Challenges on implementing token-based A&A

Sonia Zorba - INAF

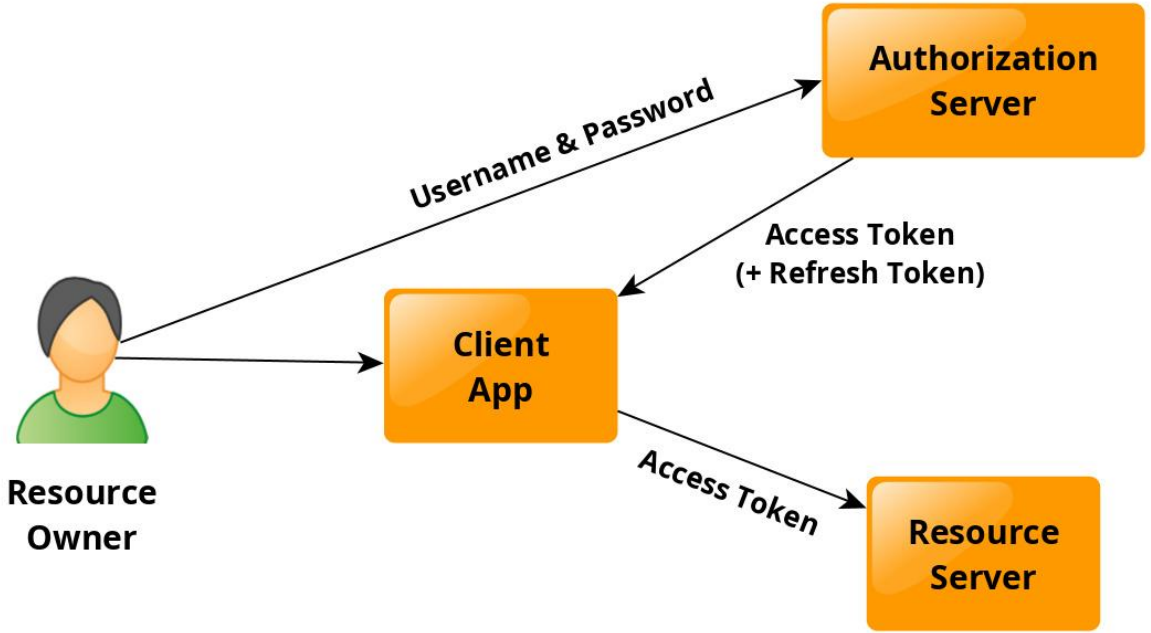
IVOA October 2019 Groningen Interoperability Meeting

Overview

- Overview of OAuth2, OpenID Connect and JWT
- IA2 use case
- Non-browser access with OAuth2 (discussed at ADASS BoF on Science Platforms)
- Key distribution using JWKS
- Credential delegation with tokens

OAuth2

It’s an authorization protocol. A simplified view:



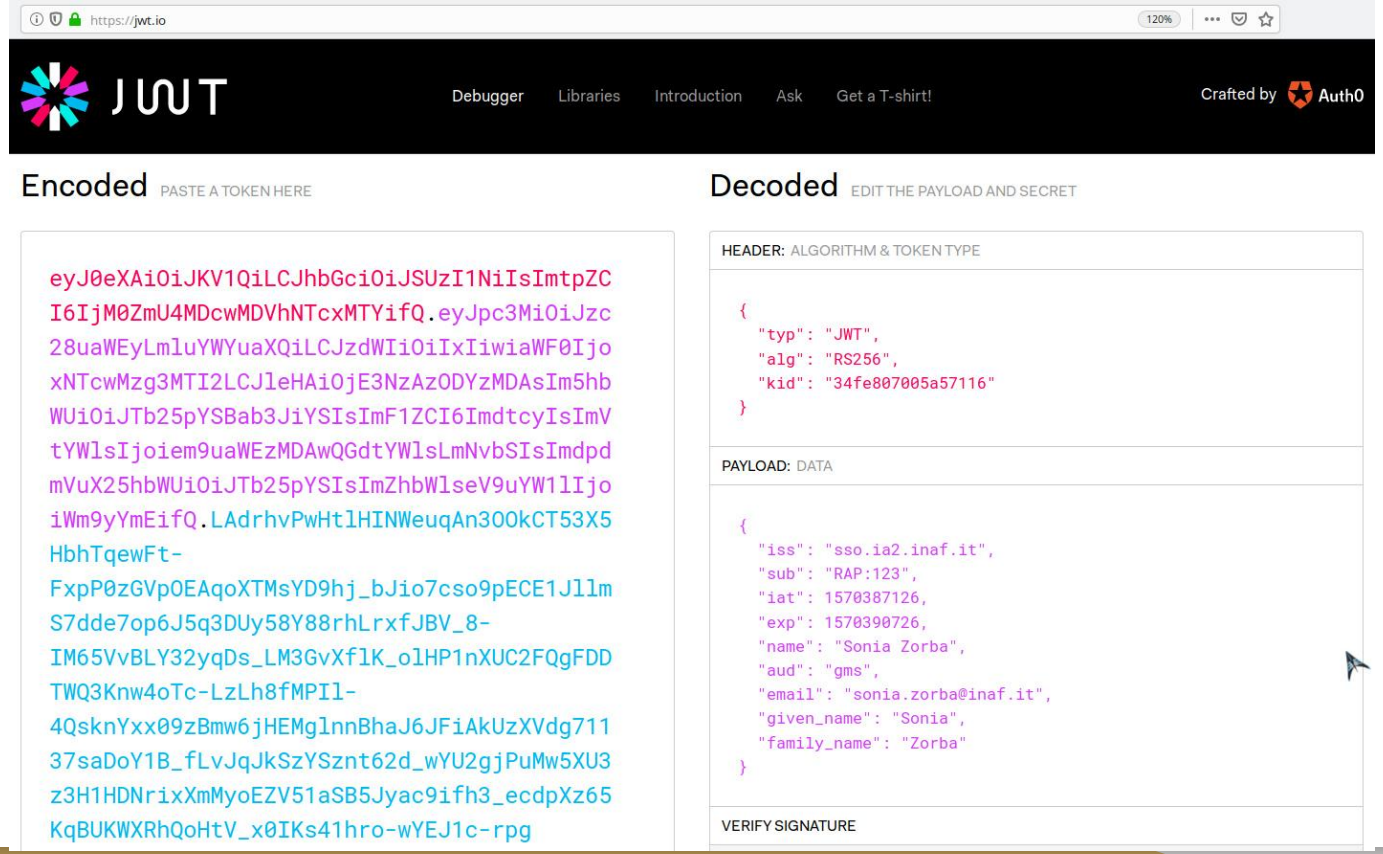
Tokens

2 categories:

- reference tokens
  - they are just a handle
  - an additional call is needed to validate the token and retrieve its content (e.g. /auth/check_token)

- self-contained tokens (e.g. JWT)
  - all the information is stored inside the token
  - can be validated by the receiver because they are signed or encrypted

JWT


https://jwt.io
120%
JWT
Debugger
Libraries
Introduction
Ask
Get a T-shirt!
Crafted by Auth0
Encoded PASTE A TOKEN HERE
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IjM0ZmU4MDcwMDVhNTcxMTYifQ.eyJpc3MiOiJzc28uaWEyLmluYWYuaXQiLCJzdWIiOiIxIiwiaWF0IjoxNTcwMzg3MTI2LCJleHAiOjE3NzAzODYzMDAsIm5hbWUiOiJTb25pYSBab3JiYSIsImF1ZCI6ImdtcyIsImVtYWlsIjoiem9uaWEzMDAwQGdtYWlsLmNvbSIsImdpdmVuX25hbWUiOiJTb25pYSIsImZhbWlseV9uYW1lIjoiWm9yYmEifQ.LAdrhvPwHtlHINWeuqAn3OOkCT53X5HbhTqewFt-FxpP0zGVpOEAqoXTMsYD9hj_bJio7cso9pECE1JllmS7dde7op6J5q3DUy58Y88rhLrxfJBV_8-IM65VvBLY32yqDs_LM3GvXflK_olHP1nXUC2FQgFDDTWQ3Knw4oTc-LzLh8fMPIl-4QsknYxx09zBmw6jHEMglnnBhaJ6JFiAkUzXVdg71137saDoY1B_fLvJqJkSzYSznt62d_wYU2gjPuMw5XU3z3H1HDNrixXmMyoEZV51aSB5Jyac9ifh3_ecdpXz65KqBUKWXRhQoHtV_x0IKs41hro-wYEJ1c-rpg
Decoded EDIT THE PAYLOAD AND SECRET
HEADER: ALGORITHM & TOKEN TYPE
{
"typ": "JWT",
"alg": "RS256",
"kid": "34fe807005a57116"
}
PAYLOAD: DATA
{
"iss": "sso.ia2.inaf.it",
"sub": "RAP:123",
"iat": 1570387126,
"exp": 1570390726,
"name": "Sonia Zorba",
"aud": "gms",
"email": "sonia.zorba@inaf.it",
"given_name": "Sonia",
"family_name": "Zorba"
}
VERIFY SIGNATURE

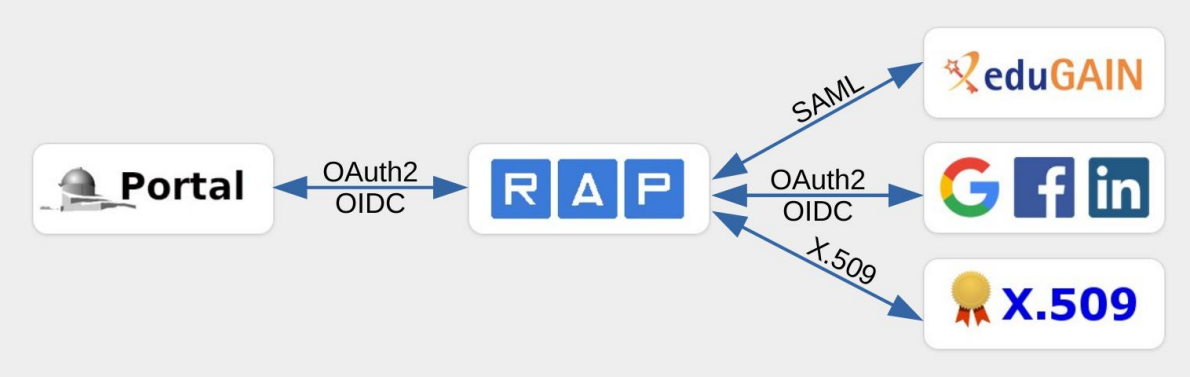
OpenID Connect (OIDC)

OpenID Connect implements authentication as **an extension to the OAuth 2.0** authorization process.

An id_token, a JWT containing identity information, is returned together with the access_token and refresh_token in a OAuth2 response.


```
{
  "access_token":
    "BjYzcyNmEzMmI3YzA2MGY2MmEyNDMzNzRlYWYzM2VlNzg3ZDM3
    ZGE3OTIxYjI0OGEwZWNhMTY4ZWI5YWZkOWI2OWY3NTIzNjZlNDV
    jMDRkYThmYmE3ZGFlN2==",
  "refresh_token":
    "NmI3YjE1MGI1NjQxZWRjNjU2YjYwOGY0OWRmYzk3MGM1NWIyOG
    M1Zjk0N2I5ZWE1MTE3YVlNDk2MTIzYjhhMDFjNzEwZDEyYTM5YW
    QyMmQ5ZDIzMRlMzc1MQ==",
  "token_type": "Bearer",
  "expires_in": 3600,
  "id_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
    .eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIi
    wiaWF0IjoxNTE2MjM5MDIyfQ
    .SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c"
}
```

Tokens in IA2 portals



- Different protocols translated to OAuth2/OIDC
- Tokens used for communications between our services

	file_name	policy
☐	HARPN.2019-09-09T00-03-59.378.fits.gz	PRIV
☐	GIANO-B.2019-09-09T00-01-52.000.fits.gz	PRIV
☐	GIANO-B.2019-09-09T00-06-34.000.fits.gz	PRIV
☐	GIANO-B.2019-09-09T00-11-16.000.fits.gz	PRIV
☐	GIANO-B.2019-09-09T00-15-58.000.fits.gz	PRIV
☐	GIANO-B.2019-09-09T00-20-41.000.fits.gz	PRIV
☐	HARPN.2019-09-09T00-19-28.650.fits.gz	PRIV

← Portal is built over a TAP service

Command line access to private data

- **Basic-Auth**

```
wget --http-user=<username> --http-password=<password> -i files_list.txt
```

- **X.509**

```
wget --certificate=<certificate-file> -i files_list.txt
```

- **OAuth2?**
  - **Step 1:** obtain a token **[...]**
  - **Step 2:** use it:

```
wget --header="Authorization: Bearer <token>" -i files_list.txt
```

  - **Step 3:** refresh the token?

RFC 8252 - OAuth 2.0 for Native Apps

- Main issue: OAuth2 protocol is based on **redirects**, so a browser is needed in order to insert the credentials and go back to the caller application.

- Solution: redirect to a special URI:
  - listen on a port (127.0.0.1:<port>)
  - register private-use URI schemes (com.example.app://something)
  - claimed https URI

+ Proof Key for Code Exchange (PKCE)

- + keep in mind that tokens expire

It is necessary to build complex clients!

Key distribution using JWKS endpoint

- JWT are used for communication between services
- When JWT are signed/encrypted with asymmetric cryptography (e.g. RSA) they are good for scalable infrastructures
- Public keys have to be distributed between nodes
- Keys must be renewed after some time (key rotation)

RFC 7517 (JSON Web Key) defines **JWKS**:

```
{
  "keys": [
    {
      "kty": "RSA",
      "kid": "32dc45a9102504cf",
      "use": "sig",
      "n": "yyQk0-yE5hXvveSV23zAnnhkrrLMO2wZFTcWte
        -gQhkwZVm1en4VHkfOuCfIjb2QMKujyUHQkWdRRCwyaJEdx
        OL-kI8SSLibqYpyj5lPyXY3RfP6D_2A0QTbHp_XUJmwzl3h
        AwANsMGlaZ8n1A7gof_5-dqs1ZIJH-B9sA8LrYc=",
      "e": "AQAB"
    },
```

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "32dc45a9102504cf"
}
```

PAYLOAD: DATA

```
{
```

Credential delegation with tokens

Simpler case: token relay (service B forwards received bearer token to service C)

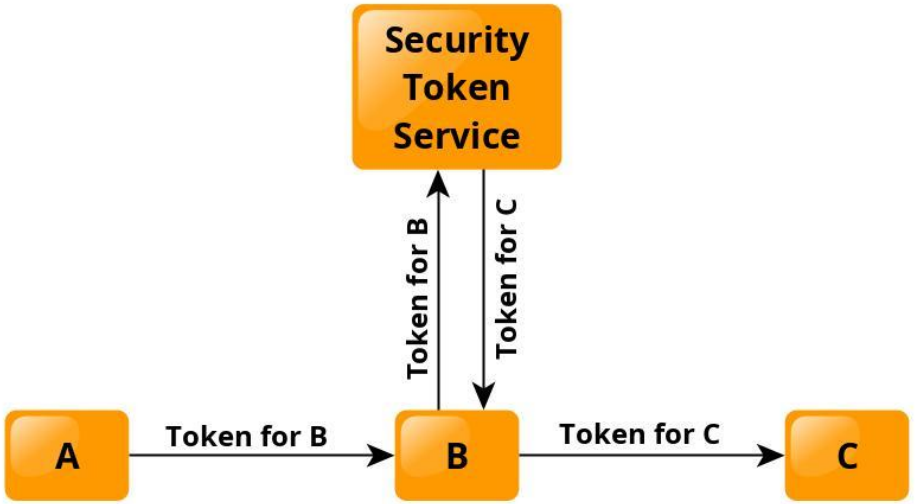
Not always possible:

- some systems require the usage of the “aud” (audit) claim: the name of the recipients that the JWT is intended for (the target service)
- some systems use the “jti” (JWT ID) claim to prevent token reuse

Credential delegation with tokens

Using OAuth 2.0 Token Exchange (Internet Draft)

```
{
  "sub": "bob",
  "client_id": "A",
  "aud": "B"
}
```




```
{
  "sub": "bob",
  "client_id": "B",
  "aud": "C",
  "act": {
    "client_id": "A"
  }
}
```

Final thoughts

OAuth2/OIDC (but also SAML) is difficult to implement for non-browser access.
Moreover the standard is quite new and evolving compared to X.509.

Involved VO standards:

- SSO Profile: Authentication Mechanisms
- Credential Delegation Protocol

Should VO client applications (e.g. TOPCAT) support OAuth2/OIDC authentication?