Scopes

On scope terminology

Some terminology needs to be centralized here relating to scopes. This is probably the most over-used word in OAuth. In short, ***scopes are requests for either claims in various tokens or request for how to process information***. People use the term scope (for the request) interchangeably with the response to the scope (properly called a claim). It does not help that there is also a claim labeled a**`scope`** in many places (such as in a SciTokens), Scope requests refer to

- **Metadata** (about the user**,** asserted in the ID token**)**
  E.g. **`email, profile`** for a user's email or eppn in the id token.
- **Permissions** (asserted in the access token *if* the format is JWT) These are vetted according to the security policy for the client
  E.g. **`read:/igwn/data/2021`** for access to a resource. These are usually of the form **`action:path_to_resource`**
- **Capabilities** (asserted in the access token, usually vetted then passed along as is)
  E.g. **`compute.modify`**, **`mysql.read`** for use of a component (which may have its own access policies too). These are effectively flags – they are there or not.
- **Processing directives** which will be parsed as requests for specific actions, but are generally not extensible (see below for definition).
  E.g. **`wlcg.capabilityset:/duneana`** which is a directive to a claim source about which set of permissions, capabilities etc. to assert for the user.
  E.g. **`igwn.robot:ligorobot`** which is a directive that the request should be processed as if handled by a robot. This means the user starts the flow to be run as the robot on behalf of the user, and the robot name (here **`ligorobot`**) has its "user" information retrieved and returned in the tokens along with assertions about the user.

Directives and capabilities may also have side-effects such as adding claims to the identity token and refresh tokens as well.

I propose they replace "scope" with the more vivid term, "kitchen-sink," but this does not seem to be catching on.

Semantics of scopes

Aside from the function, there is the practical issue of what the semantics are.

- ***uri-scopes***: Scopes of the form **scheme : path**. E.g. **`storage.create:/home/users/bob`**

- ***simple scopes***: Scopes that are not uris. E.g. **`compute.create`**. These are effectively opaque strings.[1] They cannot be up or down scoped.

Uri-scopes may then be *extensible, i.e.*, the additional path components may be added. This is termed sub or downscoping too (see note below). It is policy as to what is fixed or not and there is no *a priori* method to determine which is which. Since URIs have a regular format and are easily parsed, many institutions use them for everything, which is a good idea.

Handy table relating these concepts:

	fixed	**extensible**
uri	Y	Y
simple	Y	N

Subscope, downscope, scope reduction

Many people also refer to subscopes as *scope reduction* and refer to super scopes "upscopes" and subscopes as "downscopes". The semantics for dealing with subscopes is by path component, so **`read:/home/jeff1`** and **`read:/home/jeff`** are considered unrelated – not super or sub scopes of each other, but **`read:/home/jeff/data`** is a sub-scope of the latter. Subscopes policies are often applied in certain classes of tokens, such as those relating to file permissions, if at all possible.

The policy document

How scopes are vetted *i.e.* interpreted by the server is referred to as the *policy.* It is critical that every client that needs one (not all do!!) have a *policy document* which is a human readable, computer language neutral explanation of how to process requests. In practice, a very large number of mishaps happen if there is no such document because as systems evolve, extensions are required which are not necessarily consistent nor compatible with each other. Then it may happen that the most critical piece of an organization's function – how information and shared and flows – breaks or is otherwise less than reliable. Far too many assume that a clearly written policy need not be done only to be very surprised later. Every interaction with the server is an interaction with its policies.

Standard user metadata scopes supported in OA4MP

OIDC has a standard set of scopes for the ID token, aka user metadata encoded as a JWT. A quick table of these is useful. When a client is created, these are listed as to whether or not the client can request them. For instance, if a public client (so it just wraps if the login worked), you would only want an openid scope rather than a full set of user metadata. If the client is allowed to request the openid scope, then it is assumed to want OIDC compliance.

---

1 Properly they should be a type of URN, in that they name something and are immutable, but they do not follow that syntax. Think of house numbers. A house number is a number, but nobody is going to start adding them because it is understood they are not for computation. So the *policy* regarding these is that they are immutable. This adds another layer of confusion since they look as if you should do something with them but you will never figure that out without context.

Scope	Claims	Description
openid	sub acr amr iat nbf exp auth_time	The subject claim is usually the login name of the user.
address	address address_verified	
profile	name nickname first_name middle_name last_name preferred_name given_name display_name	Not all of these may be asserted by the IDP or elsewhere. If a claim is missing, OA4MP did not get it.
email	email email_verified	The single email address preferred by the user.
phone	phone phone_verified	
org.oa4mp:userinfo	affiliation cert_subject_dn entitlement eppm eptid eduPersonAssurance eduPersonEntitlement eduPersonOrcid idp idp_name isMemberOf[2] itrustuin oidc openid ou pairwise_id subject_id uidNumber voPersonExternalID voPersonID cn dn sn	This scope requests that if the IDP asserts any of these, they will be asserted as claims. If the IDP sends other assertions about the user, they will not be forwarded. For that, use the more permissive org.cilogon.userinfo scope.

2 IsMemberOf can only be asserted if there is some way for OA4MP to get the groups of the user. This is normally set in the policy document.

org.cilogon.userinfo	(anything the IDP asserts)	This is much more permissive and will return everything the IDP asserts, including the list for org.oa4mp:userinfo. If both of these scopes are present, this one is honored
edu.uiuc.ncsa.myproxy.getcert	--	This is presented to the getcert endpoint which returns an X 509 chain of certificates, ***if and only if*** OA4MP is setup to issue X 509 certificates (which is rare, but possible). Note that the cert_subject_dn claims will only be available if this scope is supported.

Key:

Scope is the scope you pass in,
Claims are the claims asserted in the ID token,
Description is just a note about these.

Do note that scopes are usually consumed and not returned in any token.

Access token scopes

There are various standards for acccess tokens which try to capture these, in particular the WLCG specification and the SciTokens specification. Refer to those documents. The other major specification, RFC 9068 does not specify scopes, so you can set those however you like.

Appendix. Sample tokens

These are given with the request parameters that result in them. Note that the claims are standard:

- **`aud`** The entity that is the intended audience, *i.e.* recipient of this token. Most usually that is the client or the resource server.
- **`sub`** The entity that is the subject of the token. The claims in the token are assertions about this subject
- **`iss`** The URI of the service that created *i.e.* issued the claim
- **`jti`** is simply a unique identifier and every token gets one
- **`iat`** (issued at) the timestamp in seconds at which the token was issued
- **`nbf`** (not valid before) the timestamp in seconds at which the token becomes valid.
- **`exp`** (expiration) the timestamp in seconds at which the token becomes invalid.

Note that the tokens are all JSON, but to facilitate reading, most of the syntax is elided. If a value is too long to display as a single line, line breaks are added for readability. The aim is to let you parse the token as much as possible in a single glance.

ID token

Requested ID token scopes: **`openid profile email org.oa4mp:userinfo`**

The server is configured to assert a custom claim named `ϑ` which is the arcsine of iat/exp. Why? Just showing off. This was done with scripting and as we said, if you can articulate it in a policy OA4MP can very probably do it.

```
    affiliation : staff@ncsa.illinois.edu;
                  employee@ncsa.illinois.edu;
                  member@ncsa.illinois.edu
            aud : oa4mp:/noms-wg/rfc9068
      auth_time : 1732892682
          email : gaynor@illinois.edu
            exp : 1732893587
            iat : 1732892687
            idp : http://github.com/login/oauth/authorize
       idp_name : GitHub
     isMemberOf : [CO:COU:math:members:active,
                   CO:COU:noms:members:active,
                   CO:COU:staff:members:active]
            iss : https://oa4mp.oxbridge.edu/noms
            jti : https://oa4mp.oxbridge.edu/idToken/
                  2b37d85f93bfa5a8d35b87345d36352b/1732892682654
            nbf : 1732892687
          nonce : KEOIIT4G382rS3HN-EGO8G8ZaF-4K9lSG0QG_mzX60w
           oidc : 2953537
            sub : d3668a736945b1081e9b5fc5387a3867fe80ca44
              ϑ : 1.56977714812856
```

Access token

In this case, the type of the token is an RFC 9068 token and the scopes are for permissions to run a database engine. The request contains

Requested access token scopes: `db:init:table` `db:create:table`

and based on the user's permissions (gotten from a 3rd Party), the resulting permissions are for the database mysql and the table the user can manage is called "users". The resource server knows how to use these scopes.

```
          aud : https://math.oxbridge.edu/noms-wg
    auth_time : 1732892682
    client_id : oa4mp:/noms-wg/rfc9068
          exp : 1732892987
          iat : 1732892687
          iss : https://oa4mp.oxbridge.edu/noms
          jti : https://oa4mp.oxbridge.edu/3ecfdec7abf18b0b1c3aefa6439d5f?
                type=accessToken&
                ts=1732892687204&
```

```
        version=v2.0&
        lifetime=300000
  nbf : 1732892682
scope : mysql:init:users mysql:create:users
  sub : d3668a736945b1081e9b5fc5387a3867fe80ca44
```

Refresh token

No parameters sent, this is simply a generic unsigned token. All that matters in most cases is that it comes as the response from the server and is a JWT.

```
  aud : oa4mp:/noms-wg/rfc9068
  exp : 1732893587
  iat : 1732892687
  iss : https://oa4mp.oxbridge.edu/noms
  jti : https://oa4mp.oxbridge.edu/5e5fbfb609d08f83a3da127d399?
        type=refreshToken&
        ts=1732892687204&
        version=v2.0&
        lifetime=900000
  nbf : 1732892682
```

Since the initial request that starts the flow must contain all of the scopes, the actual complete scope parameter (note it is blank delimited!) is

openid profile email org.oa4mp:userinfo db:init:table db:create:table

Appendix. The policy for this client

The following is the policy document that produces the above tokens. It is included to form a more complete example.

Prolog

- Point of contact for this policy is finn.mccool@math.oxbridge.edu.
- Client id : `oa4mp:/noms-wg/rfc9068`
- Issuer is : `https://oa4mp.oxbridge.edu/noms` and is a virtual issuer
- Resource server : `https://math.oxbridge.edu/noms-wg`
- Device and authorization code flows are supported

Authorization

- The allowed IDP list for this client should only include:
  - Oxbridge university
- CILogon should be used for login.
- The CILogon unique identifier (uid) should be returned
- User groups are found in LDAP ldap-math.oxbridge.edu with search base
  `ou=people,o=Oxbridge,o=CO,dc=dev,dc=math,dc=edu`
  under the CILogon uid.
- Tokens can only be obtained by members in the group `CO:COU:math:members:active` or `CO:COU:noms:members:active`. A rejection here results in user not found error. (See below)
- Identity linking is done using the uid resolved against the LDAP server in the voPersonID assertion there. A failure to find the user there results in a linking error (See below)

ID tokens

- Subject: sha1sum of CILogon uid. This is to conform to anonymization requirements.
- Allowed scopes are any subset of
  `openid`
  `email`
  `profile`
  `org.cilogon.userinfo`
- Other claims:The groups from LDAP are returned in the **`isMemberOf`** claim as an array of strings.

Access tokens

- Tokens are RFC 9068
- Lifetime: 4 hours.
- Subject: sha1sum of CILogon uid. This is to conform to anonymization requirements.
- Audience: Should be set to the resource server URI or any the client passes in during the initial request.
- Other claims: If users are in the group `CO:COU:noms:members:active` , then the access token must assert the claim “grant_id” with the value “NSF-123456”.
- Allowed scopes are
  `db:init:table`
  `db:create:table`
  `db:delete:table`
  `db:modify:table`
- Allowed database permissions are found in LDAP, ldap-math.oxbridge.edu with search base `ou=services,o=Oxbridge,o=CO,dc=dev,dc=math,dc=edu`
  under the CILogon uid. The database and table will be set from this.

Refresh tokens

- JWT format
- Lifetime: 7 days
- No subject

Error handling

Authorization denied handling

- Users that are denied authorization at the issuer should be redirected to the help page located at https://math.oxbridge.edu/registry-login-error.php
  - The following entries should be sent
    - error
      - Same as would normally go to client callback uri
    - error_description
      - Same as would normally go to client callback uri
    - service_id
      - “no_auth”

No linking handling

- Users that are authorized but for whom there is no identity linking at the issuer should be redirected to the help page located at https://math.oxbridge.edu/registry-linking.php
  - The following entries should be sent
    - error
      - Same as would normally go to client callback uri
    - error_description

        - Same as would normally go to client callback uri
    - service_id
        - “no_linking”

Epilog

The **sub** in this case is the internal CILogon identifier for the user. Since that is guaranteed unique, many clients (such as this one) use that for identifying their users.
The issuer is the OA4MP server. In this case, it is configured as a virtual issuer, with its own set of signing keys, for the NOMS group. The resource server is the server that interprets the access token and serves/manages resources (such actually copying a file or in this case, hosting a database engine).

The authorization policy allows for users who are not at the institution (oxbridge.edu) to use GitHub as an alternate login and link their account to their set of permissions. It does this by handing off the logon to CILogon (which, incidentally, is an OA4MP server too!). This is why this server uses the CILogon user id as the subject claim – it denotes the user uniquely across logins. Linking is managed by Oxbridge and recorded in their LDAP.
