

Policy Documents

A *policy document* refers to human readable, machine language neutral document which makes clear how the server processes requests by the client. In particular, how scopes are to be processed by the server. Each client may have a policy in OA4MP, although the basic functioning does not require one. Policies are for when basic OAuth is insufficient. There are very simple user metadata policies in effect if the client is, say, an OIDC client, but there are no general policies in effect for a wide variety of behaviors on a server. The type of access token used, such as WLCG, RFC 9068 or SciToken are mostly concerned with semantics of parts of the token and may bring some rudimentary policies along, but these are usually insufficient for most practical needs.

For example, the [SciTokens specification](#) carefully outlines the semantics of the returned permissions in an access token, but it cannot possibly articulate how these are determined. So the fact that a permission like `read:/home/fionna/public/data/nsf-12345` is asserted to user Bob may actually involve looking up Bob's group memberships, checking he is currently on the given grant and that he is given explicit permission to read (but possibly not write) the given path. The point is that quite obviously the SciTokens specification cannot reasonably know such a thing.

On the need for a policy document.

The [OIDC specification](#) sets some basic policies on the ID token, such as asserting the `sub` claim. See the [scope document](#) for more. Clients that only need that do not need additional policies. Public clients, which are OIDC clients that essentially wrap whether or not the user has a valid login also need no additional policies. In a well-run OA4MP instance, however, clients that have processing needs above and beyond the basic should always have one. The practical reason is that as systems grow, extensions to existing policies are invariably needed. In such a case, if the policies are not carefully managed, client behavior may become "predictable and unwanted." The correct workflow is to update the policy document and then update the implementation, and no changes to the implementation happen without a change to the policy document first.

Policy documents should reside some place public and be world-readable but have a small set of committers that can change them. It is common to put them under version control *e.g.* in GitHub. The correct workflow is that if something is not in the policy document, it does not exist and policy changes should be done carefully with good consideration for the effect. Remember that a bad set of policies can cripple the ability of the organization to operate.

Structure of the policy document

There is no required format, but the major sections should cover various parts of the flow. Tokens all require audience, subject, lifetime and issuer. There are defaults for these but if there are special requirements they should be stated. Note that you should be aware of the requirements of the service. Tokens are usually validated based on the issuer, audience and lifetime, so be sure your policy produces valid tokens.

OA4MP allows you to assert any claim you want in a token as part of your policy. For instance, some clients require grant information to be asserted based on group memberships so they can track resource usage.

A great source of confusion is what tokens are vs. their content. Many articles are unclear because they assume that every token is a JWT and some even go so far as to refer to the access token as “the JWT”. This is not the case at all. ID tokens are signed JWTs (JSON web tokens) as per the OIDC specification. Access and refresh tokens, as per the OAuth specification, are simply opaque strings¹. OA4MP returns opaque strings unless the policy states otherwise. Having access tokens that are WLCG, SciToken, &c. compliant means the token is a JWT.

Finally, OA4MP works with virtually every OAuth 2 client library in existence and a few of these have non-standard quirks. Be sure your take these into account in designing you policy.

Prolog

This should be generally informational and at least contain the clients (by ID) for which this applies. It is also possible to apply it to every administered client. Flows that this policy supports (such as device flow, authorization code flow et al) should be noted.

Authorization

This relates to who can authorize, such as restricting users to a specific IDP or group membership. It should articulate rejection modes and (if applicable) any signing requirements. If you are using group memberships as part of the policy logic, be sure to tell OA4MP where it can find them. OA4MP is usually considered a trusted party, so if the information is not world readable, OA4MP needs some authorization mechanism to access it.

ID tokens

You will get these normally if your client is an OIDC client and sends the openid scope. This token is, in fact, an assertion of the user’s metadata, such as (potentially) the name, authorization time, various bits of information asserted by the IDP at login. The format of these is always a JWT.

Access tokens

Opaque tokens cannot assert scopes (permissions), so if you need the scope claim asserted, you need to specify which to use. The major access token specifications are [WLCG](#), [SciTokens](#) and [RFC 9068](#). The first two specify the format off the permissions and capabilities, the latter does not. The choice will mostly be driven by interoperability with whatever processes the permissions. In this section, you list what requests (passed in as “scope”) corresponds in what permissions, capabilities etc. This may involve articulating a mechanism (e.g. LDAP, RESTful service, database lookup, &c., &c.) where these reside.

1 Here’s a typical basic (opaque) OA4MP access token:

NB2HI4DTHIXS63DPMNQWY2DPON2DU0JUGQZS633B0V2GQMRPG44DG0DBGU4DA0BSMQYGKMBSGI2DSNDDGM3WC0BRGA2GENLG
HBST65DZ0BST2YLDMNSXG42UN5VWK3RG0RZT2MJXGMZDMNJWHA3DS0BTGQTHMZLS0NUW63R50YZC4MBGNRUWMZLUNFWWPJZ
GAYDAMBO

which is a base 32 encoding of the actual token

<https://localhost:9443/oauth2/7838a58082d0e022494c37a8104b5f8e?>
`type=accessToken&ts=1732656869834&version=v2.0&lifetime=900000`

If you are getting one of these and don’t know what it is, you probably need to specify the token format to be JWT.

Specific behavior on getting an access token, refresh and exchanges of them (*i.e.* as per this table as listed by grant type to the token endpoint) should be clarified here as well.

Action	Grant type
Get the very first access token. The first two grant types are standard user facing, the last two are used by services	authorization_code device_code client_credentials urn:ietf:params:oauth:grant-type:jwt-bearer
Refresh the access token and any refresh token	refresh_token
Exchange one of the id, access or refresh tokens for a new one	urn:ietf:params:oauth:grant-type:token-exchange

Refresh tokens

Refresh tokens can be a JWT and many client libraries require them to be so, mostly to validate them against the server, but there is rarely content in them *per se*. There are at this writing no actual specifications adopted for the format of these but the common usage pattern is that they are unsigned. If your library requires a JWT, just ensure that your policy returns one it will validate. This may mean setting the subject or audience to something specific. Scopes are generally not asserted in refresh tokens.

Error handling

In the event of any errors that arise, what should OA4MP do? Generally in the authorization phase of the code flow, OA4MP can redirect to another site. Failing that, it will show an error page or perhaps simply forward the user to the home page for the client. Errors at other points of the flow (*e.g.* during token exchange) have no user facing component so no redirect is possible, but customizing error messages is possible.

Epilog

Final observations, notes, etc.

An Example Policy Document

In the following example, a project named NOMS (for non-orientable minimal surfaces) Working Group needs an issuer on an OA4MP server. It does two lookups in the institution-run LDAP, first for the user's groups, then if the user is in the correct group(s), access token permissions are looked up based on the user's ID.

NOMS Working Group Policy Document

Prolog

- Author of this policy : cú.chulainn@math.oxbridge.edu

- Client IDs: oa4mp:/client_id/noms/fts, oa4mp:/client_id/noms/data_explorer
- Issuer : <https://oa4mp.oxbridge.edu/noms>
- Resource server : <https://wlcg.cern.ch/jwt/v1/any> (WLCG default)
- Both device and authorization code flow are supported
- All tokens should be signed with EC ES512 algorithm

Authorization

- The allowed IDP list for this client should only include:
 - Oxbridge university
- User groups are found in LDAP ldap-math.oxbridge.edu with search base
`ou=people,o=Oxbridge,o=C0,dc=dev,dc=math,dc=edu`
 under the username.
- Tokens can only be obtained by members in the group `C0:COU:math:members:active` or
`C0:COU:noms:members:active`

ID tokens

- Subject: head of user's EPPN
- 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

- Access token permission query allowed in authorization request.
- Use WLCG type for access tokens
- Lifetime: 4 hours.
- Subject: sha1sum of client ID.
- Audience: Should be the same as the resource server or to that requested by client in the initial request.
- Other claims: If users are in the group `C0:COU:noms:members:active`, then the access token must assert the claim “grant_id” with the value “NSF-123456”.
- Allowed scope are any subset of the following permissions:
`storage.create:/`
`storage.read:/`
`storage.modify:/`
- Only `oa4mp:/client_id/noms/data_explorer` can request any of the following capabilities in addition to the above:
`compute.read`
`compute.cancel`

```
compute.create  
compute.modify
```

- Base storage paths are found in LDAP, ldap-math.oxbridge.edu with search base ou=services, o=0xbbridge, o=c0, dc=dev, dc=math, dc=edu under the username.

Refresh tokens

- Refresh tokens are JWT format
- Subject: sha1sum of the client ID
- No scope claim is asserted.
- Lifetime: 30 days.

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”

Epilog

- Each token has a lifetime, audience, subject and issuer. There are defaults applied for each, so omitting them means you get the default. If you want them to be overridden in some way, please make that clear.
- Errors during authorization typically just sends the user to a portal at the institution where the user can register. The error and error_description are standard OAuth, and in this case, the institution requires a “serviceid” so it can route the user to the correct webpage. This is optional and not all institutions do it.
- You can assert any claims you want as additional claims.

A typical request for this client might have scopes

```
openid profile storage.read:/home/bob/public/data storage.write:/home/bob/nsf-123456  
for user bob@math.oxbridge.edu.
```

A common use pattern: Querying the server

The above example allows for querying the server during authorization. This is becoming a more common use pattern. This means that if the requested scopes can be generic and the returned values

will be all the basic scopes that can be requested. This gives the maximum set of permissions possible for a given authorization E.g. if the authorization request contains

```
storage.read:/  
compute.modify
```

The scopes for the given user returned in the (first) access token might be

```
storage.read:/home/bob/public  
storage.read:/home/bob/nsf-123456  
storage.read:/home/public/noms  
compute.modify
```

This is a list of all the locations that the user is allowed to read from, plus compute.modify flag (which cannot be qualified, since it is merely present or not). Subsequent refreshes and exchanges may use any of these including downscopes. Not all policies use this and it should be made explicit.