Cisco Notes

Yann Esposito

RBAC for clients [2023-04-07 Fri]

Effects for Q4

Visible Changes

Which changes to expects:

1. The User model field `role` could have more than just `admin` or `user`.
2. The access tokens (JWT) claim for `role` will also have the same new potential value.

Expected roles for Q4: `admin,user,sat,sec-eng,it-ops,observer`

During Q4; potentially frequent changes of permission for every role, in particular for application external to IROH.

JWT detail

Claim prefix for unique identifier: "https://schemas.cisco.com/iroh/identity/claims"

The claim .../role will have more values than just `user` and `admin`.

Expect this to change, and potentially, the roles could be entirely random ids without any central static table.

- 1st step: from: `admin,user` to `admin,user,sat`
- Then more roles will be added; for now `admin,user,sat,sec-eng,it-ops,observer`

Respect Permissions from tokens

Roles should be easily added/removed, and we even prepared the ability to add "custom roles". We potentially want to easily changes the permissions associated to roles. Thus all user's permissions should pass via *scopes*.

To check if a token provide some permission you should only check the scopes of this token. The recommended way to do that is to use the `/iroh/profile/permissions` endpoint.

1. permissions endpoint (recommended)

   This endpoint provides a way to ask with a single HTTP call multiple different permissions questions using a token:

   You provide the endpoint a body with a JSON Object with the following format:

   ```
   {"widget-1": ["inspect", "response:read"],
    "can-do-x": ["scope-1", "scope-2/sub-scope"]
    "xdr":      ["cisco/feature-flag/xdr"]}
   ```

   And you get back a JSON Object with boolean values:

   ```
   {"widget-1": true,
    "can-do-x": false
    "xdr":true}
   ```

   Using this endpoint will also provide you the opportunity to change your client configuration to use a new JWT format that is a lot smaller (guaranteed to be <4kB).

2. check scopes directly (not recommended)

   It is also possible to retrieve the tokens by decoding the JWT directly (or also calling the `/iroh/profile/scopes` endpoint). The main issue with directly checking the list of scopes is that IROH scopes have a tree-like structure with specific rules and in order to duplicate the permissions endpoint you need to have a local duplicate library able to understand this scope structure. For all the technical details see https://github.com/threatgrid/scopula/

   Notes:

   (a) If an entity can have the scopes: `foo` `foo/permission-1` `foo/permission-2` it will

   be compressed as `foo` only. So it is up to the client to understand that the set of scopes `foo` also implicitly contains `foo/permission-1` and `foo/permission-2`.

   (a) While this is not recommend in general, this could be preferred for very simple

   permission synchronization. For example, for `orbital` we only have 3 cases, `orbital`, `orbital:read`, nothing. No sub-scopes involved, not many specific permission to manage.

3. Why not check for roles in the JWT?

   Say your application should allow be used by admins but not allow any other role. Say we create a specific scope for your application `my-app`.

Currently any admin can create an OAuth2 client without the `my-app` scope and thus expect this client not to be allowed to use your application.

If your application only check the role there is no way to construct a client for an admin that is not allowed to use your application. Worse during Client creation and Client Authorization, we display a UI that explain the permissions associated to every scope.

Pushing your internal permissions inside IROH

If you want PMs to easily change the permissions associated with some role for your API/Application. You should ask me to add a new scope for your Application (many already exists, `orbital`, `ao`, `cognitive`, `sse`, etc...)

From there you can use the notion of *sub-scopes* to associate different permission to different roles. Here is an SXO example:

```
[[:scope        "admin" "user" "sat" ]
 ["ao"          :rw     :r_    :r_   ]
 ["ao/execute"  :rw     :rw    :__   ]]
```

This mean:

- `admin` will have the full root scope `ao` granting everything
- `user` will have only read-only scope for `ao` (denoted `ao:read`) providing read-only access to SXO but will also have the sub-scope `ao/execute`.
- `sat` will only have read-only for `ao`.

So SXO team can, just by looking at the scopes (and not the role anymore) decide what permissions a token can provide.

This is also very important to use scopes only for permissions because this is the only single way provided by the OAuth2 RFC to limit permissions to OAuth2 clients. So even though an admin as full SXO access, the same admin might not want to provide this full access to a 3rd party that uses an OAuth2 client.

Post Q4

Future; potentially, custom roles, which will mean that the list of role will never be fixed and the relation between a role and a set of permission could be dynamically changed.

So we need to only care about the permissions (scopes) and not roles that will be random ids.

OAuth2 Clients Tokens without any scope

Any admin can create an OAuth2 Client Credential client (via the UI). If this admin does not select any scope the JWT will contains the following properties:

```
role: "admin"
scopes: []
```

So according to the role they could do everything. But according to the allowed permission they should not be allowed to perform most operation.

Regarding the UI to create such Client it is expected to not be allowed to perform admin operations at all.