API Client Authentication via OAuth

Original Content Provided by Jim Fox

Some caveats:

1) We can separate the part of the Enitity Registry that handles credentialing into an Entity Registry Token Service (TS).

2) I think entities in the registry must have 'friendly' ids, in addition to the uuid ids. Practice suggests this is useful.

kh: Yes, it occurred to me that UUIDs are not the friendliest identifier for people to work with.

3) Entity passwords could be maintained in Kerberos or LDAP, the same way user passwords are handled by the institution.

4) Long lasting passwords, of the type in '3', are only for authentication to the TS.

5) This part is about identity, not about authorization.

kh: Agreed.

Background:

Read and understand:

- RFC6749, *The OAuth 2.0 Authorization Framework*
- RFC7521, *Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants*
- RFC7523, *JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants*
- (and RFC7522 if you must), *Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants*

1) We are interested in the client identification pieces of the RFCs.

2) Current RFCs address bearer tokens. Future ones will likely address HoK (holder of key) ones. Nothing here should suggest we cannot migrate to HoK tokens in the future.

kh: I think that's a really important point. We should do our best to keep a migration path open. There are proposals from Justin Richer and others that illustrate one way to do HoK, but that might not be the exact path the as-yet-unpublished specification ends up taking

Our Term	Definition
Client	An app that wants some resource from the Service.
Service	An app that has some resource wanted by the Client. I consider 'authentication service' to be an endpoint on the Service, which is the way the world works. If you insist on a separate authentication service, well then do that.
Token Service	Our Entity Registry's Token Service (TS)

Authentication flow (from 7521 and 7523):

1) Client request access token from the TS, authenticating with a long-term password or a client certificate.

```
GET (TS)/token?service=(id of the service)
```

2) TS responds with an assertion_type and an assertion:

```
type: urn:ietf:params:oauth:grant-type:jwt-bearer
 assertion: a JWT containing:
 "iss": issuer, our TS,
 "sub": the Client id,
 "aud": the Service id.
```

3) Client presents this to the Service's *authorization* endpoint for an access token.

```
POST (Service)/access

grant_type=client_credential
 client_assertion_type=urn:ietf:params:oauth:grant-type:jwt-bearer
 client_assertion=(the JWT assertion from the TS)
```

4) From the assertion the Service knows the Client and replies with an access token, which the Client can use for further access to the Service's resources.

It is simpler than we might imagine.

– Jim Fox

OAuth for use cases that involve the client, the subject using the client and the service being invoked

OAuth Authorization Code Grant with JWT authorization and JWT identity assertion

This is the situation where there is a user who is the resource owner. The flow is an OAuth2 Authorization Code Grant as extended by 7521 and 7523.

Postulate: Suppose a department at the UW wasn't so happy with our Group Service (GS) UI. They might provide a member manager application (MM) to present their users with something friendlier to their users.

In OAuth terms:

"User": The resource owner. In this case an owner of groups.

"MM": The client application. ("Member Manager")

"GS": The resource server ("Group Service")

"GS-AS": The authorization service aspect of the resource server ("Group Service - Authorization Service")

Authorization flow:

0) The MM has already acquired an identity JWT from TS to GS-AS. (API Client Authentication via OAuth)

1) User browses to MM.

2) MM uses Shib, so it knows the User. MM presents a list of the User's groups the MM is willing to manage. How the list was acquired is not in scope at present.

3) User selects groups to manage.

4) MM redirects User to GS-AS with:

```
response_type: urn:ietf:params:oauth:grant-type:jwt-bearer
client_id:  MM's uuid
scope: list of groups to member manage
redirect_uri: MM's return url
```

5) GS-AS asks User, "Application MM wants permission to manage the memberships of ..., ..., and ...? (Including what will be done with which groups.)"

6) User says OK or selects some of the options.

7) User redirected to MM with authorization assertion

```
"assertion": the authorization assertion JWT
{"iss": GS-AS,
   "sub":"user_id@washington.edu",
   "aud": GS,
   "nbf": now,
   "exp": some-time,
   claims about authorizations on the groups}
```

8) MM POSTs to GS-AS token endpoint with

```
grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer
```

```
assertion=the assertion JWT
client_assertion_type=urn:ietf:params:oauth:grant-type:jwt-bearer
client_assertion=(the assertion from the TS)
(other_optional_arguments)
```

9) GS responds with an access token.

```
"access_token": some opaque string
"token_type": "Bearer",
"expires_in": some_seconds,
"refresh_token: some opaque string
```

10) MM uses access token to authenticate when managing group memberships for the user.

Notes

1) I did some interpretation of 7521 and 7523. We might have to examine some existing code to see what people realy do.

2) Bearer tokens have the advantage that they can be easily used with GET.

3) If we omitted the JWT this is simply RFC6749 OAuth with a JWT client credential.

– Jim Fox