NRC·CNRC

Authentication in Web Services and TAP-1.1 specific issues

Patrick Dowler
Canadian Astronomy Data Centre

Web Service A&A @ CADC

- 25 RESTful web services in operation (CADC + CANFAR)
  - 16 are IVOA standard services
  - 9 are custom services
- all of these use VOSI-capabilities
- all of these have at least one capability which describes authenticated access (~45 capabilities)

- clients consult a runtime-registry to find the capabilities
  - optimised for resourceID → capabilities URL
- clients read the capabilities document and look for the combination of {standardID,securityMethod} that match:
  - the feature they want to invoke
  - the credentials they want to use to authenticate
- @CADC: {resourceID,standardID,securityMethod} → {accessURL} happens several times per request & millions of times per day

Why is it so hard to add A&A to web services?

- allow use of a variety of technologies: legacy, current, shiny
- allow flexibility in web service implementation and deployment
  - sometimes constrained by other API rules
- services have to describe what authentication methods they support (IVOA registry, VOSI-capabiities)
- clients have to make use of the (self) description to be able to use services

- balance -- can't make any of these too onerous or restrictive

VOSI-capabilities 101

- a web service endpoint for a self-describing service
  - e.g. http://example.net/service/capabilities
  - (contains 1+) capability standardID: what feature is this?
  - (contains 1+) interface: a single callable endpoint
  - contains 1 accessURL
  - contains 0+ securityMethod*

```
<capability standardID="ivo://ivoa.net/std/FOO#feature”>
   <interface xsi:type="vs:ParamHTTP" role="std" version="1.0">
     <accessURL use="full"> https://www.example.net/impl/foo </accessURL>
     <securityMethod standardID="ivo://ivoa.net/sso#tls-with-certificate" />
   </interface>
</capability>
```

Test Particle: TAP and Authentication

- VOSI-capabilities / VOResource model is that a capability is a single feature
- In TAP-1.0, we specified relative names for the endpoints:
  - /availability
  - /capabilities
  - /tables
  - /async
  - /sync
- BUT we specified one standardID for the base URL
  - clients have to append the specified names
  - auth methods that use alternate path names not feasible
- TAP-1.0 doesn't play nice with all securityMethod(s)
- TAP-1.1 must support authentication and must provide a good backwards-compatible experience for older client s/w

TAP and Authentication

- prototype #1: one capability for each securityMethod

- pros:
  - none
- cons:
  - naive client that assumed one anonymous capability per standardID would fail or depend on ordering
  - lots of redundancy in VOSI-capabilities documents
  - inside-out with respect to the VOResource model where securityMethod is at the leaf
  - makes an assumption about what multiple capability(s) with the same standardID means …

TAP and Authentication

- prototype #2: separate standardID for sync and async
  ivo://ivoa.net/std/TAP#sync-1.1
  ivo://ivoa.net/std/TAP#async-1.1
  SODA-1.0 defines #sync-1.0 and #async-1.0
  VOSpace-2.1 defines #transfers and #sync-2.1

- pros:
  - did not break any old clients (we had this in operational use for years)
  - matches design of VOResource
  - backwards compatible records simple
  - allows for different TAPRegExt metadata (e.g. optional features, limits) in sync and async
- cons:
  - duplicates TAPRegExt info in sync and async
  - makes example RegTAP queries return different (more) results

TAP and Authentication

- prototype #3: separate interface type for sync and async
- lookup becomes:

{resourceID,standardID,interfaceType,securityMethod} → accessURL

- pros:
  - does not break any old clients (in operational use for a few months)
  - backwards compatible records possible
- cons:
  - backwards compatible records are subtle
  - set of interface(s) mixes base (client appends resource name) and full (accessURL includes resource name)
  - makes example RegTAP queries return different (more) results that users have to grok

TAP and Authentication

- approach #1: it's horrible and it breaks stuff
- approach #3 works, BUT: introduces subtle use of interface types and mixed interface style in a single capability
  - ruled out at College Park Interop (Nov 2018)
- approach #2: separate #sync-1.1 and #async-1.1
  - matches the VOResource/VOSI-capabilities design
  - works the same way as all other IVOA services
  - agreed to go back to this at College Park Interop (Nov 2018), dissenters remain
- So…. Now what?

One URL to rule them all

- We have had people (CADC resident astronomers) ask questions like:

*What is **the** URL to download this file?*

*What is **the** URL for this service?*

One URL to rule them all

- We have had people (CADC resident astronomers) ask questions like:

*What is **the** URL to download this file?*

*What is **the** URL for this service?*

- new goal: achieve that in deployment and then it would by nature be easy to describe

```
<capability standardID="ivo://ivoa.net/std/TAP">
    <interface xsi:type="vs:???" role="std" version="1.1">
      <accessURL use="full"> https://www.example.net/tap </accessURL>
      <securityMethod standardID="ivo://ivoa.net/sso#anon" />
      <securityMethod standardID="ivo://ivoa.net/sso#tls-with-certificate" />
      <securityMethod standardID="ivo://ivoa.net/sso#cookie" />
      <securityMethod standardID="ivo://ivoa.net/sso#OAuth" />
     </interface>
</capability>
```

One URL to rule them all

- we have one prototype service where one accessURL works for:
  - anonymous
  - #tls-with-client-cert
  - #cookie
  - and should work with other token systems

  - BUT: separate URL for #BasicAA because that URL behaves differently (use of HTTP status codes to trigger client to retry with auth)
  - everything is on https (OK)
  - could simplify VOSI-capabilities if multiple securityMethod(s) OK again (SSO-2.0 says something, VOResource plans... TBD)

One URL to rule them all

- my thoughts, in no particular order:
  - VOSI-capabilities are for users with a client tool
  - the people who really want to use #BasicAA are using curl/wget and not reading the capabilities anyway
  - use one interface/accessURL per capability
  - I would not “register” the #BasicAA endpoints; just local docs

  - simple anon service: OK
  - simple service with only #BasicAA: OK
  - service with multiple authentication methods: OK but cannot include #BasicAA
  - clients can still do username/password auth but it would be implemented as ***call this related service and get a cookie or token*** (this is how the astroquery cadc and esac TAP clients work) -- interoperable? TBD

Final Thoughts

- One URL to rule them all
  - auth in web services to be simple
  - some technologies/combinations not supported
  - restricts deployment
  - answers that question from astronomers
  - do we need an #anon securityMethod or just assume/try?
  - looks like this works; haven't gotten stuck yet

- One standardID per feature aka everything is a capability
  - use all the 1-n relations in capabilities (VOResource et al)
  - all technologies/combinations supported
  - allows more flexibility in deployment
  - more complex for clients
  - proven to work