OpenID Connect Authentication in iRODS

Kyle Ferriter
kferriter@renci.org
Renaissance Computing Institute (RENCI)
UNC - Chapel Hill

Context and Use Case

NIH Data Commons

- Web application - CommonsShare.org
- Compute Platform and Abstraction layer - DC/OS + PIVOT
- Data Management and Abstraction layer - iRODS

Assumption:

- Almost all users of the Data Commons will have an existing OpenID Connect identity through: InCommon, CILogon, OrcID, Google, Globus, and/or other identity provider which can be converted into OIDC format.
- Frontend identity federation with these is straightforward in our web app (Django).
- No need to create a brand new username and password for users to authenticate to the data backend of the platform.

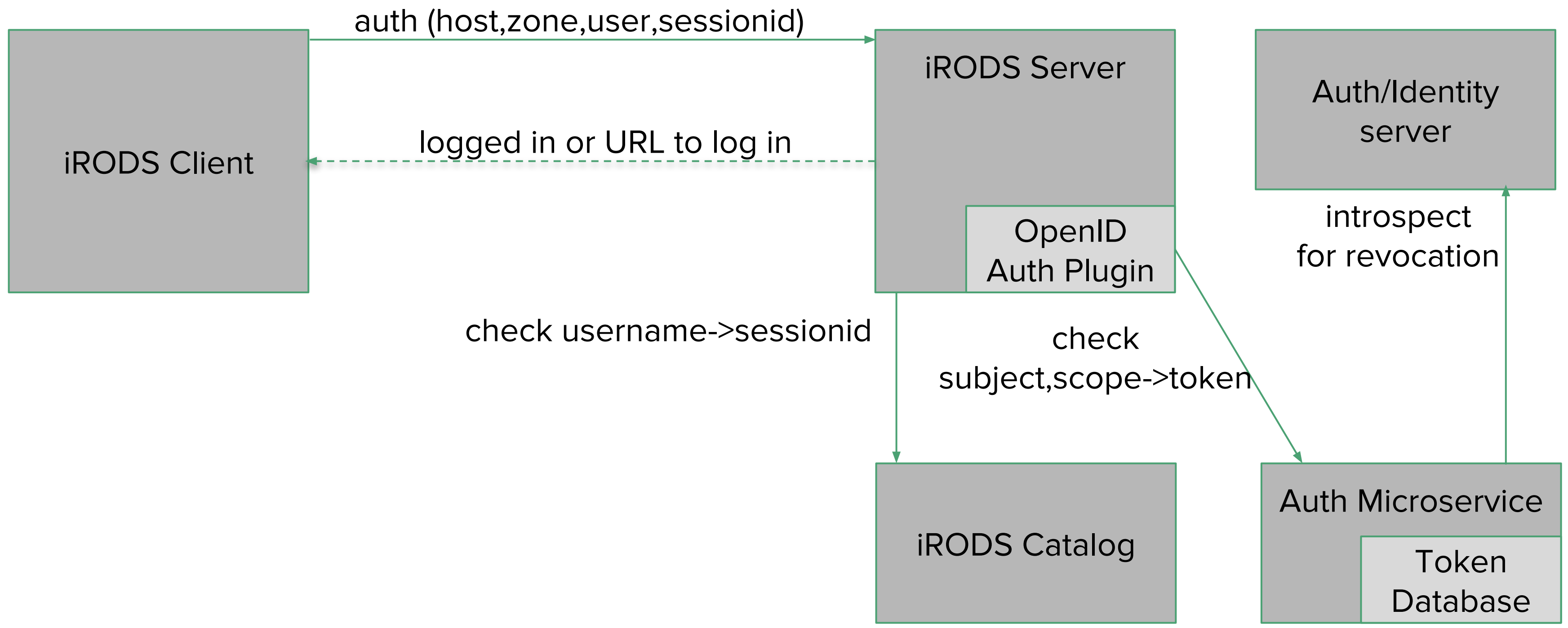
Goal

Provide an alternative authentication mechanism for iRODS users.

Allow users to link online OpenID Connect (OIDC) identities and OAuth 2.0 credentials from other platforms to an iRODS account.

Enable iRODS users to authorize iRODS plugins to access OAuth 2.0 protected APIs on their behalf (ex: personal file storage)

Architecture



auth (host,zone,user,sessionid)
iRODS Client
logged in or URL to log in
iRODS Server
OpenID Auth Plugin
Auth/Identity server
introspect for revocation
check username->sessionid
check subject,scope->token
iRODS Catalog
Auth Microservice
Token Database

Features

- Simultaneously support authentication through an arbitrary number of OIDC providers. Client application or admin can decide which each user can use.
    - Account linking must be done with admin privileges
- Multiple OIDC identities (different providers) linked to a single iRODS account.
    - Does not currently support linking multiple identities from the same OIDC provider.
- Client applications don’t need to store any passwords or raw access tokens.
- Can opaquely utilize Identity Provider functionality (Multi-factor Auth)
- If a user revokes their access via the identity/authorization provider, it immediately logs out all iRODS clients from their account**
    - 50-70% performance penalty
    - configurable cache makes this less noticeable

Auth Microservice

- Used to:
  - initiate an Authorization Code Flow (OAuth2+OIDC)
  - generate cryptographically secure and unique nonce and state values
  - provide simple REST API for token management between stack components
  - automatically refresh expired tokens
  - validate tokens
- Tokens issued by the user’s login are returned to the service and stored encrypted in a Postgresql database.
  - Database name, host, username, password, and encryption key are configurable

Microservice Configuration

Where ‘providers’ are defined. These provider keys can be used in the iRODS deployment.

Standards-based.

- OpenID Connect Core 1.0
- RFC 6749 (OAuth 2.0)
- RFC 7662 (OAuth 2.0 Token Introspection)
- RFC 7519 (JSON Web Token)*

```
{
 "redirect_uri": "https://my.auth.host/authcallback",
 "url_expiration_timeout": 60,
 "real_time_validate_default": true,
 "real_time_validate_cache_retention_timeout": 60,
 "providers": {
  "idp-1": {
   "standard": "OpenID Connect",
   "client_id": "3709214a0573d322faa0ed7dca19b460",
   "client_secret": "266d36328f3db29e5f29741092c05a39",
   "metadata_url": "https://idp-1.example.com/.well-known/openid-configuration"
  },
  "authorization-server-1": {
   "standard": "OAuth 2.0",
   "client_id": "aeaf3d5e287b72bf48cfc7f0dab9e230",
   "client_secret": "ad14671a007347c0655f02277ab90f26",
   "authorization_endpoint": "https://authorization-server-1.example.net/authorize",
   "token_endpoint": "https://authorization-server-1.example.net/token"
  }
 }
}
```

iRODS Configuration

server_config.json:

```
"plugin_configuration": {
    "authentication": {
        "openid": {
          "default_provider": "idp-1",
          "token_service": "https://my.host.example.org",
          "token_service_key": "abcd"
        }
    },
    ....
}
```

Security Considerations

- Microservice privileges.
- Microservice API DoS.
- AES-256 encryption at rest, keys can be loaded via environment.
- TLS 1.1, 1.2
- Respecting user consent and withdrawal.
- Server port exhaustion.
    - Possible mitigation: implicit grant flow
- Login URL timeouts.
    - Once a login url is created, user has N number of seconds to use it or it is no longer accepted.

Demo

Configuration

Use:

- iCommands (iinit, ils, iput)
- python-irodsclient (pyils.py)

Real-time consent validation

Additional Steps

- python-irodsclient
- BDBag python library
  - Fetching files from an iRODS server
- Davrods/WebDAV
  - Apache OpenID module for OpenID login to iRODS using Apache
- JWT/JWK authentication
  - Signed identity and privilege scopes issued to a client by a platform’s authorization server.
  - Information is in the token itself, not in an API endpoint requested using an opaque token.
- Dockerized OpenID authentication
- Globus resource plugin