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 the claim itself is labeled as **`scope`** in many places (such as in a SciToken), In tokens, scope requests refer to

- **Metadata** (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 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, and the robot name (here **`ligorobot`**) has its "user" information retrieved and returned in the tokens.

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.

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

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 types of tokens if at all possible.

Vetting scopes in OA4MP

How scopes are vetted *i.e.* interpreted by the server is referred to as the *policy.* In the case of OA4MP the two major ways to do this are via scripting or templates.

The policy document

It is critical that every client 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 is trivial and need not be done only to be very surprised later.

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.

---

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 there scopes are 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 claim jti is simply a unique identifier and every token gets one, along with iat (issuer at), nbf (not valid before) and exp (expiration timestamp in seconds). The sub in this case is the internal CILogon identifier for the user. Since that is guaranteed unique, many installs use that for identifying their users.

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.

The initial request will contain all of the scopes,

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 we can very probably do it.

```
affiliation : staff@ncsa.illinois.edu;
              employee@ncsa.illinois.edu;
              member@ncsa.illinois.edu
        aud : localhost:test/rfc9086
  auth_time : 1732892682
      email : gaynor@illinois.edu
        exp : 1732893587
        iat : 1732892687
        idp : http://github.com/login/oauth/authorize
   idp_name : GitHub
        iss : https://localhost:9443/oauth2
        jti : https://localhost:9443/oauth2/idToken/
              2b37d85f93bfa5a8d35b87345d36352b/1732892682654
        nbf : 1732892687
      nonce : KEOIIT4G382rS3HN-EGO8G8ZaF-4K9lSG0QG_mzX60w
       oidc : 2953537
        sub : http://cilogon.org/serverT/users/213363
          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 consumer of this token knows how to use these scopes.

```
      aud : https://test/rfc9068/aud
auth_time : 1732892682
client_id : localhost:test/rfc9086
      exp : 1732892987
      iat : 1732892687
      iss : https://localhost:9443/oauth2
      jti : https://localhost:9443/oauth2/3ecfdec7abf18b0b1c3aefa6439d5f?
            type=accessToken&
            ts=1732892687204&
            version=v2.0&
            lifetime=300000
      nbf : 1732892682
    scope : mysql:init:users mysql:create:users
      sub : http://cilogon.org/serverT/users/213363
```

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 : https://localhost/rfc9086
      exp : 1732893587
      iat : 1732892687
```

```
jti : https://localhost:9443/oauth2/5e5fbf4bab9b609d08f83a3da127d399?
      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