Grouper Web Services Authentication

For web services and UI authentication for Grouper v2.5 and above see this page

High level Grouper Web Services authentication

From a high level, grouper or the servlet container or web server or servlet filter will authenticate the user, and then the user needs to be resolved into a Subject.

Note the default and most popular authentication protocol in grouper-ws is http-basic (Authorization header), so for this and other reasons make sure your deployments of grouper-ws are protected with SSL. In the v2.5+ container there is a param to enable self-signed SSL certs for quick starts if you dont have a real certificate yet.

Types of authentication

1. Self-service JWT
2. JWT from trusted authority
3. Grouper built-in basic authentication
4. Grouper LDAP authentication
5. Tomcat authentication
6. Apache authentication
7. Kerberos user/pass
8. Custom authentication
9. Rampart (on top of something else)
10. SSL certificates. There is no documentation on this. Do this with Apache or Tomcat and pass in the REMOTE_USER.
11. other? if you can get a REMOTE_USER via apache or tomcat plugin or filter, it will work with Grouper WS

Types of subject sources

When the authentication happens and the principal name is given to Grouper from the authentication source, it needs to be resolvable as a subject. There are a few options

1. Use your existing subject source (e.g. ldap, if it can it hold accounts that represent systems)
2. Grouper Local Entities
3. Custom SQL table and source

Configure the default source in the grouper-ws.properties especially if you have subjectId overlap in various sources, but also to help with performance

```
# If there is an entry here for group name, then all web service client users must be in this group (before the
actAs)
#ws.client.user.group.name = etc:webServiceClientUsers

# allow these ids even if not in group, e.g. for testing
# subjectIdOrIdentifier  or  sourceId:::subjectId  or  ::::subjectId  or  sourceId::::::subjectIdentifier  or
::::::subjectIdentifier
# sourceId::::::::subjectIdOrIdentifier  or  ::::::::subjectIdOrIdentifier
# {valueType: "subject", multiple: true}
ws.client.user.group.subjects.allow =

# cache the decision to allow a user to user web services, so it doesnt have to be calculated each time
# defaults to 5 minutes:
# {valueType: "integer", required: true}
ws.client.user.group.cache.minutes = 5

# if you have subject namespace overlap (or not), set the default subject
# sources (comma-separated) to lookup the user if none specified in user name
# {valueType: "string"}
ws.logged.in.subject.default.source =

# prepend to the userid this value (e.g. if using local entities, might be:    etc:servicePrincipals:   )
# {valueType: "string"}
ws.security.prependToUserIdForSubjectLookup =
```

ActAs configuration

To enable web service users to act as another user (proxy), enable the setting in the grouper-ws grouper.properties

```
# Web service users who are in the following group can use the actAs field to act as someone else
ws.act.as.group = aStem:aGroup
```

If you specify a group name in there, you can pass in the actAs field if you connect to the web service as a user who is in the ws.act.as.group group. Here is an example with the axis generated client.

```
//set the act as id
        WsSubjectLookup actAsSubject = WsSubjectLookup.class.newInstance();
        actAsSubject.setSubjectId("GrouperSystem");
        addMember.setActAsSubjectLookup(actAsSubject);
```

There are advanced settings, you can specify multiple groups in the grouper-ws.properties, and you can even limit who the users can act as (in a specific group).