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

Martin Flores
Research Software Developer
iRODS Consortium

June 13-16, 2023
iRODS User Group Meeting 2023
Chapel Hill, NC

Overview

- 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...

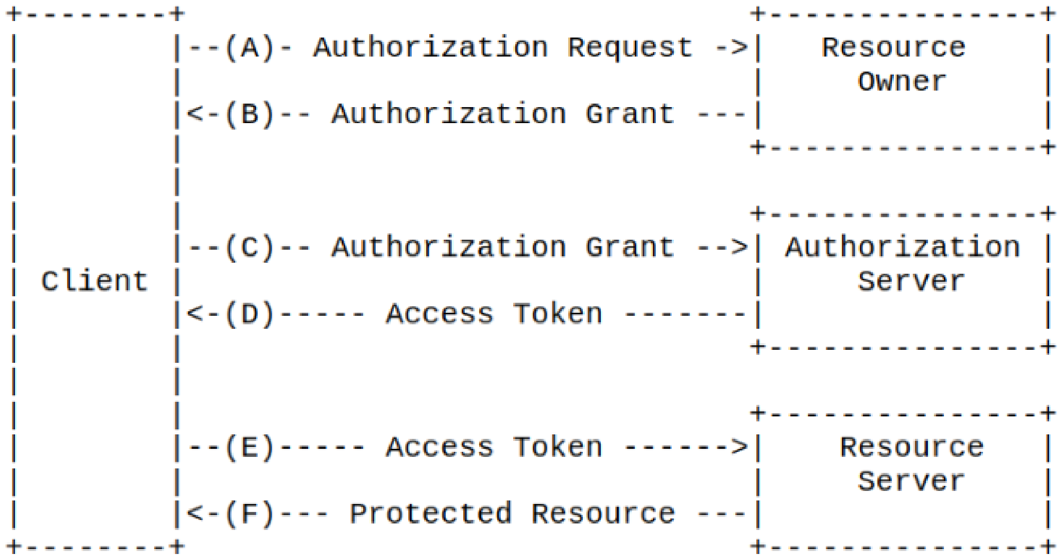
iRODS HTTP API

- 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?

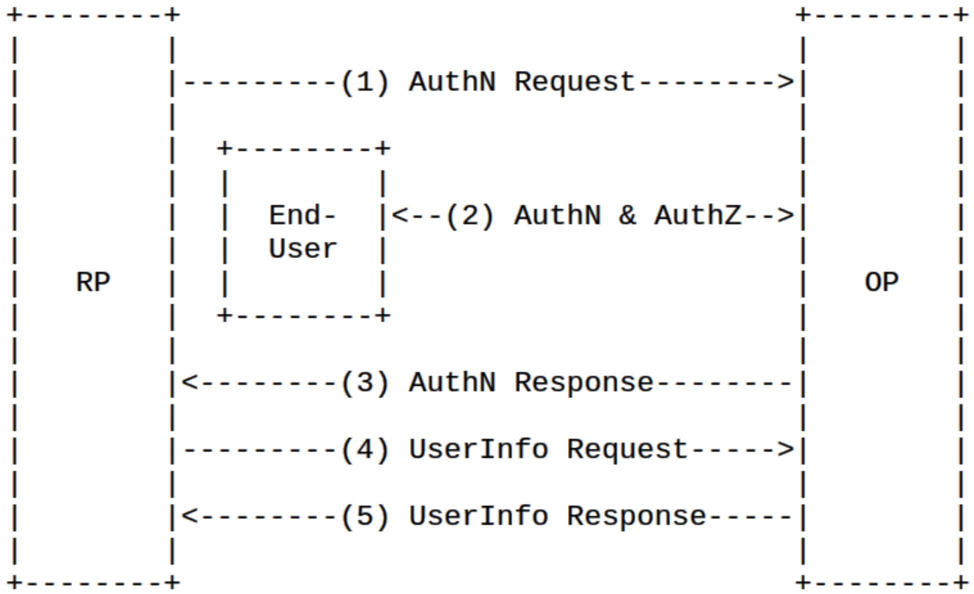
- Enables limited client access to an HTTP service
  - At choice of resource owner
- Allows for finer, revocable control of resource owner data
- Avoids sharing password credentials
- Authorization focused



```
+--------+                               +---------------+
|        |--(A)- Authorization Request ->|   Resource    |
|        |                               |     Owner     |
|        |<-(B)-- Authorization Grant ---|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(C)-- Authorization Grant -->| Authorization |
| Client |                               |     Server    |
|        |<-(D)----- Access Token -------|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(E)----- Access Token ------>|    Resource   |
|        |                               |     Server    |
|        |<-(F)--- Protected Resource ---|               |
+--------+                               +---------------+
```

What is OpenID Connect?

- Provides an identity layer
- Enables clients to verify End-User
    - Provides basic profile information
- Authentication Focused



```
+--------+                                   +--------+
|        |                                   |        |
|        |---------(1) AuthN Request-------->|        |
|        |                                   |        |
|        |  +--------+                       |        |
|        |  |        |                       |        |
|        |  |  End-  |<--(2) AuthN & AuthZ-->|        |
|        |  |  User  |                       |        |
|   RP   |  |        |                       |   OP   |
|        |  +--------+                       |        |
|        |                                   |        |
|        |<--------(3) AuthN Response--------|        |
|        |                                   |        |
|        |---------(4) UserInfo Request----->|        |
|        |                                   |        |
|        |<--------(5) UserInfo Response-----|        |
|        |                                   |        |
+--------+                                   +--------+
```

How do they work together?

- OIDC is an identity layer on top of OAuth 2.0

- OIDC
  - Authentication
- OAuth
  - Authorization

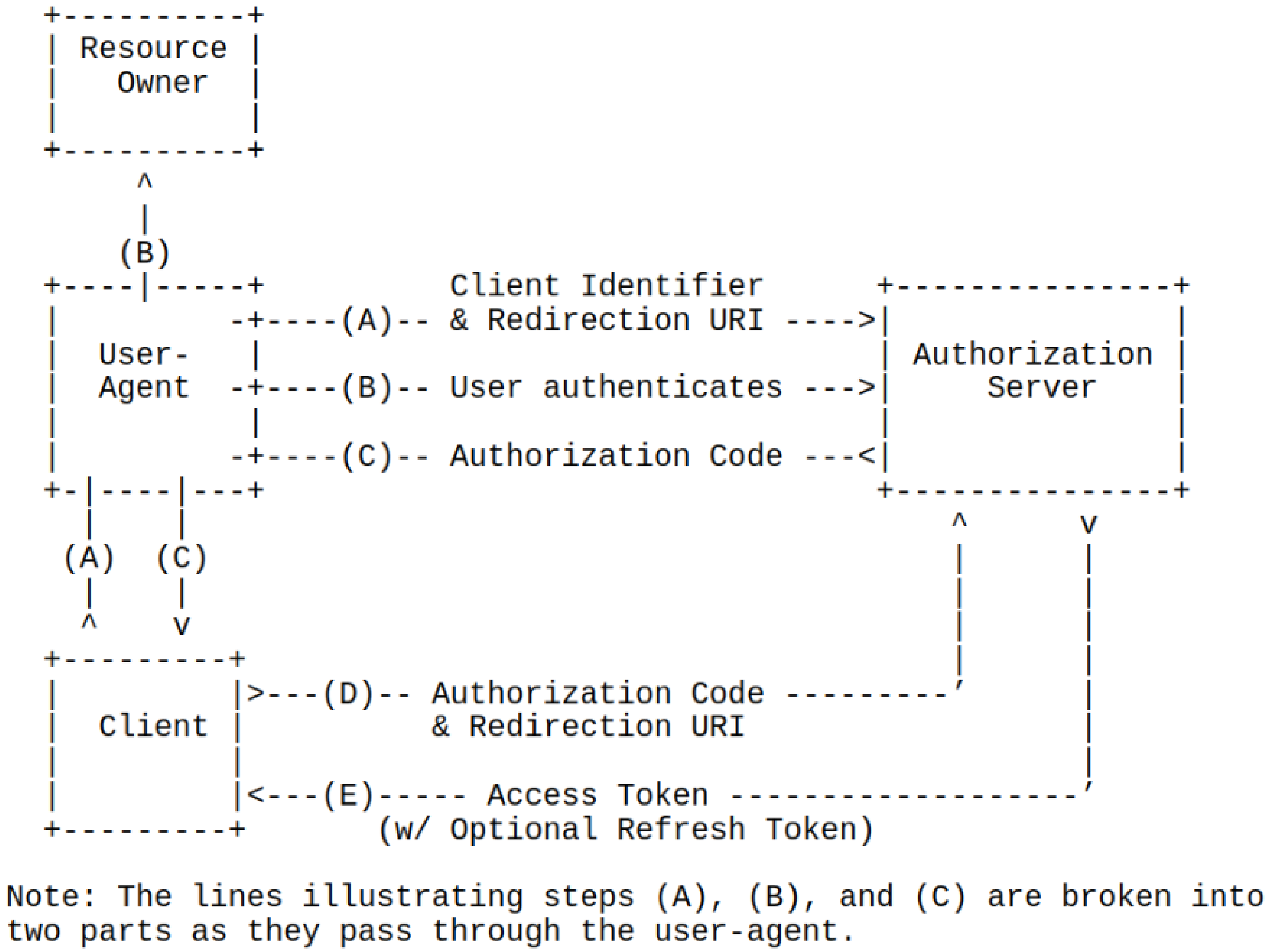
OAuth 2.0 Flows

OAuth 2.0 Grant Flows

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

Authorization Code Grant

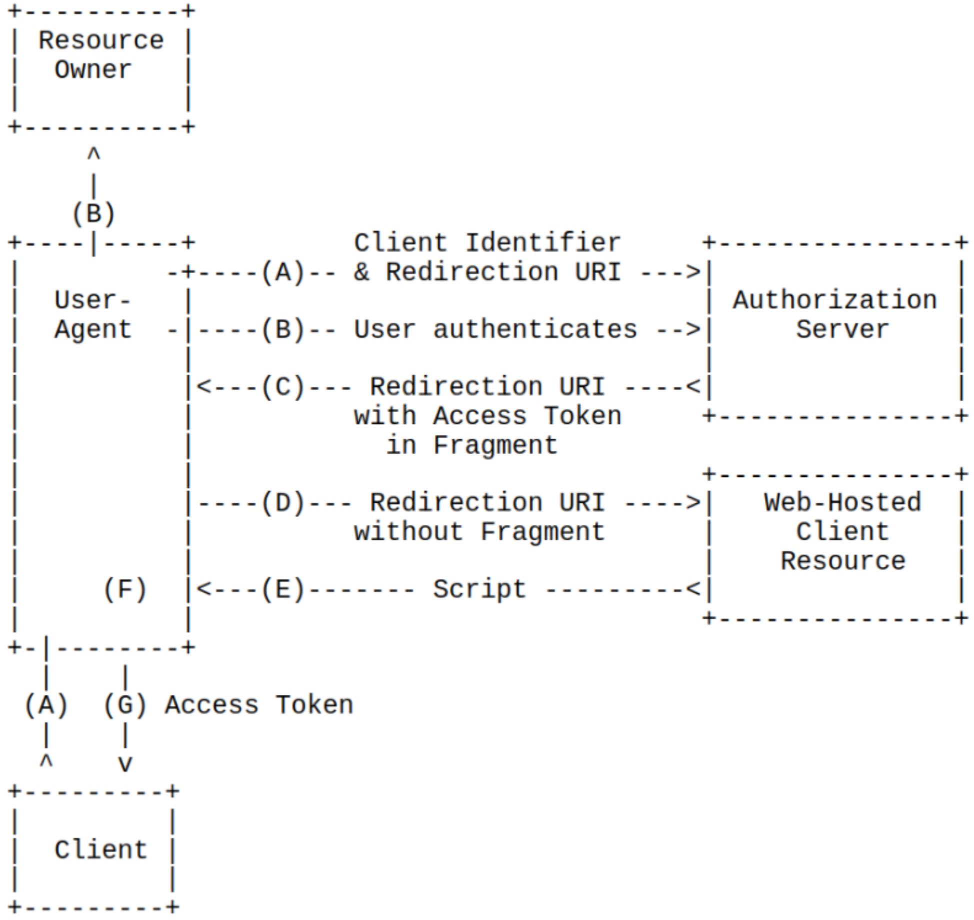
Of particular use for Confidential Clients



```
     +----------+
     | Resource |
     |   Owner  |
     |          |
     +----------+
          ^
          |
         (B)
     +----|-----+          Client Identifier      +---------------+
     |         -+----(A)-- & Redirection URI ---->|               |
     |  User-   |                                 | Authorization |
     |  Agent  -+----(B)-- User authenticates --->|     Server    |
     |          |                                 |               |
     |         -+----(C)-- Authorization Code ---<|               |
     +-|----|---+                                 +---------------+
       |    |                                         ^      v
      (A)  (C)                                        |      |
       |    |                                         |      |
       ^    v                                         |      |
     +---------+                                      |      |
     |         |>---(D)-- Authorization Code ---------'      |
     |  Client |          & Redirection URI                  |
     |         |                                             |
     |         |<---(E)----- Access Token -------------------'
     +---------+       (w/ Optional Refresh Token)

   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

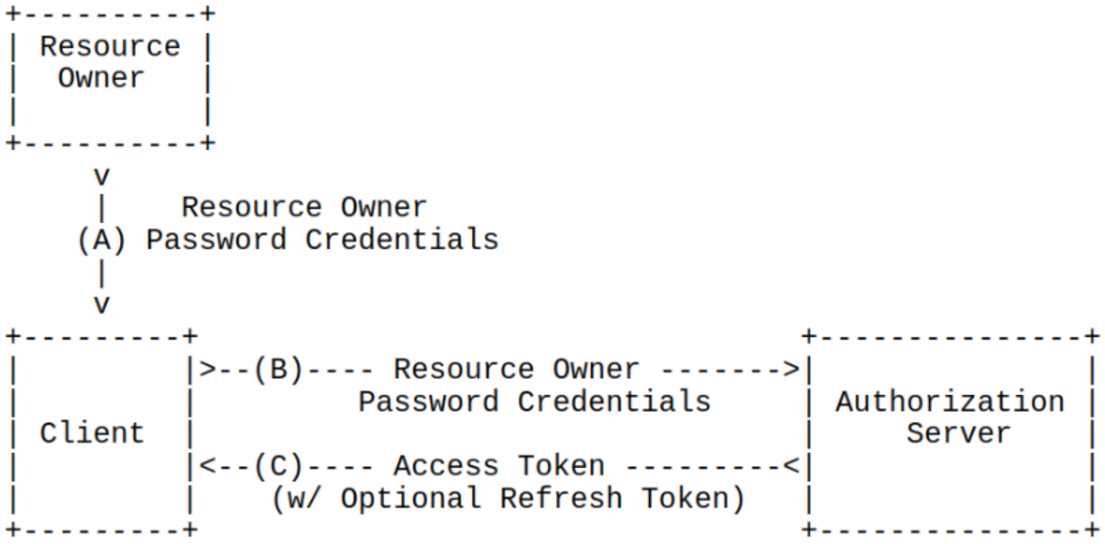


```
+----------+
| Resource |
|  Owner   |
|          |
+----------+
     ^
     |
    (B)
+----|-----+          Client Identifier     +---------------+
|         -+----(A)-- & Redirection URI --->|               |
|  User-   |                                | Authorization |
|  Agent  -|----(B)-- User authenticates -->|     Server    |
|          |                                |               |
|          |<---(C)--- Redirection URI ----<|               |
|          |          with Access Token     +---------------+
|          |            in Fragment
|          |                                +---------------+
|          |----(D)--- Redirection URI ---->|   Web-Hosted  |
|          |          without Fragment      |     Client    |
|          |                                |    Resource   |
|     (F)  |<---(E)------- Script ---------<|               |
|          |                                +---------------+
+-|--------+
  |    |
 (A)  (G) Access Token
  |    |
  ^    v
+---------+
|         |
|  Client |
|         |
+---------+
```

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

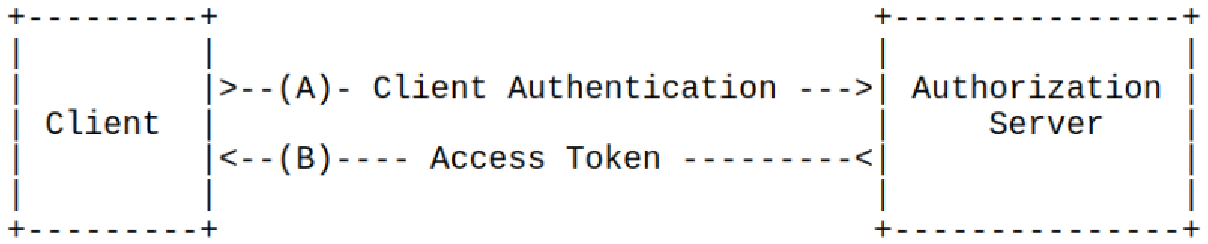
- Requires high trust between Client & Resource Owner
- Ideally used when alternatives flows are not viable, or migrating authentication schemes



```
+----------+
| Resource |
|  Owner   |
|          |
+----------+
     v
     |    Resource Owner
    (A) Password Credentials
     |
     v
+---------+                                  +---------------+
|         |>--(B)---- Resource Owner ------->|               |
|         |         Password Credentials     | Authorization |
| Client  |                                  |     Server    |
|         |<--(C)---- Access Token ---------<|               |
|         |    (w/ Optional Refresh Token)   |               |
+---------+                                  +---------------+
```

Client Credentials Grant

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



```
+---------+                                  +---------------+
|         |                                  |               |
|         |>--(A)- Client Authentication --->| Authorization |
| Client  |                                  |     Server    |
|         |<--(B)---- Access Token ---------<|               |
|         |                                  |               |
+---------+                                  +---------------+
```

Demo

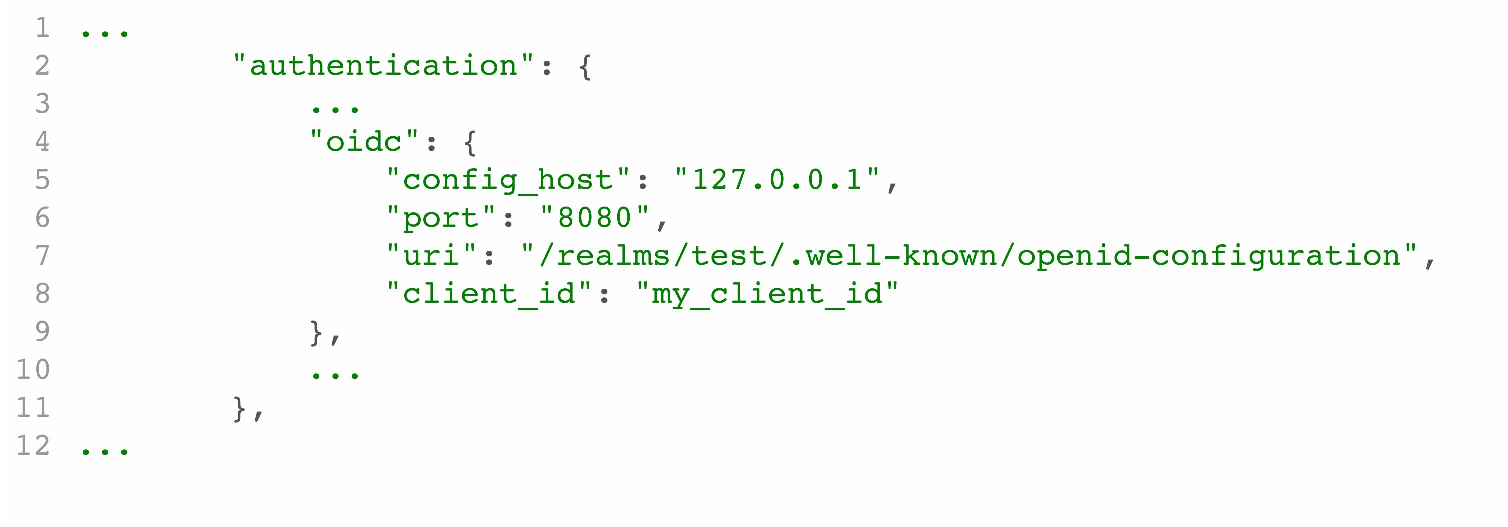
Demo Setup

- 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

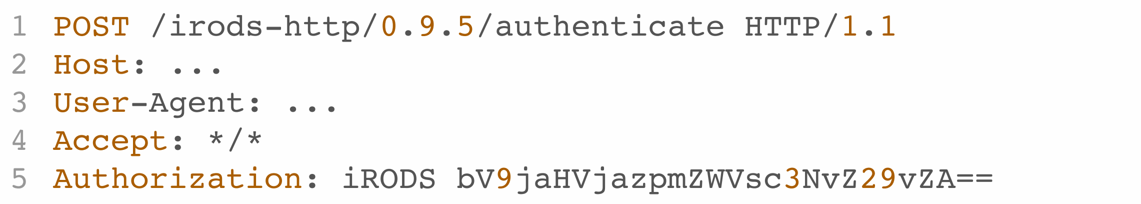


```
...
        "authentication": {
            ...
            "oidc": {
                "config_host": "127.0.0.1",
                "port": "8080",
                "uri": "/realms/test/.well-known/openid-configuration",
                "client_id": "my_client_id"
            },
            ...
   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

Additional OIDC Configuration

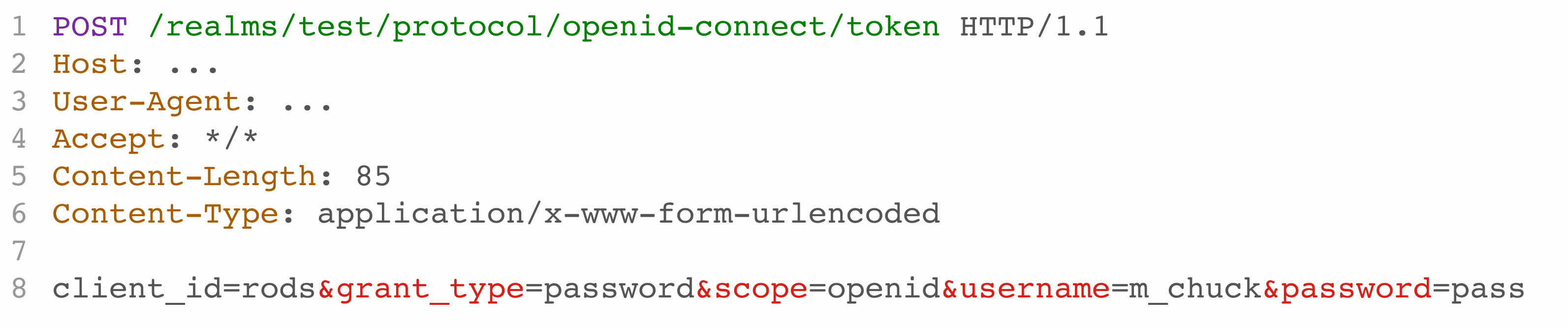
HTTP API OAuth/OIDC Flow

API Consumer Sends Authorization



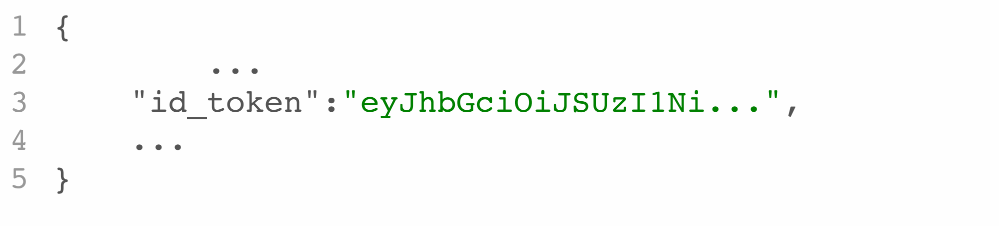
```
POST /irods-http/0.9.5/authenticate HTTP/1.1
Host: ...
User-Agent: ...
Accept: */*
Authorization: iRODS bV9jaHVjazpmZWVsc3NvZ29vZA==
```

HTTP API Forwards login to OP



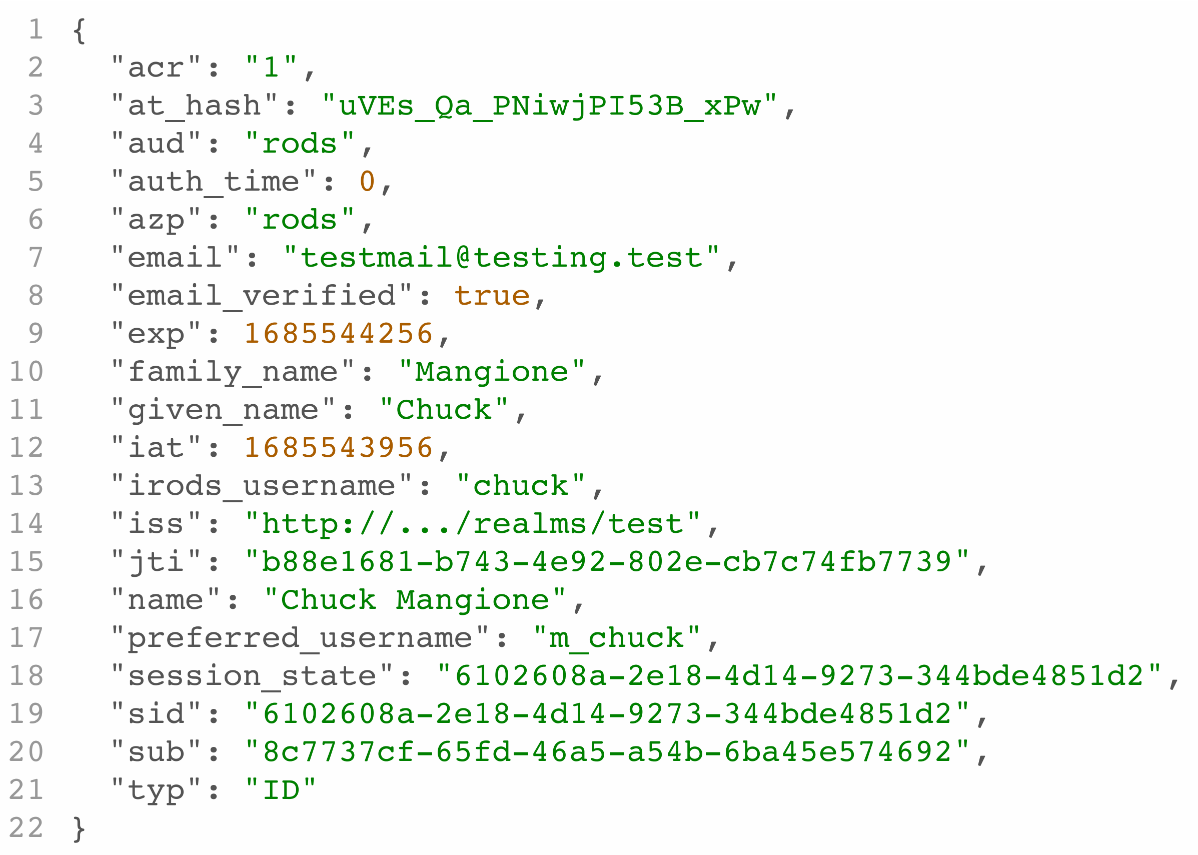
```
POST /realms/test/protocol/openid-connect/token HTTP/1.1
Host: ...
User-Agent: ...
Accept: */*
Content-Length: 85
Content-Type: application/x-www-form-urlencoded

client_id=rods&grant_type=password&scope=openid&username=m_chuck&password=pass
```

OP Provides 'id_token'



```
{
        ...
    "id_token":"eyJhbGciOiJSUzI1Ni...",
    ...
}
```

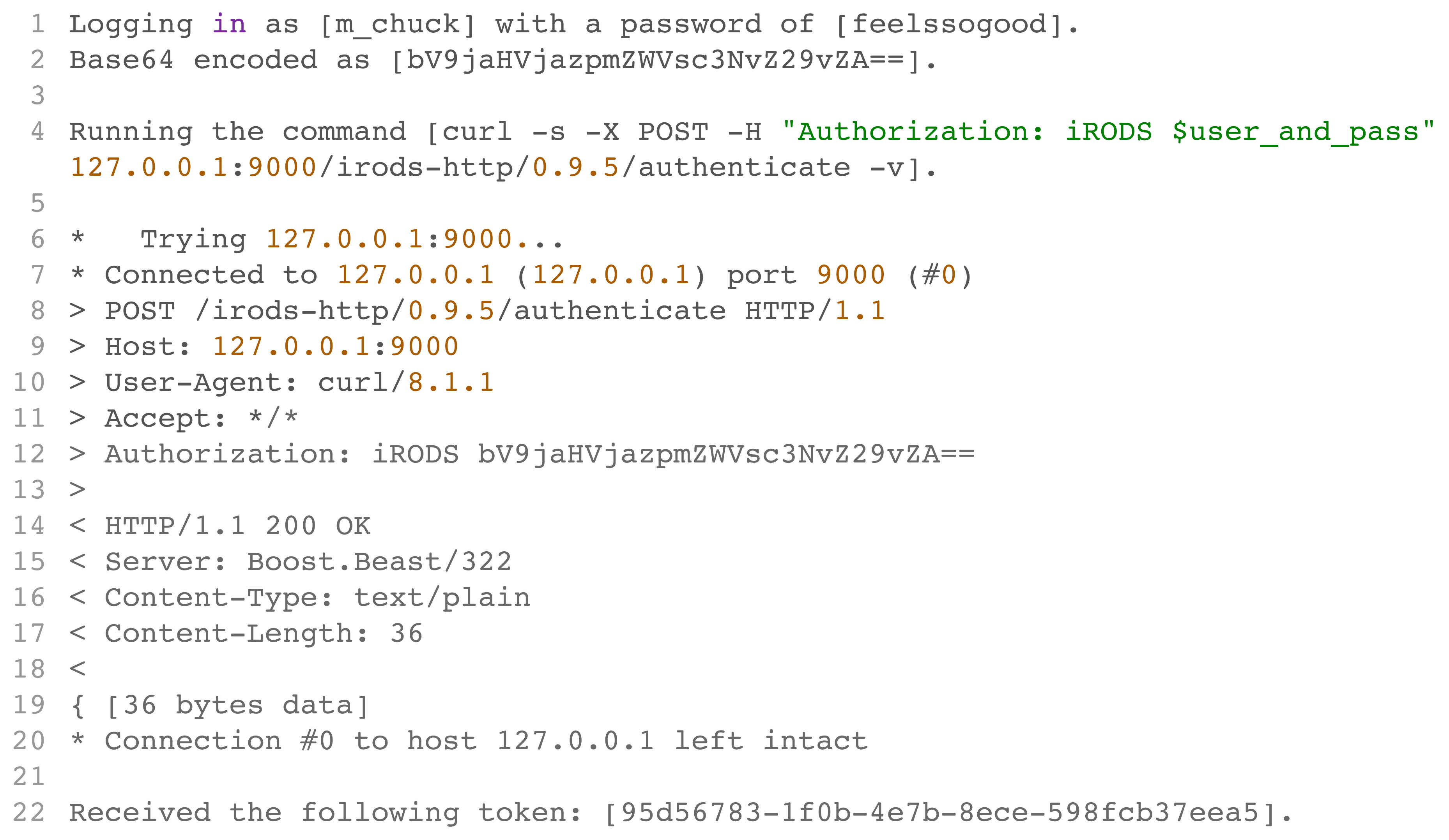
HTTP API OAuth/OIDC Flow



```json
{
  "acr": "1",
  "at_hash": "uVEs_Qa_PNiwjPI53B_xPw",
  "aud": "rods",
  "auth_time": 0,
  "azp": "rods",
  "email": "testmail@testing.test",
  "email_verified": true,
  "exp": 1685544256,
  "family_name": "Mangione",
  "given_name": "Chuck",
  "iat": 1685543956,
  "irods_username": "chuck",
  "iss": "http://.../realms/test",
  "jti": "b88e1681-b743-4e92-802e-cb7c74fb7739",
  "name": "Chuck Mangione",
  "preferred_username": "m_chuck",
  "session_state": "6102608a-2e18-4d14-9273-344bde4851d2",
  "sid": "6102608a-2e18-4d14-9273-344bde4851d2",
  "sub": "8c7737cf-65fd-46a5-a54b-6ba45e574692",
  "typ": "ID"
}
```

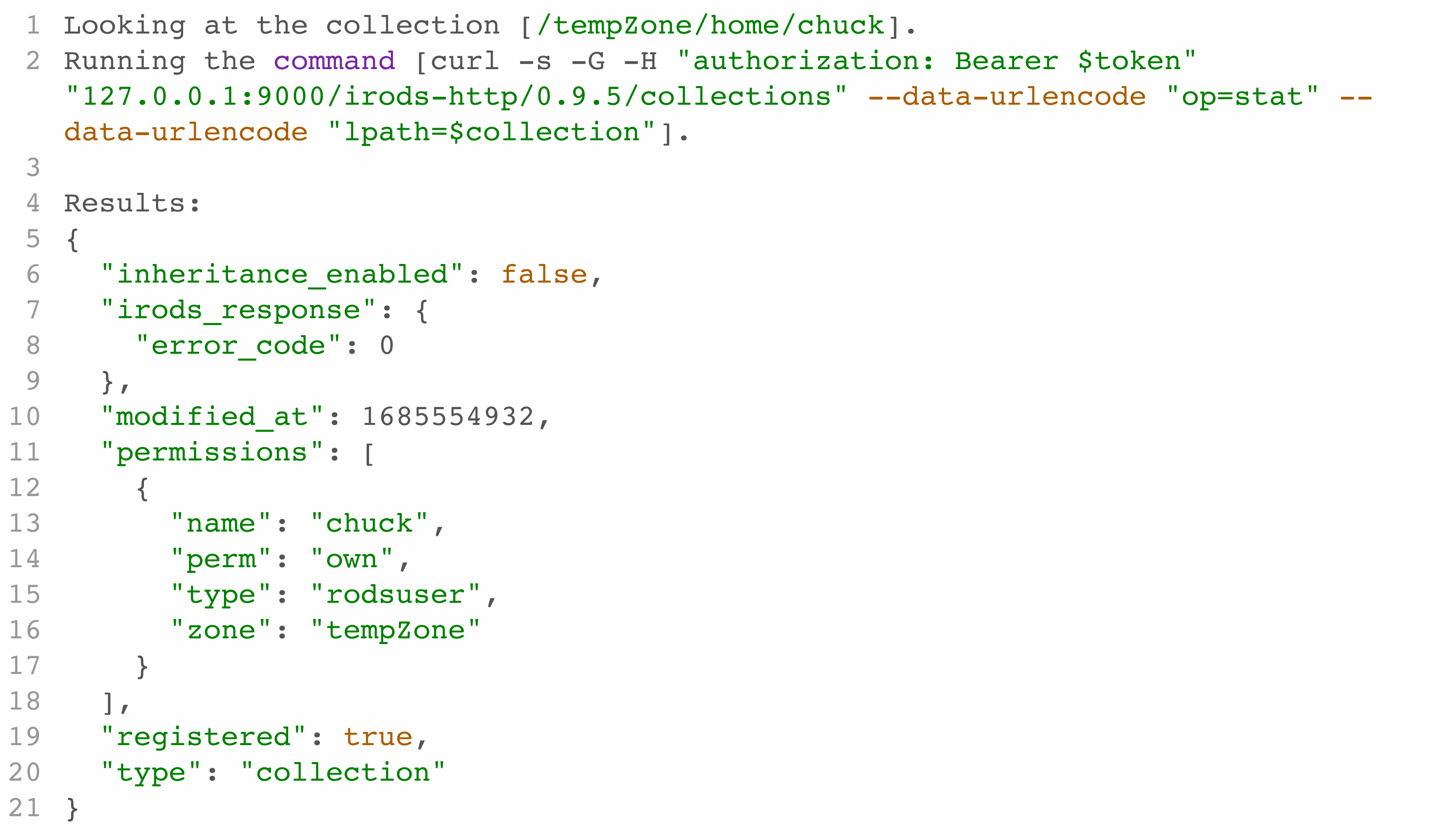
'id_token' claims

Demo Time!



```
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
<
{ [36 bytes data]
* Connection #0 to host 127.0.0.1 left intact

Received the following token: [95d56783-1f0b-4e7b-8ece-598fcb37eea5].
```

Demo Time!



```
Looking at the collection [/tempZone/home/chuck].
Running the command [curl -s -G -H "authorization: Bearer $token" "127.0.0.1:9000/irods-http/0.9.5/collections" --data-urlencode "op=stat" --data-urlencode "lpath=$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
  - https://www.ietf.org/archive/id/draft-ietf-oauth-security-topics-22.html

Thank you!

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