Claim source examples

This blurb has several examples for creating claim sources. Since these are apt to be wordy. it is better to have them organized in a specific place.

Examples.

Adding to the system claims

If you have a configuration named **`my_source.`** then you can simply add it to the default claims for the system:

```
   claim_sources. := claim_sources. ~ [create_source(cfg.)];
```

(Assuming you did not already invoke **`create_source.`**)

Using a script: NCSA claims

If there are existing QDL scripts use them. Here is a common example for the NCSA. It assumes that the user has come in through the NCSA IDP (it will simply do nothing for any other IDP) and gives the option of returning group claims as a JSON structure or a flat list. Generally you want to use this unless there is something very special you need. This is the block you would paste into your client's cfg attribute:

```
tokens{
 identity{
 type=identity
 qdl{
 load="ncsa/ncsa-default.qdl"
 xmd={exec_phase=["pre_auth","post_token", "post_exchange","post_refresh"]}
 args=["true"] // true if the member of claim is just a list. Default is false.
 }// end qdl
 } //end identity token
} //end tokens
```

NCSA claims directly

This is a specific type of LDAP handler that only talks to the internal NCSA LDAP.

```
cfg. := new_template('ncsa');
// any overrides you may need.
cfg. := create_source(cfg.);
```

You may then invoke this directly or add it to the system **`claim_sources.`** to be run for you. This is far more work than using the default script, but it is good to know this is just a very specific LDAP configuration that can be tweaked.

Generic LDAP claims

You may also create custom claims for an LDAP server. Here is a complete example for LIGO.

```
script_load('utils/init.qdl');
// return if the IDP is not right.
if[claims.idp != idp.ligo][return();];

// The ldap.search_name below is 'uid', so it must be set as a claim
           claims.uid := head(claims.eppn, '@');
                 cfg. := new_template('ldap');
        cfg.auth_type := 'none';
          cfg.address := 'ldap.ligo.org';
             cfg.port := 636;
       cfg.claim_name := 'uid';
      cfg.search_base := 'ou=people,dc=ligo,dc=org';
        cfg.ldap_name := 'uid';
          cfg.groups. := ['isMemberOf'];
cfg.search_attributes. := ['email', 'uin', 'uid', 'isMemberOf'];
          cfg.rename. := {'isMemberOf':'is_member_of'};

        claim_sources. := claim_sources. ~ [create_source(cfg.)];
```

This will now be run with the standard claims for the server. The utility script at **`utils/init.qdl`** loaded at the beginning has, among other things, a list of common IDPs. If you have a login through the LIGO IDP, you can just create the **`cfg.`** and then issue a **`get_claims`** if you prefer. Again, adding it to the system claims means it always gets invoked in every phase, so you only have to set it in the id token handler. Note that this example will also rename the isMemberOf claim to is_member_of. To use renaming, you must explicitly list all of the search_attributes you want (normally omitting search_attributes should return everything for the entry).

Another LDAP example: Getting claims directly from NCSA LDAP

Just as an example, here is how to construct an LDAP query to NCSA. You can cut and paste this, but it does require either being onsite at the NCSA or a VPN connection thereto

```
cfg2. := new_template('ldap');
cfg2.address := 'ldap1.ncsa.illinois.edu,ldap2.ncsa.illinois.edu';
cfg2.auth_type := 'none';
cfg2.search_base := 'ou=People,dc=ncsa,dc=illinois,dc=edu';

// So the next two lines are key:

// This is the name of the attribute in LDAP to search on
cfg2.ldap_name := 'uid';

// There is a claim with this name, pass the value to the search engine
cfg2.claim_name := 'uid';

// The search in LDAP then is ldap_name == claim_name
// The next commands specify what attributes to get back from
// the server (rather than all of them)
cfg2.search_attributes. := ['email','uid','uin','memberOf'] ;
```

```
// Since memberOf is a group, we want it parsed rather than being a long messy
string
cfg2.groups. := ['memberOf'];

// Finally, the name in LDAP is 'memberOf' and the expected claim name is
'isMemberOf'
// So we set the rename of
cfg2.rename.memberOf := 'isMemberOf';

claims. := get_claims(create_source(cfg2.), 'YOUR_USER_NAME');
claims.
```

This gets the record for **`YOUR_USER_NAME`** so do supply your own.

HTTP Header claims

A typical example would be getting claims in the HTTP Headers from SATOSA. These arrive prefixed with OIDC_ so a sub claim from the IDP would be

```
      h. := new_template('http');
h.prefix := 'OIDC__'; // required to filter only headers we want
headers. := get_claims(create_source(h2.), 'jgaynor');
```

A typical claim would be

```
   headers.OIDC__sub
http://cilogon.org/serverT/users/12345
```

To rename all of these is easy in QDL with the **`rename_keys`** function:

```
   headers. := rename_keys(headers., keys(headers.)-'OIDC_');
   headers.sub
http://cilogon.org/serverT/users/12345
```

You can add them all to the claims.

```
claims. := claims. ~ headers.
```

or just some subset of them:

```
claims. := claims. ~ [headers.given_name, headers.family_name, headers.eptid];
```

File Claims

An example test file giving the claims structure can be found at test-claims.json.

```
f. := new_template('file');
f.file_path :=
'/home/ncsa/dev/ncsa-git/oa4mp/oa4mp-server-test-oauth2/src/main/resources/test-
claims.json';
file_claims. := get_claims(create_source(f.) , 'jeff');
```

Note that in the sample file, there is also a configured default user, so if you wanted to enable that you would have

```
               f. := new_template('file');
      f.file_path := '/path/to/test-claims.json';
    f.use_default := true;
  f.default_claim := 'default_claim'
     file_claims. := get_claims(create_source(f.) , 'arglebargle');
```

And you would get whatever is the default record for the user.

Creating claims from Java code.

There is a test class for this `edu.uiuc.ncsa.myproxy.oa4mp.oauth2.claims.TestClaimSource` which will simply return a claims object that echos the parameters. Here's how to invoke it. Note that all you need to do is change the java class to your class (make sure its in the classpath!) and pass in your arguments. See the documentation for BasicClaimSourceImpl. In this example, we invoke this test claim source with two custom parameters, foo and bar:

```
           cfg. := new_template('code')
  cfg.java_class := 'edu.uiuc.ncsa.myproxy.oa4mp.oauth2.claims.TestClaimSource'
         cfg.foo := 'foo-test';
         cfg.baz := 'baz-test';

  my_claims. := get_claims(create_source(cfg.), 'jeff');
```

The **`create_source`** function adds required properties. If you display **`my_claims.`** A name must be supplied to the **`get_claims`** function, but the handler actually just ignores it. You get

```
   my_claims.
{
  type:code,
  foo:foo-test,
  baz:baz-test,
  enabled:true,
  fail_on_error:false,
  java_class:edu.uiuc.ncsa.myproxy.oa4mp.oauth2.claims.TestClaimSource,
  info:Echoing configuration parameters.,
  id:qdl_claim_source,
  notify_on_fail:true
  username:jeff
}
```

Note that the default parameters like **`notify_on_fail`** were added by the **`create_source`** function. The username (argument to **`get_claims`**) is simply added.

End to end example of using an LDAP claim source.

This will just grab some attributes from an LDAP server and stash them in the claims (so they will be returned in the ID token).

Enable QDL in your server.

You would need to add the following inside your server config tag:

```
<config>
   <service name=”whatever_you_called_it”>
   <!-- everything else in your config -->
   <qdl name="qdl-default"
            enabled="true"
            debug="info"
            restricted_io="false"
            strict_acls = "false"
            server_mode_on="true"
            script_path="vfs#/scripts/">
           <virtual_file_systems>
               <vfs type="pass_through"
                    access="rw">
                   <root_dir>/PATH/TO/SCRIPTS</root_dir>
                   <scheme><![CDATA[vfs]]></scheme>
                   <mount_point>/scripts</mount_point>
               </vfs>

           </virtual_file_systems>
           <modules>
               <module type="java"
                       import_on_start="true">

             <class_name>edu.uiuc.ncsa.myproxy.oa4mp.qdl.OA2QDLLoader</class_name>
               </module>
           </modules>
           <modules>
               <module type="java"
                       import_on_start="true">
         <class_name>edu.uiuc.ncsa.myproxy.oa4mp.qdl.claims.TokenHandlerLoader</class_name>
               </module>
           </modules>
       </qdl>
   </service>
</config>
```

The only bit of this you need to edit **`/PATH/TO/SCRIPTS`** is to point the virtual file system at a directory that is dedicated. This should be readable by the service only with the right permissions. We will assume that `SCRIPT_PATH=/PATH/TO/SCRIPTS` henceforth and refer to it as $SCRIPT_PATH. Note that the configuration is not aware of environment variables.

This sets up the QDL environment on your server. When you write scripts, they should be in $SCRIPT_PATH

Write the QDL

To access LDAP, you need any credentials (if applicable) as well as the search base. Typically you might create a directory $SCRIPT_PATH/myscripts and in the directory you would put the following script called `get_claims.qdl`

```
                    cfg. := new_template('ldap');
             cfg.auth_type := 'simple';
               cfg.address := 'ADDRESS';
                  cfg.port := 636;
         cfg.fail_on_error := true;
            cfg.claim_name := 'uid';
          cfg.search_base := 'SEARCH_BASE';
                 cfg.type := 'ldap';
            cfg.ldap_name := 'uid';
    cfg.search_attributes. :=  ['email','eppn'];
             cfg.username := 'PRINCIPAL';
             cfg.password := 'PASSWORD';

              new_claims. := get_claims(create_source(cfg.), user);
           claims.'email' := new_claims.'email';
```

You need to supply:

- ADDRESS - the address of the LDAP server
- SEARCH_BASE = the search base
- PRINCIPAL = the principal (akin to a user name for authenticating)
- PASSWORD = the password.

`cfg.search_attributes.` is the entry of interest. It is a list of properties to get from LDAP, here email and eppn. These are stashed with the like named entries in new_claims. You may process them however you want. To assert them, set them in the `claims.` with whatever you want, In this case the email claim will appear in the user's ID token.

Invoking this. This has to be put into the cfg for the client. It is easiest to use the OA4MP CLI (command line client). A typical configuration would look like this

```
tokens{
 identity{
    type=identity
    qdl{
        load="myscripts/get_claims.qdl"
       xmd={exec_phase=["post_token","post_refresh","post_exchange"]}
     }// end qdl
 } //end identity token
 } //end tokens
```

This runs this script when getting tokens, on refresh and on exchanges, so updates to LDAP are always recent.

Start oa2-cli and (assuming your client id is $CLIENT_ID, you'd do this):

```
oa2>use clients
  clients>set_id $CLIENT_ID
id set to $CLIENT_ID
  client>update >cfg
(current cfg if any)
    Enter new JSON value. An empty line terminates input. Entering a line with
/exit will terminate input too.
 Hitting /c will clear the contents of this.
(paste the above block here, then hit return)
```

```
done!
```
Your client should be ready for use.

Suggestions.

This uses QDL – the policy language for OA4MP – and you should get a copy to play with from GitHub: https://github.com/ncsa/OA4MP/releases/download/5.2.4/oa2-qdl-installer.jar (check release number). Install it locally. Note that OA4MP has its own embedded QDL install, so you only need this if you want to experiment with QDL (which I suggest).

Look at the documentation at the [QDL website]

But but but I just need to tweak a single claim

In that case, you can set up QDL to run as minimal as possible. In this example the most minimal QDL configuration is done and the configuration to run a single line of it that appends “@noaa.org” to the subject is done.

```
<qdl name="qdl-default"
           enabled="true"
           debug="info"
           strict_acls="false"
           restricted_io="false"
           server_mode_on="true"/>
```

Then set the following as the client cfg.

```
{"tokens":
  {"identity":
    {"qdl":{
             "code":"claims.'email':=claims.'sub'+'@noaa.org';",
             "xmd":{"exec_phase":["post_token","post_refresh","post_exchange"]}
           },
         "type":"identity"
     }
  }
}
```