if the aggregate (ultimate) decision is Permit, the effect of the Permit-overrides combining algorithm is an “OR” operation on Permit (any decision can evaluate to Permit), and the effect of a Deny-overrides is an “AND” operation on Permit (all decisions must evaluate to Permit).

3.3 Obligations and Advice

XACML includes the concepts of obligation and advice expressions. An obligation optionally specified in a Rule, Policy, or PolicySet is a directive from the PDP to the Policy Enforcement Point (PEP) on what must be carried out before or after an access request is approved or denied. Advice is similar to an obligation, except that advice may be ignored by the PEP. A few examples include:

- If Alice is denied access to document X: email her manager that Alice tried to access document X.
- If a user is denied access to a file: inform the user why the access was denied.
- If a user is approved to view document X: watermark the document “DRAFT” before delivery.

3.4 Example Policy

Consider the following example XACML policy specification. For purposes of maintaining the same semantics as XACML, we use the same element names, but specify policies and rules in pseudocode for purposes of enhanced readability (instead of exact XACML syntax).

Policy 1 applies to “All read or write accesses to medical records by a doctor or intern” (the target of the policy) and includes three rules. As such, the policy is considered “applicable” whenever a subject with a role of “doctor” or “intern” issues a request to read or write a “medical-records” resource. The rules do not refine the target, but describe the conditions under which read or write requests from doctors or interns to medical records can be allowed. Rule 1 will deny any access request (read or write) if the ward in which the doctor or intern is assigned is not the same ward where the patient is located. Rule 2 explicitly denies “write” access requests to interns under all conditions. Rule 3 permits read or write access to medical-records for “doctor”, regardless of Rule 1, if an additional condition is met. This additional condition pertains to patients in critical status. Since the intent of the policy is to allow access under these critical situations, a policy combining algorithm of “permit-overrides” is used, while still denying access if only the conditions stated in Rule 1 or Rule 2 apply.

```
<Policy PolicyId = “Policy 1” rule-combining-algorithm=”permit-overrides”>
        // Doctor Access to Medical Records //
  <Target>
    /* :Attribute-Category    :Attribute ID    :Attribute Value */
         :access-subject      :Role            :doctor
         :access-subject      :Role            :intern
         :resource            :Resource-id     :medical-records
         :action              :Action-id       :read
         :action              :Action-id       :write
  </Target>

  <Rule RuleId = “Rule 1” Effect=”Deny”>
      <Condition>
  Function: string-not-equal
         /* :Attribute-Category   :Attribute ID */
              :access-subject     :WardAssignment
              :resource           :WardLocation
      </Condition>
   </Rule>

   <Rule RuleId = “Rule 2” Effect=”Deny”>
      <Condition>
  Function: string-equal
       /* :Attribute-Category   :Attribute ID    :Attribute Value */
              :access-subject   :Role            :intern
              :action           :Action-id       :write
      </Condition>
   </Rule>

   <Rule RuleId = “Rule 3” Effect=”Permit”>
      <Condition>
 Function:and
  Function: string-equal
       /* :Attribute-Category   Attribute ID     :Attribute Value */
              :access-subject   :Role            :doctor
  Function: string-equal
       /* :Attribute-Category   :Attribute ID    :Attribute Value */
              :resource         :PatientStatus   :critical
      </Condition>
   </Rule>
 </Policy>
```

Together policies (PolicySets and Policies) and attribute assignments define the authorization state. Table 1 defines the authorization state for Policy 1 by specifying attribute names and values. We use a functional notation for reporting on attribute values with the format A(), where the parameter may be a subject or resource.

Table 1. Attribute Names and Values and the Authorization

State for Policy 1

Subject Attribute Names and their Domains: Role = {doctor, intern} WardAssignment = {ward1, ward2}
Resource Attribute Names and their Domains: Resource-id = {medical-records} WardLocation = {ward1, ward2} PatientStatus = {critical}
Action Attribute Names and their Domains: Action-id = {read (r), write (w)}
Attribute value assignments when there are two subjects (s1, s2) and three resources (r1, r2, r3): A(s1) = <doctor, ward2>, A(s2) = <intern, ward1>, A(r1) = <medical-records, ward2, ‘ ’>, A(r2) = <medical-records, ward1, ‘ ’>, and A(r3) = < medical-records, ward1, critical>.
Authorization state: (s1, r, r1), (s1, w, r1), (s1, r, r3), (s1, w, r3), (s2, r, r2)

3.5 Delegation

The XACML Policies discussed thus far have pertained to Access Policies that are created and may be modified by an authorized administrator. These access policies are not associated with a specific “Issuer” and are considered “trusted”. As such, a “Trusted Access Policy” is directly used by the PDP in a combining algorithm applicable for the policy. In situations where policy creation needs to be delegated from a centralized administrator to a subordinate administrator, there is the need to create a new category of policies that control what policies can be created by the subordinate administrators. This new category of policies is called “Administrative Policies”. Similer to Access Policies, Administrative Policies not associated with a specific issuer are considered trusted and refered to as “Trusted Administrative Policies”.

Administrative policies include a delegate and a situation in its Target. A *situation* scopes the access rights that can be delegated and may include some combination of subject, resource, and action attributes. The *delegate* is an attribute category of the same type as a subject, representing the entity(s) that has (have) been given the authority to create either access or further delegation rights. If the delegate creates an Access Policy, then he/she becomes the “Issuer” for that policy. Such an Access Policy then is considered an “Untrusted Access Policy” since the authority under which it was created has to be verified. Similarly, when the delegate creates an “Administrative Policy”, the newly created policy is considered as an “Untrusted Administrative Policy” with the same trust verification requirement as “Untrusted Access Policy”.

Trusted Administrative Policies serve as a root of trust. They are created under the same authority used to create trusted Access Policies. A Trusted Administrative Policy gives the delegate the authority to create Untrusted Administrative Policies or Untrusted Access Policies. The *situation* for a newly created Untrusted Administrative Policy or Untrusted Access Policy is a subset (the same or narrower in scope) of that specified in the Trusted Administrative Policy. In addition, an Untrusted Administrative Policy or Untrusted Access Policy includes a *policy issuer* tag with a value that is the same as that of the delegate in the Administrative Policy under which it was created. Both of these policies have at least one rule with a PERMIT or DENY effect.

XACML with delegation profile recognizes two types of requests – Access Requests and Administrative Requests. Access Requests are issued to (attempt to match targets of) Access Policies or Untrusted Access Policies. An Untrusted Access Policy includes a Policy Issuer tag and an Access Policy does not. If the Access Request matches the target of an Access Policy, the PDP considers the Access Policy authorized and it is directly used by the PDP in a combining algorithm to arrive at a final access decision. If the Access Request matches the target of an Untrusted Access Policy, the authority of the policy issuer must first be verified before it can be considered by the PDP. Authority is determined through establishment of a *delegation chain* from the Untrusted Access Policy, through potentially zero or more Untrusted Administrative Policies, to a Trusted Administrative Policy. If the authority of the policy issuer can be verified, the PDP evaluates the access request against the Untrusted Access Policy; otherwise it is considered an unauthorized policy and discarded. In a graph where policies are nodes, a delegation chain consists of a series of edges from the node representing an Untrusted Access Policy to a Trusted Administrative Policy. To construct each edge of the graph, the XACML context handler formulates Administrative Requests.

An Administrative Request has the same structure as an Access Request except that in addition to attribute categories – access-subject, resource, and action – it also uses two additional attribute categories, delegate and decision-info. If a policy Px happens to be one of the applicable (matched) Untrusted Access Policies, the administrative request is generated using policy Px to construct an edge to policy Py using the following:

- Convert all Attributes (and attribute values) used in the original Access Request to attributes of category delegated.
- Include the value under the *PolicyIssuer* tag of Px as value for the subject-id attribute of the *delegate* attribute category.
- Include the effect of evaluating policy Px as attribute value (PERMIT, DENY, etc.) for the Decision attribute of *decision-info* attribute category.

The Administrative Request constructed is evaluated against the target for a policy Py. If the result of the evaluation is “Permit”, an edge is constructed between policies Px and Py. The objective is to verify the authority for issuance of policy Px. For this to occur there must exist a policy with its “delegate” set to the policy issuer of Px. If that policy is Py, then it means policy Px has been issued under the authority found in policy Py. The edge construction then proceeds from policy Py until an edge to a Trusted Administrative Policy is found. The process of selecting applicable policies for inclusion in the combining algorithm is illustrated in Figure 2.

By matching of the attributes in the original access request to the targets in various policies, Untrusted Access Policies P31, P32, and P33 can be found applicable. A path to a Trusted Administrative Policy P11 can be found directly from the applicable Untrusted Access Policy P31. A path to a Trusted Administrative Policy P12 can be found through Untrusted Administrative Policy P22 for the applicable Untrusted Access Policy P32. Because no such path can be found for P33, only the policies P31 and P32 will be used in the combining algorithm for evaluating the final access decision.

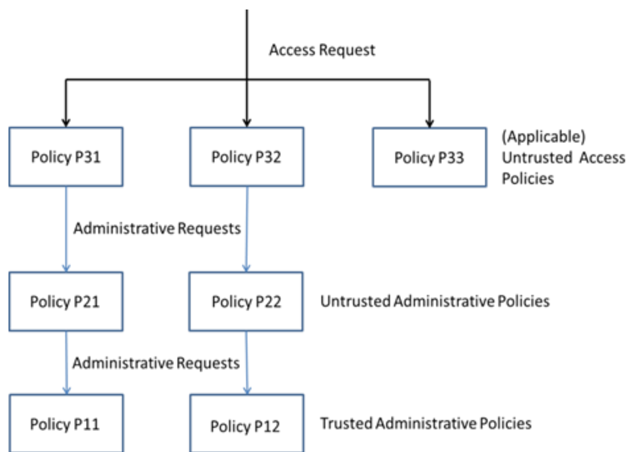

Figure 2. Utilizing Delegation Chains for Policy Evaluation

3.6 Reference Architecture

XACML reference architecture defines necessary functional components (depicted in figure 3) to achieve enforcement of its policies. The authorization process depends on four layers of functionality: Enforcement, Decision, Access Control Data, and Administration.

At its core is a PDP that computes decisions to permit or deny subject requests (to perform actions on resources). Requests are issued from, and PDP decisions are returned to a PEP using a request and response language. To convert access requests in native format (of the operating environment) to XACML access requests (or convert a PDP response in XACML to a native format), the XACML architecture includes a context handler. In the reference architecture in Figure 3, the context handler is not explicitly shown as a component since we assume that it is an integral part of the PEP or PDP.

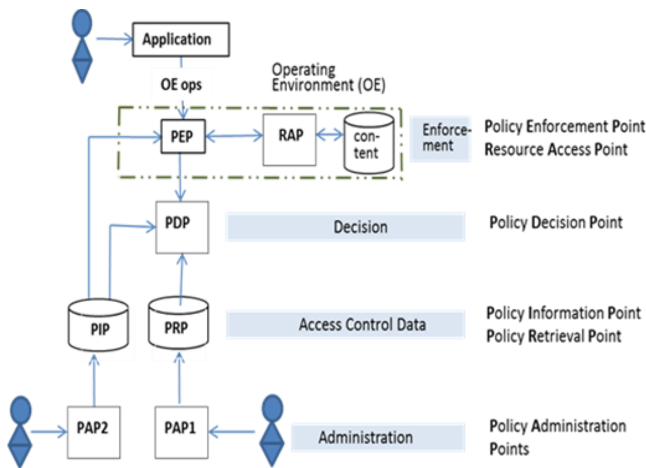

Figure 3. XACML Reference Architecture

A request is comprised of attributes extracted from the PIP, minimally sufficient for Target matching. The PIP is shown as one logical store, but in fact may comprise multiple physical stores. In computing a decision, the PDP queries policies stored in a PRP. If the attributes of the request are not sufficient for rule and policy evaluation, the PDP may request the context handler to search the PIP for additional attributes. Information and data stored in the PIP and PRP comprise the access control data and collectively define the current authorization state.

A Policy Administration Point (PAP1) using the XACML policy language creates the access control data stored in the PRP in terms of rules for specifying Policies, PolicySets as a container of Policies, and rule and combining algorithms. The PRP may store trusted or untrusted policies. Although not included in the XACML reference architecture, we show a second Policy Administration Point (PAP2) for creating and managing the access control data stored in the PIP. PAP2 implements administrative routines necessary for the creation and management of attribute names and values for users and resources. The Resource Access Point (RAP) implements routines for performing operations on a resource that is appropriate for the resource type. In the event that the PDP returns a permit decision, the PEP issues a command to the RAP for execution of the approved operation resource pair. As indicated by the dashed box in Figure 3, the RAP, in addition to the PEP, runs in an application's operating environment, independent of the PDP and its supporting components. The PDP and its supporting components are typically implemented as modules of a centralized Authorization Server that provides authorization services for arbitrary types of operations.

4. NGAC

NGAC takes a fundamentally different approach from XACML for representing requests, expressing and administering policies, representing and administering attributes, and computing and enforcing decisions. NGAC is defined in terms of a standardized and generic set of relations and functions that are reusable in the expression and enforcement of policies.

For purposes of brevity and readability, the NGAC specification is presented as a summary that highlights NGAC's salient features and should not be considered complete. In some instances, actual NGAC relational details and terms are substituted with others to accommodate a simpler presentation.

4.1 Policy and Attribute Elements

NGAC's access control data is comprised of basic elements, containers, and configurable relations. While XACML uses the terms subject, action, and resource, NGAC uses the terms user, operation, and object with similar meanings. In addition to these, NGAC includes processes, administrative operations, and policy classes. Like XACML, NGAC recognizes user and object attributes; however, it treats attributes along with policy class entities as containers. These containers are instrumental in both formulating and administering policies and attributes. NGAC treats users and processes as independent but related entities. Processes through which a user attempts access take on the same attributes as the invoking user.

Although an XACML resource is similar to an NGAC object, NGAC uses the term object as an indirect reference to its data content. Every object is an object attribute. The reference to an object is the value of its "name" attribute. Thus the value of the "name" attribute of an object is synonimus with the object. The set of objects reflects entities needing protection, such as files, clipboards, email messages, and record fields.

Similar to an XACML subject attribute value, NGAC user containers can represent roles, affiliations, or other common characteristics pertinent to policy, such as security clearances.

Object containers (attributes) characterize data and other resources by identifying collections of objects, such as those associated with certain projects, applications, or security classifications. Object containers can also represent compound objects, such as folders, inboxes, table columns, or rows, to satisfy the requirements of different data services. Policy class containers are used to group and characterize collections of policy or data services at a broad level, with each container representing a distinct set of related policy elements. Every user, user attribute, and object attribute must be contained in at least one policy class. Policy classes can be mutually exclusive or overlap to various degrees to meet a range of policy requirements.

NGAC recognizes a generic set of operations that include basic input and output operations (i.e., read and write) that can be performed on the contents of objects that represent data service resources, and a standard set of administrative operations that can be performed on NGAC access control data that represent policies and attributes. In addition, an NGAC deployment may consider and provide control over other types of resource operations besides the basic input/output operations. Administrative operations, on the other hand, pertain only to the creation and deletion of NGAC data elements and relations, and are a stable part of the NGAC framework.

4.2 Relations

NGAC does not express policies through rules, but instead through configurations of relations of four types: assignments (define membership in containers), associations (to derive privileges), prohibitions (to derive privilege exceptions), and obligations (to dynamically alter access state).

4.2.1 Assignments and Associations

NGAC uses a tuple (x, y) to specify the assignment of element x to element y. In this publication we use the notation x→y to denote the same assignment relation. The assignment relation always implies containment (x is contained in y). The set of entities used in assignments include users, user attributes, and object attributes (which include all objects), and policy classes.

To be able to carry out an operation, one or more access rights are required. As with operations, two types of access rights apply: non-administrative and administrative.

Access rights to perform operations are acquired through associations. An association is a triple, denoted by *ua---ars---at*, where *ua* is a user attribute, *ars* is a set of access rights, and *at* is an attribute, where *at* may comprise either a user attribute or an object attribute. The attribute *at* in an association is used as a referent for itself and the policy elements contained by the attribute. Similarly, the first term of the association, attribute *ua*, is treated as a referent for the users contained in *ua*. The meaning of the association *ua---ars---at* is that the users contained in *ua* can execute the access rights in *ars* on the policy elements referenced by *at*. The set of policy elements referenced by *at* is dependent on (and meaningful to) the access rights in *ars*.

Figure 4 illustrates assignment and association relations depicted as a graphs with two policy classes—Project Access, and File Management. Users and user attributes are on the left side of the graphs, and objects and object attributes are on the right. The arrows represent assignment or containment relations and the dashed lines denote associations.

Collectively associations and assignments indirectly specify privileges of the form (*u*, *ar*, *e*), with the meaning that user *u* is permitted (or has a capability) to execute the access right *ar* on element *e*, where *e* can represent a user, user attribute, or object attribute. Determining the existence of a privilege (a derived relation) is a requirement of, but as we discuss later, not sufficient in computing an access decision.

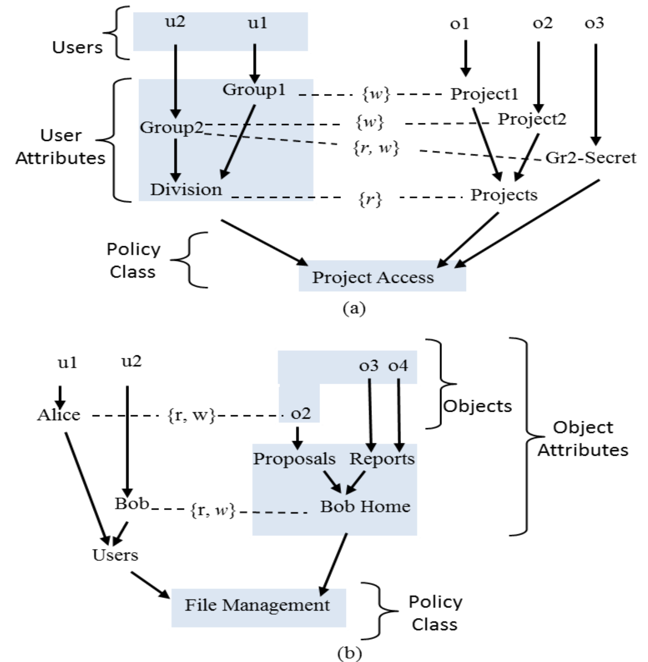

Figure 4: Two Example Assignment and Association Graphs

NGAC includes an algorithm for determining privileges with respect to one or more policy classes and associations. Specifically, (*u*, *ar*, *e*) is a privilege, if and only if, for each policy class *pc* in which *e* is contained, the following is true:

- The user *u* is contained by the user attribute of an association;
- The element *e* is contained by the attribute *at* of that association;
- The attribute *at* of that association is contained by the policy class *pc*, and
- The access right *ar* is a member of the access right set of that association.

The left and right columns of Table 2 respectively list derived privileges for Figures 4a and 4b, when considered independent of one another. Table 3 lists the privileges for these graphs in combination.

Note that (*u*1 *r*, *o*1) is a privilege in table 3 because o1 is only in policy class Project Access and there exist an association Division---{*r*}--- Projects, where u1 is in Division, r is in {r}, and o1 is in Projects. Note that (*u*1, *w*, *o*2) is not a privilege in table 3 because o2 is in both Project Access and File Management policy classes, and although there exist an association Alice---{r, w}---o2, where u1 is in Alice, w is in {r, w}, and o2 is in o2 and File Management, no such association exists with respect to Project Access.

Table 2: List of derived privileges for the independent configuration of Figures 4a and 4b

(*u*1, *r*, *o*1), (*u*1, *w*, *o*1), (*u*1, *r*, *o*2), (*u*2, *r*, *o*1), (*u*2, *r*, *o*2), (*u*2, *w*, *o*2), (*u*2, *r*, *o*3), (*u*2, *w*, *o*3)	(*u*1, *r*, *o*2), (*u*1, *w*, *o*2), (*u*2, *r*, *o*2), (*u*2, *w*, *o*2), (*u*2, *r*, *o*3), (*u*2, *w*, *o*3), (*u*2, *r*, *o*4), (*u*2, *w*, *o*4)

Table 3. List of derived privileges for the combined configurations of Figures 4a and 4b

(*u*1, *r*, *o*1), (*u*1, *w*, *o*1), (*u*1, *r*, *o*2), (*u*2, *r*, *o*1), (*u*2, *r*, *o*2), (*u*2, *w*, *o*2), (*u*2, *r*, *o*3), (*u*2, *w*, *o*3), (u2, r, o4), (u2, w, o4)

Just as access rights to perform read/write operations on resource objects are defined in terms of associations, so too are capabilities to perform administrative operations on policy elements and relations. In contrast to non-administrative access rights, where resource operations are synonymous with the access rights needed to carry out those operations (e.g., a "read" operation corresponding to an "r" access right), the authority stemming from one or more administrative access rights may be required for an administrative operation. Administrative access rights to perform an administrative operation maybe explicitly divided into two parts, as denoted by "from" and "to" suffixes.

For example, in the in context of Figure 4 we could create two associations Bob---{create ooa-from}---Bob Home and Division---{create ooa-to}---Projects, meaning that the intersection of users in Bob and Division may create "object to object attribute assignments" (ooa) from objects in Bob Home to object attributes in Projects. Remember that the set of referenced policy elements in the third term of an association (*at*) is dependent on the access rights in ars. As such, the absolute mean of the two associations is that user u2 can create assignments from o2, o3, or o4 to Projects, Project1, or Project2.

4.2.2 Prohibitions (Denies)

In addition to assignments and associations, NGAC includes three types of prohibition relations: user-deny, user attribute-deny, and process-deny. In general, deny relations specify privilege exceptions. We respectively denote a user-based deny, user attribute-based deny, and process-based deny relation by u_deny(*u*, *ars*, *pe*), ua_deny(*ua*, *ars*, *pe*), and p_deny(*p*, *ars*, *pe*), where *u* is a user, *ua* is a user attribute, *p* is a process, *ars* is an access right set, and *pe* is a policy element used as a referent for itself and the policy elements contained by the policy element. The respective meanings of these relations are that user *u*, users in *ua*, and process *p* cannot execute access rights in *ars* on policy elements in *pe*. User-deny relations and user attribute-deny relations can be created directly by an administrator or dynamically as a consequence of an obligation (see Section 4.2.3). An administrator, for example, could impose a condition where no user is able to alter their own Tax Return, in spite of the fact that the user is assigned to an IRS Auditor user attribute with capabilities to read/write all tax returns. When created through an obligation, user-deny and user attribute-deny relations can take on dynamic policy conditions. Such conditions can, for example, provide support for separation of duty policies (if a user executed capability x, that user would be immediately precluded from being able to perform capability y). In addition, the policy element component of each prohibition relation can be specified as its complement, denoted by ¬. The respective meaning of u_deny(*u*, *ars*, ¬*pe*), ua_deny(*ua*, *ars*, ¬*pe*), and p_deny(*p*, *ars*, ¬*pe*) is that the user *u,* and any user assigned to *ua*, and process *p* cannot execute the access rights in *ars* on policy elements not in *pe*.

Process-deny relations are exclusively created using obligations. Their primary use is in the enforcement of confinement conditions (e.g., if a process reads Top Secret data, preclude that process from writing to any object not in Top Secret).

4.2.3 Obligations

Obligations consist of a pair (*ep*, *r*) (usually expressed as **when** *ep* **do** *r*) where *ep* is an *event pattern* and *r* is a sequence of administrative operations, called a *response*. The event pattern specifies conditions that if matched by the context surrounding a process's successful execution of an operation on an object (an event), cause the administrative operations of the associated response to be immediately executed. The context may pertain to and the event pattern may specify parameters like the user of the process, the operation executed, and the attribute(s) of the object.

Obligations can specify operational conditions in support of history-based policies and data services.

Included among history-based policies are those that prevent leakage of data to unauthorized principals. Consider, for example the "Project Access" policy depicted in Figure 4(a). Although this policy suggests that only Group2 users can read Gr2-Secrets, data in Gr2-Secrets can indeed be leaked to Group1 users. Specifically, u2 or one of u2's processes can read o3, and subsequently write its content to o2, thereby providing u1 the capability to read the content of o3. Such leakage can be prevented with the following obligation:

When any process *p* performs (*r*, *o*) where *o*→Gr2-Secret **do**
create p-deny(*p*, {*w*}, ¬Gr2-Secret)

The effect of this obligation will prevent a process (and its user) from reading an object in Gr2-Secret and subsequently writing its content to an object in a different container (not in Gr2-Secret).

Other history-based policies include conflict of interest (if a user reads information from a sensitive data set, that user is prohibited from reading data from a second data set) and Work Flow (approving (writing to a field of)) a work item enables a second user to read and approve the work item).

4.3 NGAC Decision Function

The NGAC access decision function controls accesses in terms of processes. The user on whose behalf the process operates must hold sufficient authority over the policy elements involved. The function process_user(*p*) denotes the user associated with process *p*.

Access requests are of the form (*p*, *op*, *argseq*), where *p* is a process, *op* is an operation, and *argseq* is a sequence of one or more arguments, which is compatible with the scope of the operation. The access decision function to determine whether an access request can be granted requires a mapping from an operation and argument sequence pair to a set of access rights and policy element pairs (i.e., {(*ar*, *pe*)}) the process's user must hold for the request to be granted.

When determining whether to grant or deny an access request, the

authorization decision function takes into account all privileges and restrictions (denies) that apply to a user and its processes, which are derived from relevant associations and denies, giving restrictions precedence over privileges:

A process access request (p, op, $argseq$) with mapping (op, $argseq$)$\rightarrow${(ar, pe)}) is granted iff for each (ar_i, pe_i) in {(ar, pe)}, there exists a privilege (u, ar_i, pe_i) where u = process_user(p), and (ar_i, pe_i) is not denied for either u or p.

In the context of Figure 4, an access request may be (p, read, o1) where p is u1's process. The pair (read, o1) maps to (r, o1). Because there exists a privilege (u1, r, o1) in table 3 and (r, o1) is not denied for u1 or p, the access request would be granted. Assume the existence of associations Division---{create ooa-to}---Projects, and Bob---{create ooa-from}---Bob Home in the context of Figure 4, and an access request (p, assign, <o4, Project1>) where p is u2's process. The pair (assign, <o4, Project1>) maps to {(create ooa-from, o4), (create ooa-to, Project1)}. Because privileges (u2, create ooa-from, o4) and (u2, create ooa-to, Project1) would exist under the assumption, and (create ooa-from, o4) and (create ooa-to, Project1) are not denied for u2 or p, the request would be granted.

4.4 Delegation

The question remains, how are administrative capabilities created? The answer begins with a superuser with capabilities to perform all administrative operations on all access control data. The initial state consists of an NGAC configuration with empty data elements, attributes, and relations. A superuser either can directly create administrative capabilities or more practically can create administrators and delegate to them capabilities to create and delete administrative privileges. Delegation and rescinding of administrative capabilities is achieved through creating and deleting associations. The principle followed for allocating access rights via an association is that the creator of the association must have been allocated the access right over the attribute in question (as well as the necessary create-assoc-from and create-assoc-to rights) in order to delegate them. The strategy enables a systematic approach to the creation of administrative attributes and delegation of administrative capabilities, beginning with a superuser and ending with users with administrative and data service capabilities.

4.5 NGAC Administrative Commands and Routines

Access requests bearing administrative operations can create and destroy basic elements, containers and relations. Each administrative operation corresponds on a one-to-one basis to an administrative routine, which uses the sequence of arguments in the access request to perform the access. Each administrative operation is carried out through one or more primitive administrative commands. NGAC defines the complete set of administrative commands and their behavior in detail. The definitions specify the preconditions that need to exist for the effect of a command to occur, and the specific effect that the command has on the contents of NGAC's Policy Information Point (policies and attributes store).

The access decision function grants the access request (and initiation of the respective administrative routine) only if the process holds all prohibition-free access rights over the items in the argument sequence needed to carry out the access. The administrative routine, in turn, uses one or more administrative commands to perform the access. Administrative commands and routines are thus the means by which policy specifications and attributes are formed.

Consider the administrative command CreateAssoc shown below, which specifies the creation of an association. The preconditions here stipulate membership of the x, y, and z parameters respectively to the user attributes (UA), access right sets (ARs), and attributes (AT) elements of the model. The body describes the addition of the tuple (x, y, z) to the set of associations (ASSOC) relation, which changes the state of the relation to ASSOC′.

```
createAssoc (x, y, z)
  x ∈ UA ⋀ y ∈ ARs ⋀ z ∈ AT ⋀ (x, y, z) ∉ ASSOC
    {
    ASSOC′ = ASSOC ⋃ {(x, y, z)}
    }
```

An administrative routine consists mainly of a parameterized interface and a sequence of administrative command invocations. Each formal parameter of an administrative routine can serve as an argument in any of the administrative command invocations that make up the body of the routine. Administrative routines are used in a variety of ways. Although an administrative routine must be in place on a one-to-one basis to carry out an administrative operation, they can also be used to carry out more complex administrative tasks comprising of a sequence of administrative actions.

Consider the following administrative routine that creates a "file management" user in the context of Figure 4b. The routine assumes the pre-existence of the user attribute "Users" assigned to the "File Management" policy class shown in Figure 4b.

```
create-file-mgmt-user(user-id, user-name, user-home) {
    createUAinUA(user-name, Users);
    createUinUA(user-id, user-name);
    createOAinPC(user-home, File Management);
    createAssoc(user-name, {r, w}, user-home);
    createAssoc(user-name, {create-o-to, delete-o-from}, user-
        home);
    createAssoc(user-name, {create-ooa-from, create-ooa-to,
        delete-ooa-from, create-oaoa-from, create-oaoa-to,
        delete-oaoa-from}, user-home);
    createAssoc(user-name, {create-assoc-from, delete-assoc-
        from}, Users);
    createAssoc(user-name, {create-assoc-to, delete-assoc-to, r-
        allocate, w-allocate}, user-home);}
```

This routine with parameters (u1, *Bob* and *Bob Home*) could have been used to create "file management" data service capabilities for user u1 already in Figure 4b. Through the routine the user attribute "Bob" is created and assigned to "Users", and user u1 is created and assigned to "Bob". In addition, the object attribute "Bob Home" is created and assigned to policy class "File Management". In addition, user u1 is delegated administrative capabilities to create, organize, and delete object attributes (presented folders) in Bob Home, and u1 is provided with capabilities to create, read, write, and delete objects that correspond to files and place those files into his folders. Finally, u1 is provided with discretionary capabilities to "grant" to other users in the "Users" container capabilities to perform read/write operations on individual files or to all files in a folder in his

Home.

4.6 Arbitrary Data Service Operations

NGAC recognizes administrative operations for the creation and management of its data elements and relations that represent policies and attributes, and basic input and output operations (e.g., read and write) that can be performed on objects that represent data service resources. In accommodating data services, NGAC may establish and provide control over other types of operations, such as send, submit, approve, and create folder. However, it does not necessarily need to do so. This is because the basic data service capabilities to consume, manipulate, manage, and distribute access rights on data can be attained as combinations of read/write operations on data and administrative operations on data elements, attributes, and relations. For example, the create-file-mgmt-user routine specified above provides a user with capabilities to create and manage files and folders, and control and share access to objects in the user's home directory.

4.7 NGAC Functional Architecture

NGAC's functional architecture (shown in Figure 5), like XACML's, encompasses four layers of functional decomposition: Enforcement, Decision, Administration, and Access Control Data, and involves several components that work together to bring about policy-preserving access and data services.

Among these components is a PEP that traps application requests. An access request includes a process id, user id, operation, and a sequence of one or more operands mandated by the operation that pertain to either a data resource or an access control data element or relation. Administrative operational routines are implemented in the PAP and read/write routines are implemented in the RAP.

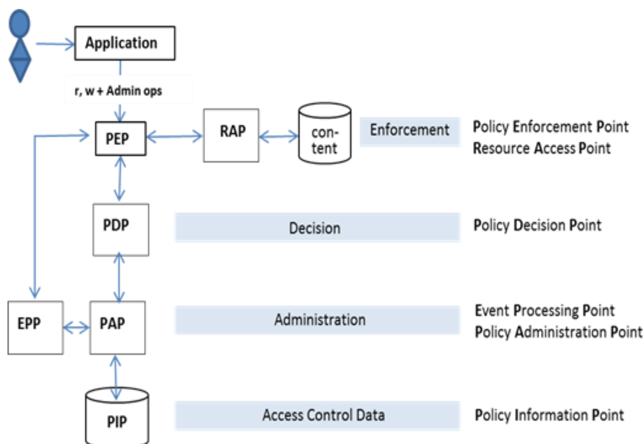

Figure 5: NGAC Standard Functional Architecture

To determine whether to grant or deny, the PEP submits the request to a PDP. The PDP computes a decision based on current configuration of data elements and relations stored in the PIP, via the PAP. Unlike the XACML architecture, the access request information from an NGAC PEP together with the NGAC relations (selectively retrieved by the PDP) provide the full context for arriving at a decision. The PDP returns a decision of grant or deny to the PEP. If access is granted and the operation was read/write, the PDP also returns the physical location where the object's content resides, the PEP issues a command to the appropriate RAP to execute the operation on the content, and the RAP returns the status. In the case of a read operation, the RAP also returns the data type of the content (e.g., PowerPoint) and the PEP invokes the correct data service application for its consumption. If the request pertained to an administrative operation and the decision was grant, the PDP issues a command to the PAP for execution of the operation on the data element or relation stored in the PIP, and the PAP returns the status to the PDP, which in turn relays the status to the PEP. If the returned status by either the RAP or PAP is "successful", the PEP submits the context of the access to the Event Processing Point (EPP). If the context matches an event pattern of an obligation, the EPP automatically executes the administrative operations of that obligation, potentially changing the access state. Note that NGAC is data type agnostic. It perceives accessible entities as either data or access control data elements or relations, and it is not until after the access process is completed that the actual type of the data matters to the application.

5. COMPARISON OF XACML AND NGAC

XACML is similar to NGAC insofar as they both provide flexible, mechanism-independent representations of policy rules that may vary in granularity, and they employ attributes in computing decisions. However, XACML and NGAC differ significantly in their expression of policies, treatment of attributes, computation of decisions, and representation of requests. In this section, we analyze these similarities and differences with respect to the degree of separation of access control logic from proprietary operating environments and four ABAC considerations identified in NIST SP 800-162: operational efficiency, attribute and policy management, scope and type of policy support, and support for administrative review and resource discovery. For the purposes of comparison we normalize some XACML and NGAC terminology.

5.1 Separation of Access Control Logic from Operating Environments

Both XACML and NGAC achieve separation of access control logic of data services from proprietary operating environments, but to different degrees. XACML's separation is partial. XACML does not envisage the design of a PEP that is data service agnostic. An XACML deployment consists of one or more data services, each with an operating environment-dependent PEP, and operating environment-dependent operational routines and resource types, that share a common PDP and access control information consisting of policies and attributes. In other words, a PEP under the XACML architecture is tightly coupled to a specific operating environment for which it was designed to enforce access.

The degree of separation that can be achieved by NGAC is near complete. Although an NGAC deployment could include a PEP with an Application Programming Interface (API) that recognizes operating environment-specific operations (e.g., send and forward operations for a messaging system), it does not necessarily need to do so. NGAC includes a standard PEP with an API that supports a set of generic, operating environment-agnostic operations (read, write, create, and delete policy elements and relations). This API enables a common, centralized PEP to be implemented to serve the requests of multiple applications. Although the generic operations may not meet the requirements of every application (e.g., transactions that perform computations on attribute values), calls from many applications can be accommodated. This includes operations that generically pertain to consumption, alteration, management, and sharing of data resources. As a consequence, NGAC can completely displace the need for an access control mechanism of an operating environment in that through the same

PEP API, set of operations, access control data elements and relations, and functional components, arbitrary data services can be delivered to users, and arbitrary, mission-tailored access control policies can be expressed and enforced over executions of application calls.

5.2 Operational Efficiency

An XACML request is a collection of attribute name, value pairs for the subject (user), action (operation), resource, and environment. XACML identifies relevant trusted and untrusted access policies and rules for computing decisions through a search for Targets (conditions that match the attributes of the request). Because multiple Policies in a PolicySet and/or multiple Rules in a Policy may produce conflicting access control decisions, XACML resolves these differences by applying collections of potentially several rule and policy combining algorithms. If the attributes are not sufficient for the evaluation of an applicable rule, the PDP may search for additional attributes. The entire process involves converting a PEP request into an XACML canonical form, collecting attributes, matching target conditions, computing rules, (optionally) issuing administrative requests (for determining a chain of trust for applicable untrusted access policies), resolving conflicts, and converting an XACML access decision to a PEP specific response, involving at least two data stores.

NGAC is inherently more efficient. An NGAC request is composed of a process id, user id, operation, and a sequence of one or more operands mandated by the operation that affects either a resource or access control data. NGAC identifies relevant policies, attributes and prohibitions, by reference (through relations) when computing a decision. Like XACML, NGAC combines policies. However, it does not compute and then combine multiple local decisions, but rather takes multiple policies into consideration when determining the existence of an appropriate privilege. All information necessary in computing an access decision resides in a single database. NGAC does not include a context handler for converting requests and decisions to and from its canonical form or for retrieving attributes. Although considered a component of its access control process, obligations do not come into play until after a decision has been rendered and data has been successfully altered or consumed.

5.3 Attribute and Policy Management

Because XACML is implemented in XML, it inherits XML's benefits and drawbacks. The flexibility and expressiveness of XACML, while powerful, make the specification of policy complex and verbose [12]. Applying XACML in a heterogeneous environment requires fully specified data type and function definitions that produce a lengthy textual document, even if the actual policy rules are trivial. In general, platform-independent policies expressed in an abstract language are difficult to create and maintain by resource administrators [14]. Unlike XACML, NGAC is a relations-based standard, which avoids the syntactic and semantic complexity in defining an abstract language for expressing platform-independent policies [12]. NGAC policies are expressed in terms of configuration elements that are maintained at a centralized point and typically rendered and manipulated graphically. For example, to describe hierarchical relations and inheritance properties of attributes, NGAC requires only the addition of links representing assignment relations between them; in XACML, relations need to be inserted in precise syntactic order.

XACML's ability to specify policies as logical conditions provides policy expression efficiency. Consider the XACML Policy specified in Section 3.4 and the attribute names, values and value assignments in table 1. NGAC could express this same policy and authorization state using enumerated attributes, assignments, and associations. See [21] for a detailed configuration. The NGAC eqivelent policy would include five association relations, while XACML uses just three rules. As the number of Wards that are considered by the policy increases, so will the number of NGAC association relations, but the number of XACML rules will always remain the same. Recognize that for this policy, the number of attributes and attribute assignments will always be the same for XACML and NGAC regardless of the number of Wards considered. On the other hand, for some policies, the number of XACML attribute assignments can far exceed those necessary for an NGAC equivalent policy. Consider the TCSEC MAC Policy [3, 5] expressed using XACML rules and NGAC relations. For the XACML TCSEC MAC policy to work (using static rules), all resources whether classified or unclassified are required to be assigned to attributes to prevent classified data from being leaked to unclassified data. For the NGAC TCSEC MAC policy to work (using obligations (e.g., **when** any process p performs (read, o) where o→Top Secret **do** create p-deny(p, {write}, ¬Top Secret)), only objects that are actually classified (e.g., Secret and Top Secret) are required to be assigned to attributes. See [21] for detailed XACML and NGAC expressions of the TCSEC MAC policy.

Proper enforcement of data resource policies is dependent on administrative policies. This is especially true in a federated or collaborative environment, where governance policies require different organizational entities to have different responsibilities for administering different aspects of policies and their dependent attributes.

XACML and NGAC differ dramatically in their ability to impose policy over the creation and modification of access control data (attributes and policies). NGAC manages attributes and policies through a standard set of administrative operations, applying the same enforcement interface and decision making function as it uses for accessing data resources. XACML does not recognize administrative operations, but instead manages policy content through a Policy Administration Point (PAP) with an interface that is different from that for accessing data resources. XACML provides support for decentralized administration of some of its access policies. However the approach is only a partial solution in that it is dependent on trusted and untrusted policies, where trusted policies are assumed valid, and their origin is established outside the delegation model. Furthermore, the XACML delegation model does not provide a means for imposing policy over modification of access policies, and offers no direct administrative method for imposing policy over the management of its attributes.

NGAC enables a systematic and policy-preserving approach to the creation of administrative roles and delegation of administrative capabilities, beginning with a single administrator and an empty set of access control data, and ending with users with data service, policy, and attribute management capabilities. NGAC provides users with administrative capabilities down to the granularity of a single configuration element, and can deny users administrative capabilities down to the same granularity.

5.4 Scope and Type of Policy Support

Although data resources may be protected under a wide variety of different access policies, these policies can be generally categorized as either discretionary or mandatory controls. Discretionary access control (DAC) is an administrative policy that permits system users to allow or disallow other users' access to objects that are placed under their control [15]. Although XACML can theoretically implement DAC policies, it is not efficient. Consider the propagation feature of DAC. DAC permits owners/creators of objects to grant some or all of their capabilities to other users, and the grantees can further propagate those capabilities on to other users. The overall DAC feature to grant privileges to another user and the ability of the grantee to propagate those privileges cannot be supported in XACML syntax using "Access Policies" alone.

Therefore, all the capabilities of the owner/creator of an object together with administrative capabilities to grant those privileges have to be specified using a Trusted Administrative policy. The capabilities held by owner/creator can be captured by designating the owner/creator of the object as the "access-subject", and the administrative capability to grant privileges to others can be captured by designating the owner/creator as a delegate in that policy type. The creation of this trusted administrative policy enables creation of derived administrative policies with the owner/creator as the policy issuer with the specified set of capabilities. The specification of a "delegate" in this derived administrative policy (not trusted) provides a means for the owner/creator to grant capabilities to other users, as well as the ability for the grantee to propagate those capabilities to other users. However, while it is theoretically possible to implement DAC by leveraging XACML's delegation feature, this approach involves significant administrative overhead. The solution requires the specification of a trusted administrative policy and a set of derived administrative policies for every object owner/creator, and for all grantees of the capabilities.

Conversely, NGAC has a flexible means of providing users with administrative capabilities to include those necessary for the establishment of DAC policies, as shown in section 5.4.

In contrast to DAC, mandatory access control (MAC) enables ordinary users' capabilities to execute resource operations on data, but not administrative capabilities that may influence those capabilities. MAC policies unavoidably impose rules on users in performing operations on resource data. MAC policies can be further characterized as controls that accommodate confinement properties to prevent indirect leakage of data to unauthorized users, and those that do not.

Expression of non-confinement MAC policies is perhaps XACML's strongest suit. XACML can specify rules and other conditions in terms of attribute values of varying types. There are undoubtedly certain policies that are expressible in terms of these rules that cannot be easily accommodated by NGAC. This is especially true when treating attribute values as integers. For example, to approve a purchase request may involve adding a person's credit limit to their account balance. Furthermore, XACML takes environmental attributes into consideration in expressing policy, and NGAC does not. However, there are some non-confinement MAC properties, such as a variety of history-based policies that NGAC can express, and XACML cannot. Although XACML has been shown to be capable of expressing aspects of standard RBAC [1] through an XACML profile [16], the profile falls short of demonstrating support for dynamic separation of duty, a key feature used for accommodating the principle of least privilege, and separation of duty, a key feature for combatting fraud. Annex B of Draft standard Next Generation Access Control – Generic Operations and Data Structures (NGAC-GOADS) [20] demonstrates NGAC support for all aspects of the RBAC standard.

In addition to static and dynamic separation of duty, NGAC has shown support for history-based separation of duty [7]. In their seminal paper on the subject [19], Simon and Zurko describe history-based separation of duty as the most accommodating form of separation of duty, subsuming the policy objectives of other forms.

In contrast to NGAC, XACML does not recognize the capabilities of a process independent of the capabilities of its user. Without such features, XACML is ill equipped to support confinement and as such is arguably incapable of enforcement of a wide variety of policies. These confinement-dependent policies include some instances of role-based access control (RBAC), e.g., "only doctors can read the contents of medical records", originator control (ORCON) [10] and Privacy, e.g., "I know who can currently read my data or personal information", conflict of interest [4], e.g., "a user with knowledge of information within one dataset cannot read information in another dataset", or Multi-level Security [3]. [5]. Through imposing process level controls in conjunction with obligations, NGAC has shown [7] support for these and other confinement-dependent MAC controls.

5.5 Administrative Review and Resource Discovery

A desired feature of access controls is review of capabilities (*op*, *o*) of users and access control entries (*u*, *op*) of objects, where *u* is a user, *op* is an operation, and *o* is an object [15] [11]. These features are often referred to as "before the fact audit" and resource discovery. "Before the fact audit" is one of RBAC's most prominent features [18]. Being able to discover or see a newly accessible resource is an important feature of any access control system. NGAC supports efficient algorithms for both per-user and per-object review. Per-object review of access control entries is not as efficient as a pure access control list (ACL) mechanism, and per-user review of capabilities is not as efficient as that of RBAC. However, this is due to NGAC's consideration of conducting review in a multi-policy environment. NGAC can efficiently support both per-object and per-user reviews of combined policies, where RBAC and ACL mechanisms can do only one type of review efficiently, and rule-based mechanisms such as XACML, although able to combine policies, cannot do either efficiently. In other words, there exists no method of determining the authorization state without testing all possible decision outcomes.

6. REFERENCES

[1] Information technology – Role-Based Access Control (RBAC), INCITS 359-2004, American National Standard for Information Technology, American National Standards Institute, 2004.

[2] Information technology - Next Generation Access Control - Functional Architecture (NGAC-FA), INCITS 499-2013, American National Standard for Information Technology, American National Standards Institute, March 2013.

[3] D. Bell and L. La Padula. Secure computer systems: unified

exposition and MULTICS. Report ESD-TR-75-306, The MITRE Corporation, Bedford, Massachusetts, March 1976.

[4] D.F.C. Brewer and M.J. Nash, “The Chinese Wall Security Policy,” 1989 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 1-3, 1989, pp. 206-214. http://dx.doi.org/10.1109/SECPRI.1989.36295 [accessed 11/15/15]

[5] DoD Computer Security Center, Trusted Computer System Evaluation Criteria (December 1985).

[6] D.F. Ferraiolo, S.I. Gavrila, V.C. Hu, and D.R. Kuhn, “Composing and Combining Policies Under the Policy Machine,” Tenth ACM Symposium on Access Control Models and Technologies (SACMAT ‘05), Stockholm, Sweden, 2005, pp. 11-20.

[7] D.F. Ferraiolo, V. Atluria, and S.I. Gavrila, “The Policy Machine: A Novel Architecture and Framework for Access Control Policy Specification and Enforcement,” Journal of Systems Architecture, vol. 57, no. 4, pp. 412-424, April 2011. http://dx.doi.org/10.1016/j.sysarc.2010.04.005 [accessed 11/15/15]

[8] D. Ferraiolo, S. Gavrila, and W. Jansen, National Institute of Standards and Technology (NIST) IR-7987 Revision 1, “Policy Machine: Features, Architecture, and Specification,” October 2015. http://nvlpubs.nist.gov/nistpubs/ir/2015/NIST.IR.7987r1.pdf

[9] D. Ferraiolo, S. Gavrila, and W. Jansen, “On the Unification of Access Control and Data Services,” in Proc. IEEE 15th International Conference of Information Reuse and Integration, 2014, pp. 450 – 457. http://csrc.nist.gov/pm/documents/ir2014_ferraiolo_final.pdf

[10] R. Graubart, On the need for a third form of access control, in: Proc. National Computer Security Conference, 1989, pp. 296 –304.

[11] V.C. Hu, D.F. Ferraiolo, and D.R. Kuhn, National Institute of Standards and Technology (NIST) Interagency Report (IR) 7316, “Assessment of Access Control Systems,” September 2006. http://csrc.nist.gov/publications/nistir/7316/NISTIR-7316.pdf

[12] V. C. Hu, D.F. Ferraiolo, and K. Scarfone, Access Control Policy Combinations for the Grid Using the Policy Machine, Cluster Computing and the Grid, 2007, pp. 225-232.

[13] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, National Institute of Standards and Technology (NIST) SP-800-162, Guide to Attribute Based Access Control (ABAC) Definition and Considerations, January 2014. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-162.pdf

[14] M. Lorch et al, “First Experience Using XACML for Access Control in Distributed Systems, ACM Workshop on XML Security, Fairfax, Virginia, 2003.

[15] Guide to Understanding Discretionary Access Control in Trusted Systems, NCSC-TG-003, Version-1, National Computer Security Center, Fort George G. Meade, USA, September 30, 1987, 29 pp. http://csrc.nist.gov/publications/secpubs/rainbow/tg003.txt

[16] XACML Profile for Role Based Access Control (RBAC), Committee Draft 01, February 2004.

[17] The eXtensible Access Control Markup Language (XACML), Version 3.0, OASIS Standard, January 22, 2013. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf

[18] 2010 Economic Analysis of Role-Based Access Control, RTI Number 0211876, Research Triangle Institute, December 2010.

[19] R. Simon, M. Zurko, Separation of duty in role based access control environments, Proc. New Security Paradigms Workshop, 1997.

[20] Information technology – Next Generation Access Control – Generic Operations and Data Structures, INCITS 526, American National Standard for Information Technology, American National Standards Institute, to be published.

[21] D. F. Ferraiolo, R. Chandramouli, V. Hu, and R. Kuhn, National Institute of Standards and Technology DRAFT (NIST) SP-800-178, A Comparison of Attribute Based Access Control (ABAC) Standards for Data Services, December 2015. http://csrc.nist.gov/publications/drafts/800178/sp800_178_draft.pdf