Core and Hierarchical Role Based Access Control (RBAC) profile of XACML, Version 2.0

Committee Draft 01, 30 September 2004

Document identifier:

access_control-xacml-2.0-rbac_profile1-spec-cd-01

Location:

http://docs.oasis-open.org/xacml/access_control-xacml-2.0-rbac_profile1-spec-cd-01.pdf

Editor:

Anne Anderson, Sun Microsystems (anne.anderson@sun.com)

Abstract:

This specification defines a profile for the use of XACML in expressing policies that use role based access control (RBAC). It extends the XACML Profile for RBAC Version 1.0 to include a recommended AttributeId for roles, but reduces the scope to address only “core” and “hierarchical” RBAC. This specification has also been updated to apply to XACML 2.0.

Status:

This version of the specification is an approved Committee Draft within the OASIS Access Control TC.

Access Control TC members should send comments on this specification to the xacml@lists.oasis-open.org list. Others may use the following link and complete the comment form: http://oasis-open.org/committees/comments/form.php?wg_abbrev=xacml.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Access Control TC web page (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml).

For any errata page for this specification, please refer to the Access Control TC web page (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml).

Table of Contents

1 Introduction (non-normative)

This specification defines a profile for the use of the OASIS eXtensible Access Control Markup Language (XACML) [XACML] to meet the requirements for "core" and "hierarchical" role based access control (RBAC) as specified in [ANSI-RBAC]. Use of this profile requires no changes or extensions to standard XACML Versions 1.0, 1.1, or 2.0 (although examples must be modified slightly for Versions 1.0 and 1.1). It extends the XACML Profile for RBAC Version 1.0 [RBAC-V1] to include a recommended XACML `AttributeId` for roles, but reduces the scope to address only "core" and "hierarchical" RBAC. The specification has also been updated for XACML 2.0.

This specification begins with a non-normative explanation of the building blocks from which the RBAC solution is constructed. A full example illustrates these building blocks. The specification then discusses how these building blocks may be used to implement the various elements of the RBAC model presented in [ANSI-RBAC]. Finally, the normative section of the specification describes compliant uses of the building blocks in implementing an RBAC solution.

This specification assumes the reader is somewhat familiar with XACML. A brief overview sufficient to understand these examples is available in [XACMLIntro]. An introduction to the RBAC model is available in [RBACIntro].

1.1 Notation

In order to improve readability, the examples in this specification assume use of the following XML Internal Entity declarations:

```
^lt;!ENTITY xml "http://www.w3.org/2001/XMLSchema#">
^lt;!ENTITY rule-combine
 "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:">
^lt;!ENTITY policy-combine
 "urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:">
^lt;!ENTITY function "urn:oasis:names:tc:xacml:1.0:function:">
^lt;!ENTITY subject-category
 "urn:oasis:names:tc:xacml:1.0:subject-category:">
^lt;!ENTITY subject "urn:oasis:names:tc:xacml:1.0:subject:">
^lt;!ENTITY role "urn:oasis:names:tc:xacml:2.0:subject:role">
^lt;!ENTITY roles "urn:example:role-values:">
^lt;!ENTITY resource "urn:oasis:names:tc:xacml:1.0:resource:">
^lt;!ENTITY action "urn:oasis:names:tc:xacml:1.0:action:">
^lt;!ENTITY actions "urn:oasis:names:tc:xacml:2.0:actions:">
^lt;!ENTITY environment "urn:oasis:names:tc:xacml:1.0:environment:">
```

For example, "`&xml;string`" is equivalent to "`http://www.w3.org/2001/XMLSchema#string`".

1.2 Terminology

The key words *must, must not, required, shall, shall not, should, should not, recommended, may,* and *optional* in this document are to be interpreted as described in IETF RFC 2119 [RFC2119].

attribute - in this specification, the term "attribute" refers to an XACML `<Attribute>`. An XACML `<Attribute>` is an element in an XACML Request having among its components an attribute name identifier, a data type identifier, and an attribute value. Each `<Attribute>` is associated either with one of the subjects (Subject Attribute), the protected resource (Resource Attribute), the action to be taken on the resource (Action Attribute), or the environment of the Request (Environment Attribute). Attributes are referenced in a policy by using an `<AttributeSelector>` (an XPath expression) or one of the following: `<SubjectAttributeDesignator>`, `<ResourceAttributeDesignator>`, `<ActionAttributeDesignator>`, or `<EnvironmentAttributeDesignator>`.

HasPrivilegesOfRole policy – an optional type of `<Policy>` that can be included in a Permission `<PolicySet>` to allow support queries asking if a subject "has the privileges of" a specific role. See Section 2.5*: HasPrivilegesOfRole Policies and Requests.*

junior role – in a role hierarchy, Role A is *junior* to Role B if Role B inherits all the permissions associated with Role A.

multi-role permissions – a set of permissions for which a user must hold more than one role simultaneously in order to gain access.

PDP - Policy Decision Point. An entity that evaluates an access request against one or more policies to produce an access decision.

permission – the ability or right to perform some action on some resource, possibly only under certain specified conditions.

PPS – Permission `<PolicySet>`. See Section 1.5: *Policies*.

RBAC – role based access control. A model for controlling access to resources where permitted actions on resources are identified with roles rather than with individual subject identities.

Role Enablement Authority - an entity that assigns role attributes and values to users or enables role attributes and values during a user's session.

RPS – Role `<PolicySet>`. See Section 1.5: *Policies*.

role – a job function within the context of an organization that has associated semantics regarding the authority and responsibility conferred on the user assigned to the role [ANSI-RBAC].

senior role – in a role hierarchy, Role A is *senior* to Role B if Role A inherits all the permissions associated with Role B.

policy – a set of rules indicating which subjects are permitted to access which resources using which actions under which conditions.

1.3 Scope

Role based access control allows policies to be specified in terms of subject roles rather than strictly in terms of individual subject identities. This is important for scalability and manageability of access control systems.

The policies specified in this profile can answer three types of questions:

1. If a subject has roles R_1 , R_2, ... R_n enabled, can subject X access a given resource using a given action?
2. Is subject X allowed to have role R_i enabled?
3. If a subject has roles R_1 , R_2, ... R_n enabled, does that mean the subject will have permissions associated with a given role R'? That is, is role R' either equal to or *junior* to any of roles R_1 , R_2, ... R_n?

The policies specified in this profile do not answer the question “What set of roles does subject X have?” That question must be handled by a Role Enablement Authority, and not directly by an XACML PDP. Such an entity may make use of XACML policies, but will need additional information. See Section 3: *Assigning and Enabling Role Attributes* for more information about Role Enablement Authorities.

The policies specified in this profile assume all the roles for a given subject have already been enabled at the time an authorization decision is requested. They do not deal with an environment in which roles must be enabled dynamically based on the resource or actions a subject is attempting to perform. For this reason, the policies specified in this profile also do not deal with static or dynamic “Separation of Duty” (see [ANSI-RBAC]). A future profile may address the requirements of this type of environment.

1.4 Role

In this profile, roles are expressed as XACML Subject Attributes. There are two exceptions: in a Role Assignment `<PolicySet>` or `<Policy>` and in a HasPrivilegesOfRole `<Policy>`, the role appears as a Resource Attribute. See Section 2.5: *HasPrivilegesOfRole Policies and Requests* and Section 3: *Assigning and Enabling Role Attributes* for more information.

Role attributes may be expressed in either of two ways, depending on the requirements of the application environment. In some environments there may be a small number of "role attributes", where the name of each such attribute is some name indicating "role", and where the value of each such attribute indicates the name of the role held. For example, in this first type of environment, there may be one "role attribute" having the `AttributeId` "`&role;`" (this profile recommends use of this identifier). The possible roles are values for this one attribute, and might be "`&roles;officer`", "`&roles;manager`", and "`&roles;employee`". This way of expressing roles works best with the XACML way of expressing policies. This method of identifying roles is also most conducive to interoperability.

Alternatively, in other application environments, there may be a number of different attribute identifiers, each indicating a different role. For example, in this second type of environment, there might be three attribute identifiers: "`urn:someapp:attributes:officer-role`", "`urn:someapp:attributes:manager-role`", and "`urn:someapp:attributes:employee-role`". In this case the value of the attribute may be empty or it may contain various parameters associated with the role. XACML policies can handle roles expressed in this way, but not as naturally as in the first way.

XACML supports multiple subjects per access request, indicating various entities that may be involved in making the request. For example, there is usually a human user who initiates the request, at least indirectly. There are usually one or more applications or code bases that generate the actual low-level access request on behalf of the user. There is some computing device on which the application or code base is executing, and this device may have an identity such an IP address. XACML identifies each such `Subject` with a `SubjectCategory` xml attribute that indicates the type of subject being described. For example, the human user has a `SubjectCategory` of `&subject-category;access-subject` (this is the default category); the application that generates the access request has a `SubjectCategory` of `&subject-category;codebase` and so on. In this profile, a role attribute may be associated with any of the categories of subjects involved in making an access request.

1.5 Policies

In this profile, four types of policies are specified.

1. **Role `<PolicySet>`** or **RPS** : a `<PolicySet>` that associates holders of a given role attribute and value with a Permission `<PolicySet>` that contains the actual permissions associated with the given role. The `<Target>` element of a Role `<PolicySet>` limits the applicability of the `<PolicySet>` to subjects holding the associated role attribute and value. Each Role `<PolicySet>` references a single corresponding Permission `<PolicySet>` but does not contain or reference any other `<Policy>` or `<PolicySet>` elements.
2. **Permission `<PolicySet>`** or **PPS**: a `<PolicySet>` that contains the actual permissions associated with a given role. It contains `<Policy>` elements and `<Rules>` that describe the resources and actions that subjects are permitted to access, along with any further conditions on that access, such as time of day. A given Permission `<PolicySet>` may also contain references to Permission `<PolicySet>`s associated with other roles that are *junior* to the given role, thereby allowing the given Permission `<PolicySet>` to inherit all permissions associated with the role of the referenced Permission `<PolicySet>`. The `<Target>` element of a Permission `<PolicySet>`, if present, must not limit the subjects to which the `<PolicySet>` is applicable.
3. **Role Assignment `<Policy>`** or **`<PolicySet>`:** a `<Policy>` or `<PolicySet>` that defines which roles can be enabled or assigned to which subjects. It may also specify restrictions on combinations of roles or total number of roles assigned to or enabled for a given subject. This type of policy is used by a Role Enablement Authority. Use of a Role Assignment `<Policy>` or `<PolicySet>` is optional.
4. **HasPrivilegesOfRole <Policy>:** a `<Policy>` in a Permission `<PolicySet>` that supports requests asking whether a subject has the privileges associated with a given role. If this type of request is to be supported, then a HasPrivilegesOfRole `<Policy>` must be included in each Permission `<PolicySet>`. Support for this type of `<Policy>`, and thus for requests asking whether a subject has the privileges associated with a given role, is optional.

Permission `<PolicySet>` instances must be stored in the policy repository in such a way that they can never be used as the initial policy for an XACML PDP; Permission `<PolicySet>` instances must be reachable only through the corresponding Role `<PolicySet>`. This is because, in order to support hierarchical roles, a Permission `<PolicySet>` must be applicable to every subject. The Permission `<PolicySet>` depends on its corresponding Role `<PolicySet>` to ensure that only subjects holding the corresponding role attribute will gain access to the permissions in the given Permission `<PolicySet>`.

Use of separate Role `<PolicySet>` and Permission `<PolicySet>` instances allows support for Hierarchical RBAC, where a more *senior* role can acquire the permissions of a more *junior* role. A Permission `<PolicySet>` that does not reference other Permission `<PolicySet>` elements could actually be an XACML `<Policy>` rather than a `<PolicySet>`. Requiring it to be a `<PolicySet>`, however, allows its associated role to become part of a role hierarchy at a later time without requiring any change to other policies.

1.6 Multi-Role Permissions

In this profile, it is possible to express policies where a user must hold several roles simultaneously in order to gain access to certain permissions. For example, changing the care instructions for a hospital patient may require that the `Subject` performing the action have both the *physician* role and the *staff* role.

These policies may be expressed using a Role `<PolicySet>` where the `<Target>` element requires the `Subject` to have all necessary role attributes. This is done by using a single `<Subject>` element containing multiple `<SubjectMatch>` elements. The associated Permission `<PolicySet>` should specify the permissions associated with `Subjects` who simultaneously have all the specified roles enabled.

The Permission `<PolicySet>` associated with a multi-role policy may reference the Permission `<PolicySet>` instances associated with other roles, and thus may inherit permissions from other roles. The permissions associated with a given multi-role `<PolicySet>` may also be inherited by another role if the other role includes a reference to the Permission `<PolicySet>` associated with the multi-role policy in its own Permission `<PolicySet>`.

2 Example (non-normative)

This section presents a complete example of the types of policies associated with role based access control.

The example uses XACML 2.0 syntax. For XACML 1.0 and 1.1, the `xmlns` references should be changed to use the 1.0 or 1.1 schema identifiers. A `<Target>` element containing only `<AnySubject/>`, `<AnyResource/>`, and `<AnyAction/>` should be added if there is no `<Target>` element. `<AnySubject/>`, `<AnyResource/> and <AnyAction/>` elements should be added to a `<Target>` element that does not have an instance `<Subjects>`, `<Resources>`, or `<Actions>`, respectively. In `<Condition>` elements, the specified `<Condition>` tags should be removed and the top-level `<Apply>` element with its `FunctionId` should be changed to a `<Condition>` element.

Assume an organization uses two roles, *manager* and *employee*. In this example, they are expressed as two separate values for a single XACML Attribute with AttributeId "`&role;`". The `&role;` Attribute values corresponding to the two roles are "`&roles;employee`" and "`&roles;manager`". An *employee* has permission to create a purchase order. A *manager* has permission to sign a purchase order, plus any permissions associated with the *employee* role. The *manager* role therefore is *senior* to the *employee* role, and the *employee* role is *junior* to the *manager* role.

According to this profile, there will be two Permission `<PolicySet>` instances: one for the *manager* role and one for the *employee* role. The *manager* Permission `<PolicySet>` will give any `Subject` the specific permission to sign a purchase order and will reference the *employee* Permission `<PolicySet>` in order to inherit its permissions. The *employee* Permission `<PolicySet>` will give any `Subject` the permission to create a purchase order.

According to this profile, there will also be two Role `<PolicySet>` instances: one for the *manager* role and one for the *employee* role. The *manager* Role `<PolicySet>` will contain a `<Target>` requiring that the `Subject` hold a `&role;` Attribute with a value of "`&roles;manager`". It will reference the *manager* Permission `<PolicySet>`. The *employee* Role `<PolicySet>` will contain a `<Target>` requiring that the `Subject` hold a `&role;` Attribute with a value of "`&roles;employee`". It will reference the *employee* Permission `<PolicySet>`.

The actual XACML policies implementing this example follow. An example of a Role Assignment Policy is included in Section 3: *Assigning and Enabling Role Attributes*.

2.1 Permission `<PolicySet>` for the *manager* role

The following Permission `<PolicySet>` contains the permissions associated with the *manager* role. The PDP's policy retrieval must be set up such that access to this `<PolicySet>` is gained only by reference from the *manager* Role `<PolicySet>`.

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:cd-01"
    PolicySetId="PPS:manager:role"
    PolicyCombiningAlgId="&policy-combine;permit-overrides">

  <!-- Permissions specifically for the manager role -->
  <Policy PolicyId="Permissions:specifically:for:the:manager:role"
      RuleCombiningAlgId="&rule-combine;permit-overrides">

    <!-- Permission to sign a purchase order -->
    <Rule RuleId="Permission:to:sign:a:purchase:order"
        Effect="Permit">
      <Target>
        <Resources>
          <Resource>
            <ResourceMatch MatchId="&function;string-equal">
              <AttributeValue
DataType="&xml;string">purchase order</AttributeValue>
              <ResourceAttributeDesignator
                  AttributeId="&resource;resource-id"
                  DataType="&xml;string"/>
            </ResourceMatch>
          </Resource>
        </Resources>
        <Actions>
          <Action>
            <ActionMatch MatchId="&function;string-equal">
              <AttributeValue
                  DataType="&xml;string">sign</AttributeValue>
              <ActionAttributeDesignator
                  AttributeId="&action;action-id"
                  DataType="&xml;string"/>
            </ActionMatch>
          </Action>
        </Actions>
      </Target>
    </Rule>
  </Policy>

  <!-- Include permissions associated with employee role -->
  <PolicySetIdReference>PPS:employee:role</PolicySetIdReference>
</PolicySet>
```

Table 1 Permission <PolicySet> for managers

2.2 Permission `<PolicySet>` for *employee* role

The following Permission *`<PolicySet>`* contains the permissions associated with the *employee* role. The PDP's policy retrieval must be set up such that access to this `<PolicySet>` is gained only by reference from the *employee* Role `<PolicySet>` or by reference from the more senior *manager* Role `<PolicySet>` via the *manager* Permission `<PolicySet>`.

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:cd-01"
     PolicySetId="PPS:employee:role"
     PolicyCombiningAlgId="&policy-combine;permit-overrides">

  <!-- Permissions specifically for the employee role -->
  <Policy PolicyId="Permissions:specifically:for:the:employee:role"
      RuleCombiningAlgId="&rule-combine;permit-overrides">

    <!-- Permission to create a purchase order -->
    <Rule RuleId="Permission:to:create:a:purchase:order"
        Effect="Permit">
      <Target>
        <Resources>
          <Resource>
            <ResourceMatch MatchId="&function;string-equal">
              <AttributeValue
DataType="&xml;string">purchase order</AttributeValue>
              <ResourceAttributeDesignator
                  AttributeId="&resource;resource-id"
                  DataType="&xml;string"/>
            </ResourceMatch>
          </Resource>
        </Resources>
        <Actions>
          <Action>
            <ActionMatch MatchId="&function;string-equal">
              <AttributeValue
                  DataType="&xml;string">create</AttributeValue>
              <ActionAttributeDesignator
                  AttributeId="&action;action-id"
                  DataType="&xml;string"/>
            </ActionMatch>
          </Action>
        </Actions>
      </Target>
    </Rule>
  </Policy>
</PolicySet>
```

Table 2 Permission <PolicySet> for employees

2.3 Role `<PolicySet>` for the *manager* role

The following Role `<PolicySet>` is applicable, according to its `<Target>`, only to `Subjects` who hold a `&role;` Attribute with a value of "`&roles;manager`". The `<PolicySetIdReference>` points to the Permission `<PolicySet>` associated with the *manager* role. That Permission `<PolicySet>` may be viewed in Section 2.1: *Permission `<PolicySet>` for the manager role* above.

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:cd-01"
    PolicySetId="RPS:manager:role"
    PolicyCombiningAlgId="&policy-combine;permit-overrides">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="&function;anyURI-equal">
          <AttributeValue
              DataType="&xml;anyURI">&roles;manager</AttributeValue>
          <SubjectAttributeDesignator
              AttributeId="&role;"
              DataType="&xml;anyURI"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
  </Target>

  <!-- Use permissions associated with the manager role -->
  <PolicySetIdReference>PPS:manager:role</PolicySetIdReference>
</PolicySet>
```

Table 3 Role <PolicySet> for managers

2.4 Role `<PolicySet>` for *employee* role

The following Role `<PolicySet>` is applicable, according to its `<Target>`, only to `Subjects` who hold a `&role;` Attribute with a value of "`&roles;employee`". The `<PolicySetIdReference>` points to the Permission `<PolicySet>` associated with the *employee* role. That Permission `<PolicySet>` may be viewed in Section 2.2: *Permission `<PolicySet>` for employee role* above.

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:cd-01"
    PolicySetId="RPS:employee:role"
    PolicyCombiningAlgId="&policy-combine;permit-overrides">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="&function;anyURI-equal">
          <AttributeValue
              DataType="&xml;anyURI">&roles;employee</AttributeValue>
          <SubjectAttributeDesignator
              AttributeId="&role;"
              DataType="&xml;anyURI"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
  </Target>

  <!-- Use permissions associated with the employee role -->
  <PolicySetIdReference>PPS:employee:role</PolicySetIdReference>
</PolicySet>
```

Table 4 Role <PolicySet> for employees

2.5 HasPrivilegesOfRole Policies and Requests

An XACML RBAC system MAY choose to support queries of the form “Does this subject have the privileges of role X?” If so, each Permission `<PolicySet>` MUST contain a HasPrivilegesOfRole *<Policy>.*

For the Permission <PolicySet> for managers, the HasPrivilegesOfRole <Policy> would look as follows:

```
  <!-- HasPrivilegesOfRole Policy for manager role -->
  <Policy PolicyId="Permission:to:have:manager:role:permissions"
      RuleCombiningAlgId="&rule-combine;permit-overrides">

    <!-- Permission to have manager role permissions -->
    <Rule RuleId="Permission:to:have:manager:permissions"
        Effect="Permit">
      <Condition>
       <Apply FunctionId=”&function;and”>
        <Apply FunctionId=”&function;anyURI-is-in”>
          <AttributeValue
DataType=”&xml;anyURI”>&roles;manager</AttributeValue>
          <ResourceAttributeDesignator
              AttributeId="&role;"
              DataType="&xml;anyURI"/>
        </Apply>
        <Apply FunctionId=”&function;anyURI-is-in”>
            <AttributeValue
DataType=”&xml;anyURI”>&actions;hasPrivilegesofRole</AttributeValue>
            <ActionAttributeDesignator
               AttributeId=”&action;action-id”
               DataType=”&xml;anyURI”/>
        </Apply>
       </Apply>
      </Condition>
    </Rule>
  </Policy>
```

Table 5 HasPrivilegesOfRole <Policy> for manager role

For the Permission <PolicySet> for employees, the HasPrivilegesOfRole <Policy> would look as follows:

```
  <!-- HasPrivilegesOfRole Policy for employee role -->
  <Policy PolicyId="Permission:to:have:employee:role:permissions"
      RuleCombiningAlgId="&rule-combine;permit-overrides">

    <!-- Permission to have employee role permissions -->
    <Rule RuleId="Permission:to:have:employee:permissions"
        Effect="Permit">
      <Condition>
       <Apply FunctionId=”&function;and”>
        <Apply FunctionId=”&function;anyURI-is-in”>
          <AttributeValue
DataType=”&xml;anyURI”>&roles;employee</AttributeValue>
          <ResourceAttributeDesignator
              AttributeId="&role;"
              DataType="&xml;anyURI"/>
        </Apply>
        <Apply FunctionId=”&function;anyURI-is-in”>
            <AttributeValue
DataType=”&xml;anyURI”>&actions;hasPrivilegesofRole</AttributeValue>
            <ActionAttributeDesignator
               AttributeId=”&action;action-id”
               DataType=”&xml;anyURI”/>
        </Apply>
       </Apply>
      </Condition>
    </Rule>
  </Policy>
```

Table 6 HasPrivilegesOfRole <Policy> for employee role

A Request asking whether subject *Anne* has the privileges associated with *&roles;manager* would look as follows.

```
<Request>
  <Subject>
    <Attribute AttributeId=”&subject;subject-id”
DataType=”&xml;string”>
      <AttributeValue>Anne</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId=”&role;”
DataType=”&xml;anyURI”>
      <AttributeValue>&roles;manager</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId=”&action;action-id”
DataType="&xml;anyURI">&actions;hasPrivilegesOfRole</AttributeValue>
    </Attribute>
  </Action>
</Request>
```

Table 7 Example of HasPrivilegesOfRole Request

Either the `<Request>` must contain *Anne*'s direct roles (in this case, *&roles;employee*), or else the PDP's Context Handler must be able to discover them. HasPrivilegesOfRole Policies do not do the job of associating roles with subjects. See Section 3: *Assigning and Enabling Role Attributes* for more information on how roles are associated with subjects.

3 Assigning and Enabling Role Attributes (non-normative)

The assignment of various role attributes to users and the enabling of those attributes within a session are outside the scope of the XACML PDP. There must be one or more separate entities, referred to a Role Enablement Authorities, implemented to perform these functions. This profile assumes that the presence in the XACML Request Context of a role attribute for a given user (`Subject`) is a valid assignment at the time the access decision is requested

So where do a subject's role attributes come from? What does one of these Role Enablement Authorities look like? The answer is implementation dependent, but some possibilities can be suggested.

In some cases, role attributes might come from an identity management service that maintains information about a user, including the subject's assigned or allowed roles; the identity management service acts as the Role Enablement Authority. This service might store static role attributes in an LDAP directory, and a PDP's Context Handler might retrieve them from there. Or this service might respond to requests for a subject's role attributes from a PDP's Context Handler, where the requests are in the form of SAML Attribute Queries.

Role Enablement Authorities MAY use an XACML Role Assignment `<Policy>` or `<PolicySet>` to determine whether a subject is allowed to have a particular role attribute and value enabled. A Role Assignment `<Policy>` or `<PolicySet>` answers the question "Is subject X allowed to have role R_i enabled?" It does not answer the question "Which set of roles is subject X allowed to have enabled?" The Role Enablement Authority must have some way of knowing which role or roles to submit a request for. For example, the Role Enablement Authority might maintain a list of all possible roles, and, when asked for the roles associated with a given subject, make a request against the Role Assignment policies for each candidate role.

In this profile, Role Assignment policies are a different set from the Role `<PolicySet>` and Permission `<PolicySet>` instances used to determine the access permissions associated with each role. Role Assignment policies are to be used only when the XACML Request comes from a Role Enablement Authority. This separation may be managed in various ways, such as by using different PDPs with different policy stores or requiring `<Request>` elements for role enablement queries to include a `<Subject>` with a `SubjectCategory` of "`&subject-category;role-enablement-authority`".

There is no fixed form for a Role Assignment `<Policy>`. The following example illustrates one possible form. It contains two XACML `<Rule>` elements. The first `<Rule>` states that `Anne` and `Seth` and `Yassir` are allowed to have the "`&roles;employee`" role enabled between the hours of 9am and 5pm. The second `<Rule>` states that `Steve` is allowed to have the "`&roles;manager`" role enabled, with no restrictions on time of day.

```
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:cd-01"
    PolicyId="Role:Assignment:Policy"
    RuleCombiningAlgId="&rule-combine;permit-overrides">
```

```
<!--  Employee role requirements rule -->
<Rule RuleId="employee:role:requirements" Effect="Permit">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="&function;string-equal">
          <AttributeValue
              DataType="&xml;string">Seth</AttributeValue>
          <SubjectAttributeDesignator
              AttributeId="&subject;subject-id"
```

```
                    DataType="&xml;string"/>
              </SubjectMatch>
            </Subject>
            <Subject>
              <SubjectMatch MatchId="&function;string-equal">
                <AttributeValue
                    DataType="&xml;string">Anne</AttributeValue>
                <SubjectAttributeDesignator
                    AttributeId="&subject;subject-id"
                    DataType="&xml;string"/>
              </SubjectMatch>
            </Subject>
          </Subjects>
          <Resources>
            <Resource>
              <ResourceMatch MatchId="&function;anyURI-equal">
                <AttributeValue
                   DataType="&xml;anyURI">&roles;employee</AttributeValue>
                <ResourceAttributeDesignator
                    AttributeId="&role;"
                    DataType="&xml;anyURI"/>
              </ResourceMatch>
            </Resource>
          </Resources>
          <Actions>
            <Action>
              <ActionMatch MatchId="&function;anyURI-equal">
                <AttributeValue
                    DataType="&xml;anyURI">&actions;enableRole</AttributeVa
    lue>
                <ActionAttributeDesignator
                    AttributeId="&action;action-id"
                    DataType="&xml;anyURI"/>
              </ActionMatch>
            </Action>
          </Actions>
        </Target>
        <Condition>
         <Apply FunctionId="&function;and">
          <Apply FunctionId="&function;time-greater-than-or-equal">
            <Apply FunctionId="&function;time-one-and-only">
              <EnvironmentAttributeDesignator
                  AttributeId="&environment;current-time"
                  DataType="&xml;time"/>
            </Apply>
            <AttributeValue
               DataType="&xml;time">9h</AttributeValue>
          </Apply>
          <Apply FunctionId="&function;time-less-than-or-equal">
            <Apply FunctionId="&function;time-one-and-only">
              <EnvironmentAttributeDesignator
                  AttributeId="&environment;current-time"
                  DataType="&xml;time"/>
            </Apply>
            <AttributeValue
                DataType="&xml;time">17h</AttributeValue>
         </Apply>
```

```
   </Apply>
  </Condition>
</Rule>
```

376

```
<!-- Manager role requirements rule -->
<Rule RuleId="manager:role:requirements" Effect="Permit">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="&function;string-equal">
          <AttributeValue
               DataType="&xml;string">Steve</AttributeValue>
          <SubjectAttributeDesignator
               AttributeId="&subject;subject-id"
               DataType="&xml;string"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="&function;anyURI-equal">
          <AttributeValue
             DataType="&xml;anyURI">&roles;:manager</AttributeValue>
          <ResourceAttributeDesignator
              AttributeId="&role;"
              DataType="&xml;anyURI"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch MatchId="&function;anyURI-equal">
          <AttributeValue
              DataType="&xml;anyURI">&actions;enableRole</AttributeVa
lue>
          <ActionAttributeDesignator
              AttributeId="&action;action-id"
              DataType="&xml;anyURI"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>
</Policy>
```

Table 8 Role Assignment <Policy> Example

4 Implementing the RBAC Model (non-normative)

The following sections describe how to use XACML policies to implement various components of the RBAC model as described in [ANSI-RBAC].

4.1 Core RBAC

Core RBAC, as defined in [ANSI-RBAC], includes the following five basic data elements:

1. **Users**
2. **Roles**
3. **Objects**
4. **Operations**
5. **Permissions**

Users are implemented using XACML `Subjects`. Any of the XACML `SubjectCategory` values may be used, as appropriate.

Roles are expressed using one or more XACML Subject Attributes. The set of roles is very application- and policy domain-specific, and it is very important that different uses of roles not be confused. For these reasons, this profile does not attempt to define any standard set of role values, although this profile does recommend use of a common `AttributeId` value of "`urn:oasis:names:tc:xacml:2.0:subject:role`". It is recommended that each application or policy domain agree on and publish a unique set of `AttributeId` values, `DataType` values, and `<AttributeValue>` values that will be used for the various roles relevant to that domain.

Objects are expressed using XACML `Resources`.

Operations are expressed using XACML `Actions`.

Permissions are expressed using XACML Role `<PolicySet>` and Permission `<PolicySet>` instances as described in previous sections.

Core RBAC requires support for multiple users per role, multiple roles per user, multiple permissions per role, and multiple roles per permission. Each of these requirements can be satisfied by XACML policies based on this profile as follows. Note, however, that the actual assignment of roles to users is outside the scope of the XACML PDP. For more information see Section 3: *Assigning and Enabling Role Attributes*.

XACML allows multiple Subjects to be associated with a given role attribute. XACML Role `<PolicySet>`s defined in terms of possession of a particular role `<Attribute>` and `<AttributeValue>` will apply to any requesting user for which that role `<Attribute>` and `<AttributeValue>` are in the XACML Request Context.

XACML allows multiple role attributes or role attribute values to be associated with a given `Subject`. If a `Subject` has multiple roles enabled, then any Role `<PolicySet>` instance applying to any of those roles may be evaluated, and the permissions in the corresponding Permission `<PolicySet>` will be permitted. As described in Section 1.6: *Multi-Role Permissions*, it is even possible to define policies that require a given `Subject` to have multiple role attributes or values enabled at the same time. In this case, the permissions associated with the multiple-role requirement will apply only to a `Subject` having all the necessary role attributes and values at the time an XACML Request Context is presented to the PDP for evaluation.

The Permission `<PolicySet>` associated with a given role may allow access to multiple resources using multiple actions. XACML has a rich set of constructs for composing permissions, so there are multiple ways in which multi-permission roles may be expressed. Any *Role A* may be associated with a Permission `<PolicySet>` *B* by including a `<PolicySetIdReference>` to Permission `<PolicySet>` *B* in the Permission `<PolicySet>` associated with the *Role A*. In this way, the same set of permissions may be associated with more than one role.

In addition to the basic Core RBAC requirements, XACML policies using this profile can also express arbitrary conditions on the application of particular permissions associated with a role. Such conditions might include limiting the permissions to a given time period during the day, or limiting the permissions to role holders who also possess some other attribute, whether it is a role attribute or not.

4.2 Hierarchical RBAC

Hierarchical RBAC, as defined in [ANSI-RBAC], expands Core RBAC with the ability to define inheritance relations between roles. For example, *Role A* may be defined to inherit all permissions associated with *Role B*. In this case, *Role A* is considered to be *senior* to *Role B* in the role hierarchy. If *Role B* in turn inherits permissions associated with *Role C*, then *Role A* will also inherit those permissions by virtue of being senior to *Role B*.

XACML policies using this profile can implement role inheritance by including a `<PolicySetIdReference>` to the Permission `<PolicySet>` associated with one role inside the Permission `<PolicySet>` associated with another role. The role that includes the `<PolicySetIdReference>` will then inherit the permissions associated with the referenced role.

This profile structures policies in such a way that inheritance properties may be added to a role at any time without requiring changes to `<PolicySet>` instances associated with any other roles. An organization may not initially use role hierarchies, but may later decide to make use of this functionality without having to rewrite existing policies.

5 Profile (normative)

5.1 Roles and Role Attributes

Roles SHALL be expressed using one or more XACML Attributes. Each application domain using this profile for role based access control SHALL define or agree upon one or more AttributeId values to be used for role attributes. Each such AttributeId value SHALL be associated with a set of permitted values and their DataTypes. Each permitted value for such an AttributeId SHALL have well-defined semantics for the use of the corresponding value in policies.

This profile RECOMMENDS use of the "`urn:oasis:names:tc:xacml:2.0:subject:role`" AttributeId value for all role attributes. Instances of this Attribute SHOULD have a DataType of "`http://www.w3.org/2001/XMLSchema#anyURI`".

5.2 Role Assignment or Enablement

A Role Enablement Authority, responsible for assigning roles to users and for enabling roles for use within a user's session, MAY use an XACML Role Assignment `<Policy>` or `<PolicySet>` to determine which users are allowed to enable which roles and under which conditions. There is no prescribed form for a Role Assignment `<Policy>` or `<PolicySet>`. It is RECOMMENDED that roles in a Role Assignment `<Policy>` or `<PolicySet>` be expressed as Resource Attributes, where the `AttributeId` is `&role;` and the `<AttributeValue>` is the URI for the relevant role value. It is RECOMMENDED that the action of assigning or enabling a role be expressed as an Action Attribute, where the `AttributeId` is `&action;action-id`, the `DataType` is `&xml;anyURI`, and the `<AttributeValue>` is `&actions;enableRole`.

5.3 Access Control

Role based access control SHALL be implemented using two types of `<PolicySet>`s: Role `<PolicySet>`, Permission `<PolicySet>`. The specific functions and requirements of these two types of `<PolicySet>`s are as follows.

For each role, one Role `<PolicySet>` SHALL be defined. Such a `<PolicySet>` SHALL contain a `<Target>` element that makes the `<PolicySet>` applicable only to Subjects having the XACML Attribute associated with the given role; the `<Target>` element SHALL NOT restrict the `Resource`, `Action`, or `Environment`. Each Role `<PolicySet>` SHALL contain a single `<PolicySetIdReference>` element that references the unique Permission `<PolicySet>` associated with the role. The Role `<PolicySet>` SHALL NOT contain any other `<Policy>`, `<PolicySet>`, `<PolicyIdReference>`, or `<PolicySetIdReference>` elements.

For each role, one Permission `<PolicySet>` SHALL be defined. Such a `<PolicySet>` SHALL contain `<Policy>` and `<Rule>` elements that specify the types of access permitted to Subjects having the given role. The `<Target>` of the `<PolicySet>` and its included or referenced `<PolicySet>`, `<Policy>`, and `<Rule>` elements SHALL NOT limit the Subjects to which the Permission `<PolicySet>` is applicable.

If a given role inherits permissions from one or more junior roles, then the Permission `<PolicySet>` for the given (senior) role SHALL include a `<PolicySetIdReference>` element for each junior role. Each such `<PolicySetIdReference>` shall reference the Permission `<PolicySet>` associated with the junior role from which the senior role inherits.

A Permission `<PolicySet>` MAY include a HasPrivilegesOfRole `<Policy>`. Such a `<Policy>` SHALL have a `<Rule>` element with an effect of "`Permit`". This Rule SHALL permit any Subject to perform an Action with an Attribute having an `AttributeId` of `&action;action-id`, a `DataType` of `&xml;anyURI`, and an `<AttributeValue>` having a value of `&actions;hasPrivilegesOfRole` on a Resource having an Attribute that is the role to which the Permission `<PolicySet>` applies (for example, an `AttributeId` of `&role;`, a `DataType` of `&xml;anyURI`, and an `<AttributeValue>`

whose value is the URI of the specific role value). Note that the role Attribute, which is a Subject Attribute in a Role `<PolicySet>` `<Target>`, is treated as a Resource Attribute in a HasPrivilegesOfRole `<Policy>`.

The organization of any repository used for policies and the configuration of the PDP SHALL ensure that the PDP can never use a Permission `<PolicySet>` as the PDP's initial policy.

6 Identifiers (normative)

This profile defines the following URN identifiers.

6.1 Profile Identifier

The following identifier SHALL be used as the identifier for this profile when an identifier in the form of a URI is required.

```
urn:oasis:names:tc:xacml:2.0:profiles:rbac:core-hierarchical
```

6.2 Role Attribute

The following identifier MAY be used as the `AttributeId` for role Attributes.

```
urn:oasis:names:tc:xacml:2.0:subject:role
```

6.3 SubjectCategory

The following identifier MAY be used as the `SubjectCategory` for Subject Attributes identifying that a Request is coming from a Role Enablement Authority.

```
urn:oasis:names:tc:xacml:2.0:subject-category:role-enablement-authority
```

6.4 Action Attribute Values

The following identifier MAY be used as the `<AttributeValue>` of the `&action;action-id` Attribute in a HasPrivilegesOfRole `<Policy>`.

```
urn:oasis:names:tc:xacml:2.0:actions:hasPrivilegesOfRole
```

The following identifier MAY be used as the `<AttributeValue>` of the `&action;action-id` Attribute in a Role Assignment `<Policy>`.

```
urn:oasis:names:tc:xacml:2.0:actions:enableRole
```

7 References

7.1 Normative References

[RFC2119]	S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt
[XACML]	S. Godik, T. Moses, eds., *OASIS eXtensible Access Control Markup Language (XACML) Version 2.0*, Committee Draft 01, 16 September 2004, http://docs.oasis-open.org/xacml/access_control-xacml-2.0-core-spec-cd-01.pdf; S. Godik, T. Moses, eds., *OASIS eXtensible Access Control Markup Language (XACML) Version 1.1*, Committee Specification, 7 August 2003, http://www.oasis-open.org/committees/xacml/repository/cs-xacml-specification-1.1.pdf; and S. Godik, T. Moses, eds., *OASIS eXtensible Access Control Markup Language (XACML) Version 1.0,* OASIS Standard, 18 February 2003, http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf.

7.2 Non-normative References

[ANSI-RBAC]	NIST, *Role Based Access Control,* ANSI INCITS 359-2004, http://csrc.nist.gov/rbac/ .
[RBACIntro]	D. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, R. Chandramouli, *Proposed NIST Standard for Role-Based Access Control*, ACM Transaction on Information and System Security, Vol. 4, No. 3, August 2001, pages 224-274, http://csrc.nist.gov/rbac/rbacSTD-ACM.pdf.
[RBAC-V1]	A. Anderson, ed., *XACML Profile for Role Based Access Control (RBAC),* OASIS Access Control TC Committee Draft 01, 13 February 2004, http://docs.oasis-open.org/xacml/cd-xacml-rbac-profile-01.pdf.
[XACMLIntro]	OASIS XACML TC, *A Brief Introduction to XACML*, 14 March 2003, http://www.oasis-open.org/committees/download.php/2713/Brief_Introduction_to_XACML.html,.

A. Acknowledgments

The editor would like to acknowledge the contributions of the OASIS XACML Technical Committee, whose voting members at the time of publication were:

- Frank Siebenlist, Argonne National Laboratory
- Daniel Engovatov, BEA Systems, Inc.
- Hal Lockhart, BEA Systems, Inc.
- Rebekah Metz, Booz Allen Hamilton
- Ronald Jacobson, Computer Associates
- Tim Moses, Entrust
- Simon Godik, GlueCode Software
- Bill Parducci, GlueCode Software
- Michiharu Kudo, IBM
- Michael McIntosh, IBM
- Anthony Nadalin, IBM
- Steve Anderson, OpenNetwork
- Anne Anderson, Sun Microsystems
- Seth Proctor, Sun Microsystems
- Polar Humenn, Syracuse University
- Edward Coyne, Veterans Health Administration

In addition, the following people made contributions to this specification:

- Ravi Sandhu, George Mason Univ.
- John Barkley, NIST
- Ramaswamy Chandramouli, NIST
- David Ferraiolo, NIST
- Rick Kuhn, NIST
- Serban Gavrila, VDG Inc.

B. Revision History

Rev	Date	By Whom	What
CD-01	30 Sept 2004	XACML committee	Committee Draft

C. Notices