Token renewal and exchange in IAM

Andrea Ceccanti
WLCG AuthZ WG

Oct. 14th 2021

OAuth/OIDC Token renewal

Background: The OAuth refresh token flow

- Mechanism to implement the ability for an application to act on behalf of a user and get tokens without user's interaction

- Used by a client to refresh an access token that is about to expire using a refresh token (RT) obtained in a former authorization flow

- **Authenticated** POST request to the token issuer token endpoint
  - Client credentials **and** a valid RT must be provided by the caller
  - Produces a new access token and possibly an updated refresh token

- The scope request parameter can be used to attenuate the token privileges, by requesting a subset of the scopes linked to user authorization grant (RFC 6749)

- The audience request parameter can be used to suggest an audience for the requested access token (in IAM RFC 8693 approach *stretched* to apply also to the RT flow implementation; RFC 8707 will soon be supported)

Refresh tokens in IAM

In IAM, the refresh token flow can be enabled or disabled per client.

RTs can be configured, at client level, to be usable multiple times or be valid for a single refresh. When the RT is configured for single use, each time the RT is used a new RT is returned giving the same privileges (and the previous one is expired).

RTs can have an expiration date, or be unbounded in validity. This depends on the client configuration. Tokens validity settings in IAM can only be changed by administrators.

RTs can be revoked/invalidated using a standard OAuth API.

Refresh tokens: use cases

How long do we want a user “session” to last? That’s the lifetime of the refresh token

Example: users on a CLI shouldn’t be prompted for login more than once a week

RT lifetime: a week

Example minimal refresh token request

```
POST /token HTTP/2
Host: wlcg.cloud.cnaf.infn.it
Authorization: Basic YW5kcmVhLmV4YW1wbGU6Vn…
Accept: */*
Content-Type: application/x-www-form-urlencoded
Content-Length: 144

refresh_token=eyJhbGciOiJub25lIn0.eyJleHAiOjE2Mz…
```

client credentials **and** a valid refresh token are needed to get a new access token

also, RTs are client-specific, i.e. an RT issued to client A cannot be used by Client B

JWT-based client authentication

JWT-based client authentication

The OAuth and OpenID Connect protocols support **JWT-based client-authentication,** which means that clients authenticate to the token issuer sending **a signed JWT** instead of a (client_id, client_secret).

The token issuer inspect the JWT, resolves the client_id and verifies the JWT using either a shared secret or a public key linked to the client configuration

Pros

- time-limited client credentials under the control of the client

Cons

- clients need to know how to generate and sign a JWT

JWT client auth methods

https://datatracker.ietf.org/doc/html/rfc7523

https://openid.net/specs/openid-connect-core-1_0.html#ClientAuthentication

`client_secret_jwt:` shared secret scheme, clients that have received a client_secret value from the Authorization Server create a JWT using an HMAC SHA algorithm, such as HMAC SHA-256.

`private_key_jwt:` Clients that have registered a public key sign a JWT using the corresponding private key.

Main WLCG possible use cases

Reduced risk of exposed client credentials

JWT-based auth is a required for high security OpenID-connect use, e.g., the Financial Grade API OpenID Connect profile

Time-limited credential delegation

Examples:

RUCIO server delegates short-lived JWT client credential to RUCIO client that can be used for time-limited token renewal

VO job framework delegates short-lived JWT client credential to payload job for time-limited token renewal

JWT-auth support in IAM

MitreID connect library supports JWT-based client auth, but IAM does not.

Support will be added back in the next release (1.8.0).

Token exchange

Token exchange objectives

Token exchange in IAM allows a client to request the exchange of an access token with another access token (and potentially a refresh token to renew such access token).

This is useful to implement **controlled delegation of privileges** between two registered IAM client applications.

What privileges?

Basically, a subset of the scopes linked to the token being exchanged. Or a subset of the scopes allowed for the clients.

Example:

I want to move some of my files with RUCIO. So I give RUCIO permission to act on my behalf to make this file movement happen. RUCIO then delegates this task to FTS, which still acts on my behalf to trigger third-party transfers across Storage Elements. Here two different client apps act on my behalf. Different scopes are needed at different level of the infrastructure: token exchange allows to provide tokens with minimum privileges to each service without requiring that big fat tokens are used at the top of the chain.

Client configuration requirements

In order to request a token exchange, a client must be configured with the `urn:ietf:params:oauth:grant-type:token-exchange` grant type enabled.

The token exchange grant type is **disabled by default for dynamically registered clients**, can only be enabled by IAM administrators.

It is expected that token exchange is typically enabled only for a few trusted clients (VO central services).

Delegating offline_access privileges

IAM supports the ability to delegate offline_access privileges across trusted client applications using token exchange, i.e. exchange an access token for a longer-lived refresh token.

The lifetime of the RT depends on the client configuration, i.e., unbounded renewal can be prevented using appropriate configuration.

A token obtained with a token exchange cannot be further exchanged by the same client

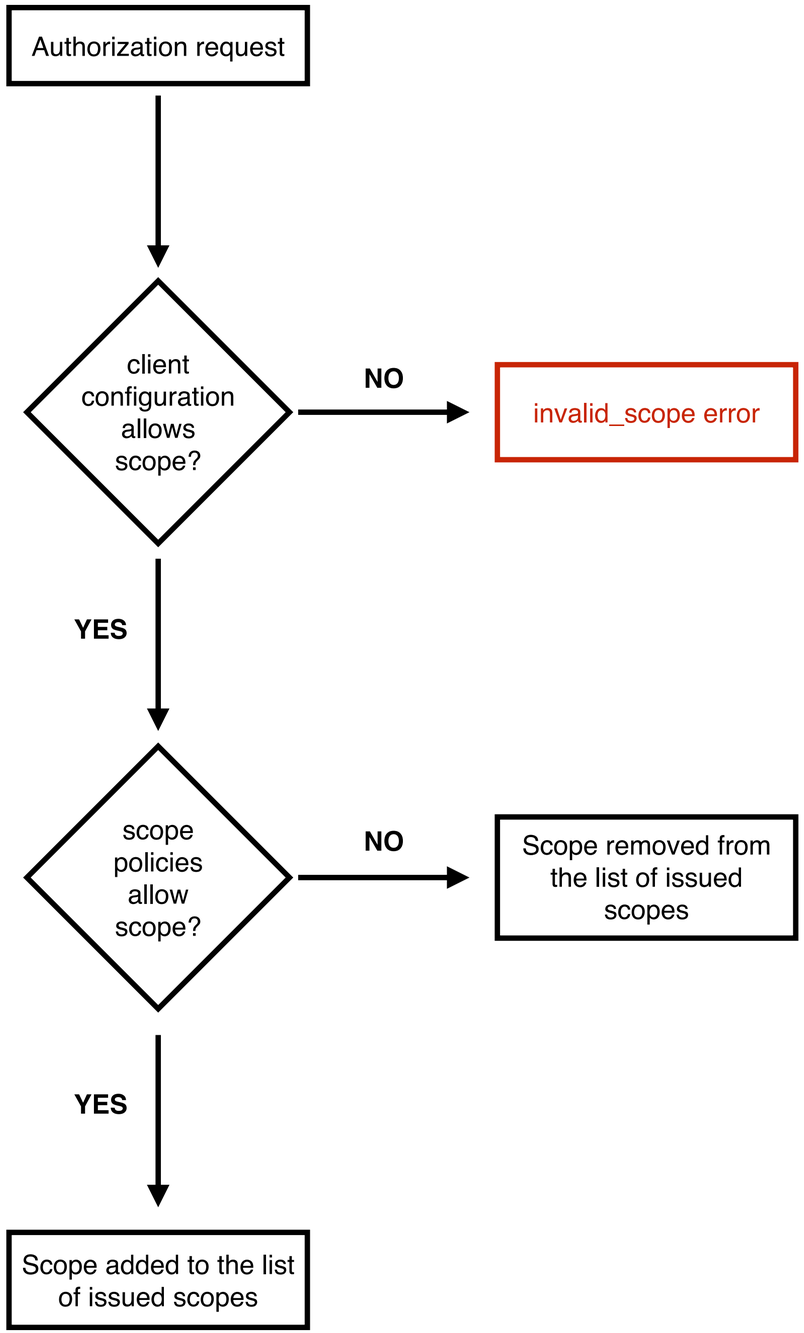
OAuth scopes control in IAM

In IAM each registered client has a list of allowed OAuth scopes.

When a client requests a scope that is not allowed to get, it gets an **invalid_scope** error.

On top of this scope vetting happening at the client level, there's an additional scope filtering used **to limit access to scopes based on the user identity**, i.e., to limit access to selected scopes only to selected users (or group of users)

- The policy that drive this additional scope filtering are managed using the IAM Scope Policy API



IAM Scope policy API

Applies to all OAuth/OIDC grant types, i.e. all ways to get a token out of IAM.

Scope policy API documentation

A scope policy defines:

- a rule, which can be PERMIT or DENY, that determines the behaviour of the policy
- a scope selector, i.e. a set of scopes for which the policy applies (and a scope `matchingPolicy` used to determine the scope matching algorithm used);
- an account or group selector, used to determine for which user account or group of accounts the policy should apply

Example Scope policies (1)


```
[
...,
{
    "id": 4,
    "description": "Deny access to compute.* scopes to normal users",
    "creationTime": "2019-12-18T15:11:04.000+01:00",
    "lastUpdateTime": "2019-12-18T15:11:04.000+01:00",
    "rule": "DENY",
    "matchingPolicy": "EQ",
    "account": null,
    "group": null,
    "scopes": [
      "compute.create",
      "compute.read",
      "compute.cancel",
      "compute.modify"
    ]
  },
```

Example Scope policies (1)

```
[
...,
{
    "id": 4,
    "description": "Deny access to compute.* scopes to normal users",
    "creationTime": "2019-12-18T15:11:04.000+01:00",
    "lastUpdateTime": "2019-12-18T15:11:04.000+01:00",
    "rule": "DENY",
    "matchingPolicy": "EQ",
    "account": null,
    "group": null,
    "scopes": [
      "compute.create",
      "compute.read",
      "compute.cancel",
      "compute.modify"
    ]
  },
```

The policy description is used to describe the effect of a given scope policy

Example Scope policies (1)


```
[
...,
{
    "id": 4,
    "description": "Deny access to compute.* scopes to normal users",
    "creationTime": "2019-12-18T15:11:04.000+01:00",
    "lastUpdateTime": "2019-12-18T15:11:04.000+01:00",
    "rule": "DENY",
    "matchingPolicy": "EQ",
    "account": null,
    "group": null,
    "scopes": [
      "compute.create",
      "compute.read",
      "compute.cancel",
      "compute.modify"
    ]
  },
```


The policy rule: DENY is used to block requests for any scope that matches the scope selector, PERMIT to allow access to scopes

Example Scope policies (1)

```
[
...,
{
    "id": 4,
    "description": "Deny access to compute.* scopes to normal users",
    "creationTime": "2019-12-18T15:11:04.000+01:00",
    "lastUpdateTime": "2019-12-18T15:11:04.000+01:00",
    "rule": "DENY",
    "matchingPolicy": "EQ",
    "account": null,
    "group": null,
    "scopes": [
      "compute.create",
      "compute.read",
      "compute.cancel",
      "compute.modify"
    ]
  },
```

Scope selector: this policy will apply to any request involving any of these scopes

Example Scope policies (1)

```
[
...,
{
    "id": 4,
    "description": "Deny access to compute.* scopes to normal users",
    "creationTime": "2019-12-18T15:11:04.000+01:00",
    "lastUpdateTime": "2019-12-18T15:11:04.000+01:00",
    "rule": "DENY",
    "matchingPolicy": "EQ",
    "account": null,
    "group": null,
    "scopes": [
      "compute.create",
      "compute.read",
      "compute.cancel",
      "compute.modify"
    ]
  },
```

Scopes will be matched using a string equality algorithm

IAM supports also regexp and path algorithm,

as documented here

Example Scope policies (1)

```
[
...,
{
    "id": 4,
    "description": "Deny access to compute.* scopes to normal users",
    "creationTime": "2019-12-18T15:11:04.000+01:00",
    "lastUpdateTime": "2019-12-18T15:11:04.000+01:00",
    "rule": "DENY",
    "matchingPolicy": "EQ",
    "account": null,
    "group": null,
    "scopes": [
      "compute.create",
      "compute.read",
      "compute.cancel",
      "compute.modify"
    ]
  },
```

Group and account selector are null, so this policy will apply to any request

Example Scope policies (2)


```
{
  "id": 13,
  "description": "Allow access to compute.* scopes to wlcg/pilot users",
  "creationTime": "2019-12-18T15:19:20.000+01:00",
  "lastUpdateTime": "2019-12-18T15:19:20.000+01:00",
  "rule": "PERMIT",
  "matchingPolicy": "EQ",
  "account": null,
  "group": {
    "uuid": "25084f30-1d71-4ab2-91e8-11148af16682",
    "name": "wlcg/pilots",
    "location": "https://wlcg.cloud.cnaf.infn.it/scim/Groups/25084f30-1d71-4ab2-91e8-11148af16682"
  },
  "scopes": [
    "compute.create",
    "compute.read",
    "compute.cancel",
    "compute.modify"
  ]
},
...]
```


This policy will apply only to members of the wlcg/pilots group

Example Scope policies (2)


```
  {
    "id": 13,
    "description": "Allow access to compute.* scopes to wlcg/pilot users",
    "creationTime": "2019-12-18T15:19:20.000+01:00",
    "lastUpdateTime": "2019-12-18T15:19:20.000+01:00",
    "rule": "PERMIT",
    "matchingPolicy": "EQ",
    "account": null,
    "group": {
      "uuid": "25084f30-1d71-4ab2-91e8-11148af16682",
      "name": "wlcg/pilots",
      "location": "https://wlcg.cloud.cnaf.infn.it/scim/Groups/25084f30-1d71-4ab2-91e8-11148af16682"
    },
    "scopes": [
      "compute.create",
      "compute.read",
      "compute.cancel",
      "compute.modify"
    ]
  },
...]
```


For the compute.* scopes…

Scope policies combination rules

There are three levels of scope policies:

- Default policies, i.e., those with empty group and account selectors
- Group-level policies
- Account-level policies

When multiple policies match a request, the combination rules are as follows:

Account-level policies > Group-level policies > Default policies

i.e. account level policies take precedence over group level policies which take precedence over default policies.

At the same level, DENY policies win over PERMIT policies

Scope policy combination example

Taking the scope policies example in the previous slides:

the DENY rule that blocks access to compute.* scopes is applied to all users, with the exception of users in group wlcg/pilot, where the PERMIT policy is applied.

The PERMIT policy takes precedence over the DENY one because it’s more specific, i.e., defined at group level.

Limiting Token exchange with the token exchange policy API

Only clients that have the token exchange grant type enabled can attempt a token exchange request.

The Token Exchange Policy API can be used to further limit the exchanges only across selected clients and for selected scopes.

An example token exchange policy


```
{
    "id": 2,
    "description": "Allow all exchanges",
    "creationTime": "2021-08-05T14:38:52.000+02:00",
    "lastUpdateTime": "2021-08-05T14:38:52.000+02:00",
    "rule": "PERMIT",
    "originClient": {
      "type": "ANY"
    },
    "destinationClient": {
      "type": "ANY"
    }
  }
```

An example token exchange policy


```
{
    "id": 3,
    "description": "Allow exchanges for openid and offline_access scopes for client token-exchange-actor",
    "creationTime": "2021-08-05T14:46:58.000+02:00",
    "lastUpdateTime": "2021-08-05T14:46:58.000+02:00",
    "rule": "PERMIT",
    "originClient": {
      "type": "ANY"
    },
    "destinationClient": {
      "type": "BY_ID",
      "matchParam": "token-exchange-actor"
    },
    "scopePolicies": [
      {
        "rule": "PERMIT",
        "type": "EQ",
        "matchParam": "openid"
      },
      {
        "rule": "PERMIT",
        "type": "EQ",
        "matchParam": "offline_access"
      }
    ]
  }
```