OSCARS Access Policy

Draft May 30, 2007

Introduction

This document describes the current design for access control in OSCARS. It was intended as an guide to the implementation and is missing much in the way of motivation or justification. The main points of the design are: We have tried to keep it as simple as possible with the intent that more features can be added when they are needed. Authorization is based on attributes which can include the identity of the user, or membership in a group, or a role. A user gets the maximum of all the privileges granted to all of the attributes that he has. We are currently intending to authorize a reservation that has been forwarded to ESnet from an adjacent domain based on the attributes assigned to that domain rather than the end user who is requesting the reservation. The name of the user is included in the forwarded request, so it can be used for auditing purposes or can later be used for authorization. All access decisions are made by one of two methods which are called from each place in the code that is about to grant some access. Some underlying assumptions are: Most users will get their permissions by being members of a group/role, rather than by individual grants. The number of group/roles will be small. The number of resources and permissions will be small. While the implementation may scale to larger numbers, the ability of the users and administrators to understand it will not. There will be web page interfaces to see and set user permissions.

Access policy

The following paragraphs list the various actions that we want to control with respect to the resources that OSCARS controls.

User access policy

- see own profile
- modify own profile
- list all users
- see all user profiles
- add/delete a user
- modify any user profile
- grant or remove authorizations for any user

Reservation access policy

- make unlimited reservation
- make reservation with bandwidth constraint
- make reservation with duration constraint
- make reservation with topology constraint
- list own reservations
- query own reservations
- cancel, modify own reservation
- list all reservations
- query any reservations
- cancel, modify any reservations

Topology access policy (not implemented yet)

- view abstract topology
- view real topology
- modify the topology

Policy Implementation

We have a policy based implementation where the access policy is stored in a set of tables, and an access decision is made by doing a table lookup based on the requestor, the action requested and the resource. Authorization may have constraints attached, in which case the parameters of the request must meet the constraints.

The following tables define the components for authorization:

- **users** – contains an entry for each registered user, containing among other things, user loginId, certSubject and certIssuer
- **resources** – has three entries: users, reservations, topology where topology is the information about specific paths and routers.
- **permissions** – has four entries: list, query, create and modify
    - **list** – shows minimum information about a reservation or user
    - **query** – shows all information about a reservation or user
    - **create** – allows a new user to added or a reservation to be created
    - **modify** – allows deletion of a user and modification to a user profile or cancellation or modification of a reservation.
- **attributes** – has entries for attributes on which authorization is granted, e.g. ESnet-engineer, ESnet-developer, ESnet-admin, ESnet-user and explicit user attributes for users with specific permissions, e.g. user-david. There is an attribute-type field as well which is currently not used.
- **userAttributes** - maps userIds to the attributes they possess
- **constraints** - The current constraints that are supported are:
    - **max-bandwidth** (int), applies to create and modify reservation
    - **max-duration** (int), applies to create and modify reservation
    - **specify-path-elements** (boolean), applies to create and modify reservation
    - **all-users** (boolean), applies to everything except create reservation.
- **authorization -** combines these elements into a row for each action that each user is permitted. Each row consists of an attribute id, a resource id, a permission id and optionally a constraint that applies to this authorization.

In order to base authorizations on attributes other than the user identity, we need a way to assign attributes to users. Users may have 1 or more attributes. Attributes are arbitrary strings, defined in a table, e.g. ESnet-Engineer, Atlas-PI, CMS-member.

The advantage of attributes is that it allows a set of authorizations to be crafted which can then be applied to a class of users. Otherwise, each time a new user registers, he must be given the correct

and complete set of permissions for whatever he wants to do. In our example above, if a new network engineer , Ned, arrives, we would need to duplicate all the authorization entries that ed has for him. If we had just entered those authorizations for "ESnet-Engineer" then we would just need to assign Ned that attribute. It also makes the tables smaller and thus may make it easier to understand the global picture of who has what permissions.

With attribute-based authorization, the PDP will need to look up user attributes, then look up authorizations for each attribute the user has and merge them (take the maximum of the permissions).

Note: <ed>,<david>,<chin>, <andy> are indices in the users table.
<ESnet-eng><ESnet-dev>,<user-chin>,<user-david> are indices in the attributes table.
A user would only need an "id-entry" in the attribute table if some authorization was to be granted explicitly to him. It is probable that a user will get all his authorizations based on group attributes.

Default behaviors

- If a user has no entry in the user table, any attempted OSCARS access will be rejected immediately by Oscars:OscarsSkeleton:checkUser or Servlets:AuthenticateUser
- If there are no attributes for the user in the userAttribute table access will be rejected immediately by Oscars:OscarsSkeleton:checkUser or Servlets:AuthenticateUser
- If no attribute that the user possesses has an entry for the resource/permission pair in the authorization table, the request is denied.
- The constraint defaults are intended to be the most common cases, not the most restricted.
    - If there is no max bandwidth constraint specified, there is no limit on the bandwidth that may be requested.
    - If there is no duration constraint, unlimited duration and persistent reservations are allowed.
    - If no all-users constraint is specified, the default is false (only access to own information is allowed).
    - If specify-path-elements is not specified, the default is false and no routers or hops may be input.

Example

We have users alice, bob, ed, david chin and andy. Alice and bob have all their permissions individually assigned. Ed, chin and andy get all their permissions from group attributes.David get most permissions from groups but has one more added individually.

attributes Table

id (int)	attribute name (string)	attribute type (int)
1	ESnet-engineer	group
2	ESnet-developer	group
3	ESnet-user	group
4	ESnet-administrator	group
5	HOPI-developer	group

6	user-bob	user
7	user-alice	user
8	user-david	user

userAttributes table

id (int)	loginId (int)	attribute id (int)
1	<ed>	<ESnet-eng>
11	<david>	<user-david>
8	<david>	<ESnet-admin>
9	<david>	<ESnet-dev>
3	<chin>	<ESnet-eng>
4	<chin>	<ESnet-user>
5	<chin>	<ESnet-dev>
6	<andy>	<HOPI-dev>
7	<andy>	<ESnet-user>
13	<alice>	<user-alice>
14	<bob>	<user-bob>

Authorization Table

id (int)	attributeId (int)	resource	permission	constraint name (string)	value (int) 1=true, 0=false
	<user-alice>	users	list	null [1]	
	<user-alice>	users	query	null [1]	
	<user-alice>	users	modify	null [1]	
	<user-alice>	reservations	create	max-bandwidth	10M
	<user-alice>	reservations	create	max-duration	600min
	<user-alice>	reservations	list	all-users	0
	<user-alice>	reservations	query	all-users	0
	<user-alice>	reservations	modify	all-users	0
	<user-alice>	reservations	modify	max-bandwidth	10M
	<user-alice>	reservations	modify	max-duration	600min
	<user-bob>	users	list	all-users	1
	<user-bob>	users	query	all-users	1
	<user-bob>	users	modify	all-users	1
	<user-bob>	reservations	create	specify-path-elements	0
	<user-bob>	reservations	list	all-users	1
	<user-bob>	reservations	query	all-users	1
	<user-bob>	reservations	modify	all-users	1
	<user-bob>	reservations	modify	specify-path-elements	0
	<ESnet-eng>	users	list	null [2]	null
	<ESnet-eng>	users	query	null [2]	null
	<ESnet-eng>	users	modify	null [2]	null
	<ESnet-eng>	reservations	list	all-users	1
	<ESnet-eng>	reservations	query	all-users	1
	<ESnet-eng>	reservations	create	specify-path-elements	1

	<ESnet-eng>	reservations	modify	all-users	0
	<ESnet-dev>	users	view	all-users	1
	<ESnet-dev>	users	query	all-users	1
	<ESnet-dev>	users	modify	all-users	0
	<ESnet-dev>	reservations	list	all-users	1
	<ESnet-dev>	reservations	query	all-users	1
	<ESnet-dev>	reservations	create	max-bandwidth	10M
	<ESnet-dev>	reservations	create	max-duration	10min
	<ESnet-dev>	reservations	modify	all-users	0
	<ESnet-dev>	reservations	modify	max-bandwidth	10M
	<ESnet-dev>	reservations	mofify	max-duration	10min
	<ESnet-user>	users	view	null [2]	
	<ESnet-user>	users	query	null [2]	
	<ESnet-user>	users	modify	null [2]	
	<ESnet-user>	reservations	list	null [3]	
	<ESnet-user>	reservations	query	null [3]	
	<ESnet-user>	reservations	create	null [3]	
	<ESnet-user>	reservations	modify	null [3]	
	<ESnet-admin>	users	list	all-users	1
	<ESnet-admin>	users	query	all-users	1
	<ESnet-admin>	users	modify	all-users	1
	<ESnet-admin>	users	create		
	<ESnet-admin>	reservations	view	all-users	1
	<ESnet-admin>	reservations	query	all-users	1
	<ESnet-admin>	reservations	modify	all-users	1
	<user-david>	reservations	create	specify-path-elements	1

Defaults
[1] same as all-users=0, selfOnly
[2] manage and view only his own user info
[3] unlimited bandwidth and duration, only modify, list and query own reservations, not input path components.

Alice is the least privileged. She is allowed to make reservations of a limited bandwidth and duration, and only see and modify her own information. Bob is the leader of a group making reservations. He is allowed to make reservations of unlimited bandwidth, unlimited duration, specify ingress, egress and hops and see and manage other users and their reservations. Ed is a network engineer. As an ESnet-engineer he can make reservations for the purpose of testing, can see but not modify other's reservations, can only see user information, and has free rein over the topology. An ESnet developer can make reservations of limited bandwidth and durations, view everybody's reservations but only modify his own, and can see and modify all user information. An ESnet user can make reservations of unlimited bandwidth and duration, but only view his own reservations and user information. An ESnet admin can manage all user information, see all reservations, but not make or modify reservations. <user-david> has all the permissions of an ESnet developer and ESnet admin and can also specify path elements when creating a reservation.

CheckAccess Implementation

The preferred way to implement access control is to isolate the access checking in one place, called the PolicyDecisionPoint that returns a permit or deny response and is called by all access control points (aka Policy Enforcement Points/ PEP) to see if an action is authorized. The aaa/UserManager classs currently provides a single VerifyAuthorizaton method that meets this goal:

```
Boolean UserManager.verifyAuthorized(String user,String resource String
permission)
```

This needs to be expanded on in order to centralize the evaluations of constraints as much as possible. We are suggesting two methods. The first would be called by everything except createReservation and the second by CreateReservation.

`AuthValue UserManager.checkAccess(String user, String resource, String permission);`

Lookup user in userAttribute table to get list of all attributes
For each attribute lookup ((attrId,,resource,permission) in authorization table.
If deny is returned, continue
If there is no constraint, set returnValue to SELFONLY and continue
If all-users=true is returned, return "ALLUSERS"
If all-users=false is returned, set returnValue to SELFONLY and continue
At end of loop return SELFONLY or DENIED

`AuthValue UserManager.checkModResAccess(String user,String resource, String permission, int ReqBandWidth, int ReqDuration, Boolean specify-path-elements);`

Start out with
ReturnValue = DENIED
bandwidthOK = false
maxdurationOK = false
specPathElemOK = false.
Lookup user in userAttributes table and get a list of attributes for this user.
For each attribute lookup and Authrorization(AttrId,resource,permission)
for the authorization returned
if the constraint is null
mark bandwidthOK=true and durationOK=true,
and returnValue=SELFONLY
if the constraint is max-bandwidth
mark bandwidthOK= true if reqBandwidth < constraintValue
if constraint is max-duration
mark durationOK if reqDuration < constraintValue
if constraint is spec-path-element
mark specPathElemOK = true
if constraint is all-users
if value = 1 set returnValue = ALLUSERS
if value = 0 set returnValue = SELFONLY
At the end of the attribute list

```
   if bandwidthOK = false or maxdurationOK = false
            return DENIED
if specify-path-element= true and specPathElemsOK = false
            return DENIED
otherwise return ReturnValue
```