Grouper permission limits

This topic is discussed in the "Grouper Permissions" training video and in the "LITE UI Permissions Part 3" training video.

Overview of Grouper Permission Limits

Grouper permission limits are used to set up runtime constraints on permissions. You can limit a permission to a condition in the environment or data passed to Grouper. Note, limits can only apply to permissions which are allowed, not disallowed.

A limit is an attribute of type limit assigned on the permission assignment, or assigned to the role membership, or assigned to the role.

At runtime, Grouper, or custom logic, or the caller of the API or WS could set environment variables for the request that the limit logic can use.

You may want to review the permission limit built-in implementations to see if any of them apply to your use case.

See also the Access Management Features Overview page for guidelines of when to use rules, roles, permission limits, and enabled / disabled dates.

Example

For instance, a built-in limit expression language could be (note, you set the Root folder where Grouper creates built in things, in this case it is school:etc):

school:etc:limits:expressionLanguage

When it is assigned, the value could be:

```
amount <= 50000
```

Those limits would take the environment variables available to the limit, and put them as EL variables.

There could be helper variables or classes to do common things (in this case no arguments are required, though the caller could override)...

```
hourOfDay >= 9 && hourOfDay <= 17
```

Or here the caller passes the user's ip address (ipv4 still :))

```
limitElUtils.ipOnNetworks(ipAddress, '1.2.3.4/24, 2.3.4.5/26')
```

In the grouper.properties you could configure other helper classes that are in scope in the expression language

In the grouper.properties you could associate custom limits with subclass of a limit base clase (or an implementation of a Java interface) to take the environment variables, the value of the limit attribute, and other data about the calculation, and give an answer.

The UI for limits has type checking for values, and validates the inputs.

If there are errors, the exception will be thrown so the caller can see what is going on without looking in the server logs.

The UI can set environment variables to simulate a real query and see the red/green results

Limit logic

For each limit, you need to associate a logic class. You can do this with Java. The Grouper administrator needs to register the implementation, though in the future we could do this with JSON/EL (Expression Language) where the admin does not need to help.

Generally you should extend the edu.internet2.middleware.grouper.permissions.limits.PermissionLimitBase class. If you really want to implement an interface, you can implement edu.internet2.middleware.grouper.permissions.limits.PermissionLimitInterface, though if more methods are added with default implementations, you will have to change your code when you upgrade Grouper. Here are the methods:

```
/**
   * if the limit allowed the permission to be allowed
   * @param permissionEntry to check
   * @param limitAssignment the assignment of the limit (e.g. to the permission
   * assignment a parent assignment, or the role, etc)
   * @param limitAssignmentValues
   * @param limitEnvVars value could be String, Long, or Double
   * @return true if allowed, false if not
   */
  public abstract boolean allowPermission(PermissionEntry permissionEntry, AttributeAssign limitAssignment,
      Set<AttributeAssignValue> limitAssignmentValues, Map<String, Object> limitEnvVars);

  /**
   * validate a user entered value(s) on the limit assignment
   * @param limitAssign
   * @return the UI key for the error code (arbitrary, in Grouper should put in nav.properties)
   * or null for ok
   */
  public abstract String validateLimitAssignValue(AttributeAssign limitAssign);

  /**
   * return a UI key to documentation about the limit.  for Grouper, put in nav.properties
   * @return a UI key
   */
  public abstract String documentationKey();

  /**
   * if we can cache the result for a some minutes.
   * i.e. for the same attribute assignment and value and input map, is the result the same...
   * e.g. ip address math can be cached, amount limits, etc.  If there are conditions about the
   * permission names, then dont cache
   * @return the number of minutes to cache
   */
  public abstract int cacheLimitValueResultforMinutes();
```

Register the limit attribute definition name with the logic class in the grouper.properties:

```
# permission limits linked to subclasses of edu.internet2.middleware.grouper.permissions.limits.
PermissionLimitBase
grouper.permissions.limits.logic.someName.limitName =
grouper.permissions.limits.logic.someName.logicClass =
```

Security

If the attribute definitions of the limits are not set correctly (e.g. so the same subjects who can READ the permissions can READ the limits), then if limit security worked like Grouper security, then the limits would not be seen and a wider set of ALLOWs would potentially occur than should. So... if you are reading permissions, and processing limits, or reading limits, you will see them if you can see the permissions. Note, the limit execution context is not as GrouperSystem, so if there is something in there that doesnt work, it should throw an exception or return false.

Caching

The Logic implementation can cache the result per the limit, limit values, and env variables. Note that if there are other data used in the logic (e.g. based on the subject the permission is for), that you shouldnt cache with this strategy, you should use your own caching or dont cache. Also, note if you want a configurable cache, just return a number from GrouperConfig.getPropertyInt(). Normally this is just a boolean, cache for a few minutes, or dont cache at all.

Limit assignment types

The obvious place to assign limits would be on permissions assignments. You can do this in Grouper, but you can also assign limits to other objects to make more general limits. For instance, here is a limit assigned to a permission assignment

```
this.adminRole.getPermissionRoleDelegate().assignSubjectRolePermission(
        this.readString, this.artsAndSciences, this.subj0, PermissionAllowed.ALLOWED);

    AttributeAssign attributeAssign = new PermissionFinder().addSubject(this.subj0).addAction(this.readString)
      .addPermissionName(this.artsAndSciences).addRole(this.adminRole).assignImmediateOnly(true).findPermission
(true).getAttributeAssign();

    attributeAssign.getAttributeValueDelegate().assignValue(PermissionLimitUtils.limitElAttributeDefName().
getName(),
      "hourOfDay >= 9 && hourOfDay <= 17");
```

If you want to assign a limit to a role, then all permissions assignments to that role, or permissions assignments to individuals in the context of that role will inherit that limit. This way you do not have to assign the limit to all permissions assignments for that role. Note, there is no way to change this limit further down the inheritance chain, all subjects in this role will have this limit. However, if you have custom logic, and you want to have an algorithm to do this, it is possible.

```
this.adminRole.getAttributeValueDelegate().assignValue(PermissionLimitUtils.limitElAttributeDefName().getName(),
"amount < 50000");
```

If you want to assign a limit to all the permissions for a user in a role, you can assign it to the AnyMembership (immediate or effective) of a subject and role.

```
GroupMember groupMember = new GroupMember(this.adminRole, this.subj0);

groupMember.getAttributeValueDelegate().assignValue(PermissionLimitUtils.limitIpOnNetworksName(), "1.2.3.0/24,
2.3.4.0/16");
```

GSH example

```
grouperSession = GrouperSession.startRootSession();

AttributeAssign attributeAssign = new PermissionFinder().addAction("Create").addPermissionName("ucla:
permissions:CV").addRole("ucla:roles:english_dept_admin").assignPermissionType(PermissionEntry.PermissionType.
role).findPermission(true).getAttributeAssign();

attributeAssign.getAttributeValueDelegate().assignValue("ucla:limits:group_id", "ucla:hierarchy:faculty");
```