Client / Service API Authentication

(Rough Starter Draft)

Terminology

tl;dr: Short term x509 certificates provide a mature and universally available means for holder-of-key authentication for TEIR API client and server interactions.

Terms:

- "Client": An app that wants some resource from the Service.
- "Service": An app that has some resource wanted by the Client.
- "Authorization Service: An endpoint on the Service providing OAuth access tokens.
- "Token Service": The Entity Registry's Token Service (TS). (aka, the CA)

Note: By issuing only short lived certificates, about the lifetime of an OAuth access token, the certificate authority (TS) does not need to provide any CRL information, nor would it need to maintain a consistent casn---except for logging. It is a very simple, database-less CA. Essentially it is a signing service

Client-Service Registry

I made a demo of a client-service registry. A few notes before we get to it.

The registry is a means by which API clients and services can know and identify one another---through a chain of trust architecture.

The registry is not OAuth. Although, considering that the OAuth authorization server is just another service,

The registry would support the registration of client to authorization servers. (Chapter 12 of the 'Book', but with a trust model.)

There's more to building a demo than meets the eye. And I'm new to much of the technology. Some of the details of the resources and API's are a bit haphazard.

There are only GET methods at present.

demo url: https://urizen.s.uw.edu/service_registry/

id: nobody

pw: asdfg12345

You can browse the site with a browser or with curl. The browser will return pretty printed HTML.

For an API view, something like this

```
$ curl -o - -X GET -H "Accept: application/json" -u nobody:asdfg12345 https://urizen.s.uw.edu/service_registry/Client/
```

would get you a list of clients.


```
- response: [
{"id":1,"entity_id":"csr_client1","title":"Demo client 1"},
{"id":2,"entity_id":"csr_client2","title":"Second demo client"}
]
```

There is an auto-swagger endpoint: https://urizen.s.uw.edu/service_registry/swagger/

It doesn't have any details of the returned resources. Don't know why.

Issuing credentials

There is also a Token endpoint, by which a client can get authn credentials to a service. It works like this.

1) Client (csr_client1) POSTs to the Token endpoint for a particular service (csr_service1).

```
$ curl -o - -X POST --data '' -H 'Content-type: application/json' \
-u (client's credentials) https://urizen.s.uw.edu/service_registry/Token/csr_service1/?type=Basic
```

and get's back

```
{"tk_token":"9e04b8f8-0218-4574-a09a-6cd696669447","tk_expires":"2017-04-04T23:25:10.463971"}
```

2) It then contacts the service, using it's own entity_id and that token as password.

3) The service contacts the registry to verify the password. e.g.:

```
$ curl -o - -X GET -H "Accept: application/json"
-u (serivce's credentials) https://urizen.s.uw.edu/service_registry/Client/csr_client1/?token=9e04b8f8-0218-4574-a09a-6cd696669447
```

If the password is valid the client representation is returned, else 404. Note the token is for that server only.

There are other possible Token types that would obviate the verification call to the registry.

JWT, cert come to mind. As does this interesting signed message:

https://datatracker.ietf.org/doc/html/draft-cavage-http-signatures

I don't know the efficacy of live authentication demos, but this does get across what I think is the utility of the registry.

Jim