Privilege and Access Management Recipes--A Discussion-starter Draft

Privileges, a generic model

This is an older version of the Recipe.

Please see the new version.

A **Group** is a collection of subjects. An example of using a group without using authorization is an email list. A **Role** is a collection of privileges that is shared by all subjects assigned to the role, which generally describes the subjects' affiliation, job function, or responsibility. A **Resource** is the part of the system which needs to be protected by authorization, and it represents a noun in a privilege assignment. The **Action** is the verb of the privilege assignment which allows a resource to be assigned to a subject in various ways without creating more resources . For example SubjectA can view (action) the Math department data (resource). A **Privilege Assignment** associates the subject with the actions and resources that they are allowed to perform. A **Limit** is a condition on the privilege assignment which must be true at run-time for the privilege assignment to be allow. Examples of limits are time of day, source IP address, amounts of approvals, etc.

Many of the parts of the privilege model can have inheritance from other objects. The privilege management system might require the inheritance to be a hierarchy, or maybe it is required to be acyclic, or maybe it could be a directed graph.

If the system allows a group to be added as a member of another group, then membership in a group implies membership to other groups. Role inheritance means that privileges assigned to a role are inherited to other roles. For instance, to reuse the definition of a role, a role SeniorLoanAdministrator could be allowed to do everything a LoanAdministrator can do, plus a few more privilege assignments. Note that if there is role inheritance, it does not mean that a subject assigned to SeniorLoanAdministrator is also assigned to role LoanAdministrator. Inheritance on actions means that assigning an action to a subject for a resource also implies that subject has other actions on that resource. For instance, subjects with Action<Write> on a resource might also implicitly be able to Action<Read> the resource. In other cases, the read action might need to be assigned separately from write.

Resource inheritance means that subjects which are assigned an action on a resource have other implied resources for that action (and inherited actions). An institution's organization chart or course structure could be modeled as privilege resources. Instructors could be granted privileges on individual courses, and deans could be granted privilege actions on entire departments or schools. Sometime a resource which implies other resources is referred to as a role, however, it might be that the bundle of resources qualifies a role instead of defines a new role.

Groups, roles, privileges, and external authorization

Applications have various access control needs to control which subjects have access to what. This is generally for applications which require authentication, though the authorization could also apply to anonymous subjects (who would get privileges from an anonymous role). There are several decisions for access control for applications including: groups vs roles, roles vs privileges vs hybrid, hard-coded privileges vs external privileges vs centralized privileges.

When applications protect resources by checking if the authenticated user is in a group, they are essentially using a group as if it were a role. For example, if the application code checks if the authenticated user is in the institution's "student" group, in order for them to see the main screen of the application, then there is an implicit hard-coded privilege resource of "main-screen", and action "view", assigned to the role "studentUser", which is assigned to the group "student". Though it is referred to as security by group, it is actually a role.

Generally an application will have a handful or up to a couple of dozen roles. These roles are generally mapped to job function or affiliation. For example there could be roles for student, staff, instructor, admin, billingAdmin, anonymousUser, etc. Note that set of roles varies across applications as needed. Roles are used as a security template that multiple subjects can be assigned to for common access.

If not all subjects in a role have the exact same access, then they need personalized privileges assigned to them. The authorization system might assign privileges directly to the subject, or in the context of an application, or in the context of a role. Either way, the subject will have privileges that are different from other subjects assigned to the same role. An example is to specify which data-sets the subject has access to when searching for data or running reports. If the authorization system supports applications with subjects which select one role at a time to use the application, then individual privileges should be able to be assigned in context of one role. The subject will not need to have elevated privileges at all times, and thus will be less likely to accidentally perform tasks that only the elevated role can perform.

If there is no access control framework or central system, then hard-coding the privileges in the application might be the easiest way to code an application. An external privilege management system separated the privilege resources from the code, but still runs in the same application for that application. A central authorization system maintains the roles and privileges in a central location as middleware for multiple applications. External or centralized authorization gives the application run-time flexibility for changes in access control policy and authorization reporting. Centralized authorization systems can show subject's privileges across applications and can ease revocation. Certainly as auditing requirements increase, and sharing of access control policies across multiple applications, centralized authorization is required. If there is a common on-boarding workflow application, then centralized authorization can make the architecture more homogeneous.

When using externalized privileges, there are caching considerations. Caching can improve the performance of the application and reduce the dependencies of middleware components. If there are not real-time updates from the authorization system, then the privileges can become stale. When there are a lot of privilege resources and assignments which need to be checked, it is a good idea to cache the application's privileges for the entire user population, or for one user when the authenticate. If there are reports or queries which need to join available data with the allowed record types, the authorization information might need to be cached directly in the application's database. If there are limits on the authorizations, then it is more complicated then just a list of allowed action/resource pairs.

-
  - Follow the attributes
  - Are there really use cases when access requires realtime evaluation or is static evaluation or cached evaluations enough?
    - Use cases using "presence detection" , is this person physically in a building or a room
    - Use cases with quota evaluation, metering or rate limiting a for fee service?

  - case studies
- attribute delivery recipe:
  - SAML between IdP & SP
  - SPML, XMPP (grouper) for push provisioning
    - XMPP is a messaging protocol that many institutions already run with known security and addressing standards
      - JMS or activeMQ ? AMQP as a wire protocol.
  - LDAP , privilege registry or webservice   for pull provisioning
    - is there existing mace-dir work to build on?
- generalizing to federated scenarios and VOs
  - What is the namespace (URIs vs URNs) and object characteristics for privileges
    - what are the special problems in namespace choice?
      - The name/meaning of a privilege may reflect the policy intention of an organization or a virtual organization and may be applied to subset of either.
      - Some architectures take the "CRUD" view where everything is mapped to create, read, update, delete on specific objects
      - Some architectures use privilege names scoped to the resources they reference.
      - examples
        - urn:mace:edu:cmu:andrew:s3:field:ssn=read
        - urn:mace:edu:cmu:andrew:s3:canreadssns
        - http://www.cmu.edu/andrew/s3/fields/ssn/read
      - The actual privilege may be represented differently by the policy enforcement points in an application or service or may be represented in multiple application for services
      -
    - Fifer using URIs e.g. group://prodGrouper.upenn.edu/penn/apps/directory/users would specify a group so would a privlege be
      - privilege://accessmanager.andrew.cmu.edu/apps/s3/fields/ssn/read be the equivalent?
- authorization and access control
  **case studies in production
- rule-based access control
  - Authorization Techniques and Strategies
    **XACML , DROOLS, others

Access control policy management

Using Paccman terminology, a generic access policy statement P reads

```
P <==> Subject S in Role Ro can perform Action A on Resource Rs constrained by Limits L
```

Note:

- Subjects can be persons or groups
- Roles, Actions and Resources can inherit <<privileges, (policy statements)?>> from other Roles, Actions and Resources
- Limits can be expressed as a sequence of atomic predicates "X Verb Y" joined by logical operators, AND, OR, NOT, XOR.
- X Verb Y is an atomic predicate that is either True or False based on
  - The attributes of X
  - Or some environmental/contextual variable,
  - and proposition Y
- The whole Limit expression evaluates to True or False
- A Limit expression that is True results in an Allow decision for the containing policy statement
- A Limit expression that is False results in a Deny decision for the containing policy statement.

Abstract Definitions of Policy Elements:

- At the most abstract level, Access Policy Management, APM, is the creation, modification or deletion of Policy Statements PS from the set of all policy statements being managed.
- A Policy Decision, PD, consists of evaluating the applicable Policy Statement(s) at the time that subject Su attempts to perform Action A on Resource Rs with the Decision being either Allow or Deny
- Policy Enforcement, PE, either allows Su to perform the requested action A on Resource Rs or not based on whether the Policy Decision is Allow or Deny.

Test of above definitions:

- Translate the policies implicit in the Paccman Primary Use Case Library (PPUCL) into the terminology of the proposed model
- Assess whether there are elements of those policies that are not expressible in terms of the proposed model
- Assess whether there are elements of the proposed model that do not figure in any of the policies implicit in the PPUCL

-
  - P*P architectures: proposed models,
    - Application policy, enterprise policy, VO policy
  - case studies - bamboo

- Examples of access management in production
  - performance issues and design tradeoffs
  - Experience with commercial and open source offerings
  - The Aegis Identity Management Suite based on open standards tools

- Delegation