Authentication in iRODS 4.3:
Investigating OAuth2 and OpenID Connect (OIDC)
Current State of Authentication in iRODS

Overview of OAuth 2.0 and Open ID Connect

OAuth 2.0 Flows

Demo Setup & Demo

Future Considerations
Current State of Authentication
Authentication in iRODS

- Plugins
  - Native
    - Username & Password
  - PAM
  - GSI
  - Kerberos
- OAuth in Plugins is awkward to use...
• Ease of Use in Any Language

• Improved OAuth integration
  ■ Possible support of multiple grant types
Overview of OAuth 2.0 and OpenID Connect
What is OAuth 2.0?

- Enables limited client access to an HTTP service
  - At choice of resource owner
- Allows for finer, revocable control of resource owner data
- Avoids sharing password credentials
- Authorization focused
What is OpenID Connect?

- Provides an identity layer
- Enables clients to verify End-User
  - Provides basic profile information
- Authentication Focused
How do they work together?

- OIDC is an identity layer on top of OAuth 2.0

- OIDC
  - Authentication

- OAuth
  - Authorization
OAuth 2.0 Flows
OAuth 2.0 Grant Flows

- Authorization Code Grant
- Implicit Grant
- Resource Owner Password Credential Grant ***
- Client Credentials Grant
Authorization Code Grant

Of particular use for Confidential Clients

Note: The lines illustrating steps (A), (B), and (C) are broken into two parts as they pass through the user-agent.
Implicit Grant

- Optimized for Public Clients
  - Well-known redirection URI
- Typically implemented in the Browser

Note: The lines illustrating steps (A) and (B) are broken into two parts as they pass through the user-agent.
Resource Owner Password Credentials Grant

- Requires high trust between Client & Resource Owner
- Ideally used when alternatives flows are not viable, or migrating authentication schemes

![Diagram of Resource Owner Password Credentials Grant](image)
Client Credentials Grant

- Only for use with confidential clients
- Used in prearranged agreements
Demo
Demo Setup

- iRODS + Connection Management PR
- HTTP API + Endpoints PR
- OpenID Provider (OP)
  - Keycloak
OpenID Provider Configuration

- Create a new scope
  - Add claims to ID token
- Custom attribute
  - Mapping to iRODS user (e.g., chuck)
HTTP API Configuration

```json
... "authentication": {
  ... "oidc": {
    "config_host": "127.0.0.1",
    "port": "8080",
    "uri": "/realms/test/.well-known/openid-configuration",
    "client_id": "my_client_id"
  },
  ... "oidc": {
    "config_host": "127.0.0.1",
    "port": "8080",
    "uri": "/realms/test/.well-known/openid-configuration",
    "client_id": "my_client_id"
  }
...}
```

Additional OIDC Configuration
HTTP API OAuth/OIDC Flow

API Consumer Sends Authorization

1 POST /irods-http/0.9.5/authenticate HTTP/1.1
2 Host: ...
3 User-Agent: ...
4 Accept: */*
5 Authorization: iRODS bV9jaHVjazpmZWVsc3NvZ29vZA==

HTTP API Forwards login to OP

1 POST /realms/test/protocol/openid-connect/token HTTP/1.1
2 Host: ...
3 User-Agent: ...
4 Accept: */*
5 Content-Length: 85
6 Content-Type: application/x-www-form-urlencoded
7 client_id=rods&grant_type=password&scope=openid&username=m_chuck&password=pass

OP Provides 'id_token'

1 {
2   ...
3   "id_token": "eyJhbGciOiJSUzI1Ni...",
4   ...
5 }
'id_token' claims
Logging in as [m_chuck] with a password of [feelssogood].

Base64 encoded as [bV9jaHVjazpmZWVsc3NvZ29vZA==].

Running the command [curl -s -X POST -H "Authorization: iRODS $user_and_pass" 127.0.0.1:9000/irods-http/0.9.5/authenticate -v].

* Trying 127.0.0.1:9000...
* Connected to 127.0.0.1 (127.0.0.1) port 9000 (#0)
> POST /irods-http/0.9.5/authenticate HTTP/1.1
> Host: 127.0.0.1:9000
> User-Agent: curl/8.1.1
> Accept: */*
> Authorization: iRODS bV9jaHVjazpmZWVsc3NvZ29vZA==
>
< HTTP/1.1 200 OK
< Server: Boost.Beast/322
< Content-Type: text/plain
< Content-Length: 36
<
{

* Connection #0 to host 127.0.0.1 left intact

Received the following token: [95d56783-1f0b-4e7b-8ece-598fcb37eea5].
Looking at the collection `/tempZone/home/chuck`.
Running the command 
```
curl -s -G -H "authorization: Bearer $token"
   "127.0.0.1:9000/irods-http/0.9.5/collections"
--data-urlencode "lpath=$collection"
```
Results:
```
{
  "inheritance_enabled" : false ,
  "irods_response" : {
    "error_code" : 0
  },
  "modified_at" : 1685554932,
  "permissions" : [ 
    { 
      "name" : "chuck",
      "perm" : "own",
      "type" : "rodsuser",
      "zone" : "tempZone"
    }
  ],
  "registered" : true ,
  "type" : "collection"
}
```
Challenges

- OAuth 2.0 & OpenID Connect Definitions
- Determining Mapping Method
- Programmatically Determining OIDC Endpoints
Future Considerations
Considerations

- Alternative mapping mechanism for OAuth users to iRODS
  - Using 'sub' OIDC attribute
- OAuth Credentials Handling
- Support More OpenID Features
  - OpenID Provider Issuer Discovery
  - Dynamic Client Registration
- Possible overlap between PAM Interactive auth plugin
Draft Specifications

- OAuth 2.0 Security Best Practices Draft (Work in Progress)
  - Resource Owner Password Credentials MUST NOT be used

- OAuth 2.1 Draft (Work in Progress)
  - Resource Owner Password Credentials Omitted
References

- OAuth 2.0
  - https://www.rfc-editor.org/rfc/rfc6749

- OpenID Connect Core
  - https://openid.net/specs/openid-connect-core-1_0.html

- OpenID Connect Client Discovery
  - http://openid.net/specs/openid-connect-discovery-1_0.html

- OAuth 2.1 Draft
  - https://www.ietf.org/archive/id/draft-ietf-oauth-v2-1-08.html

- OAuth 2.0 Security Best Current Practice Draft
Thank you!

https://github.com/irods/irods_client_http_api/pull/37