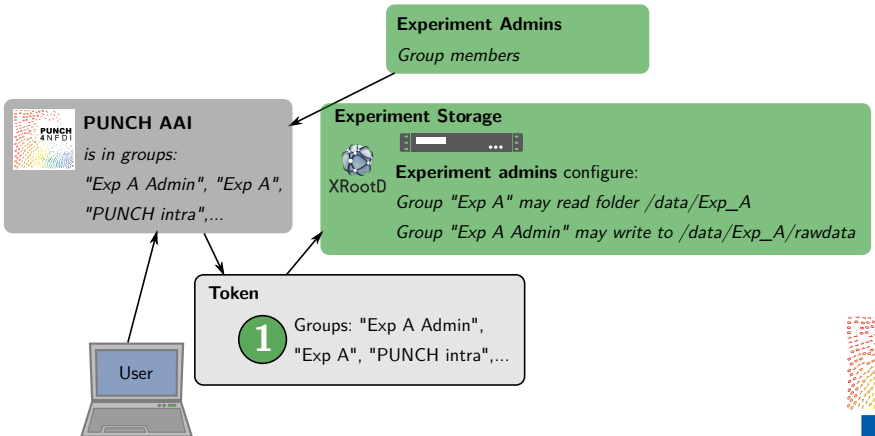
PUNCH AAI Developments

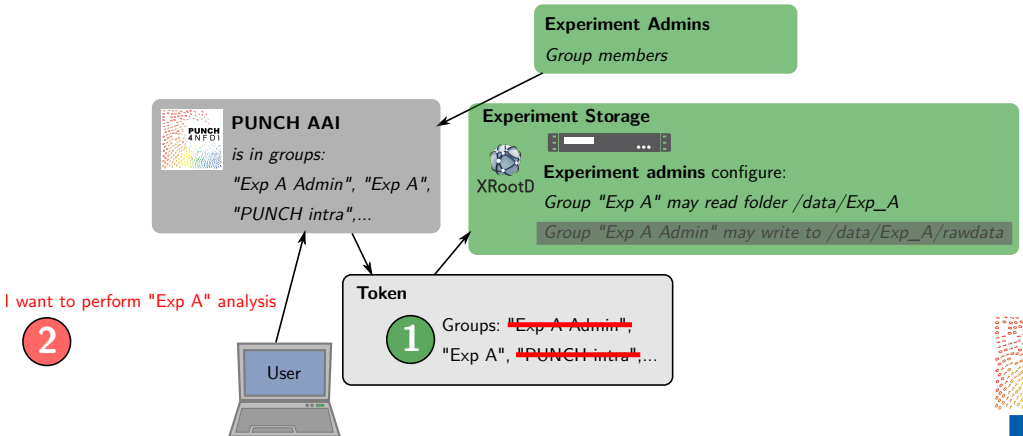
Oliver Freyermuth, Kilian Schwarz, TA6, Christoph Wissing

4th June, 2025

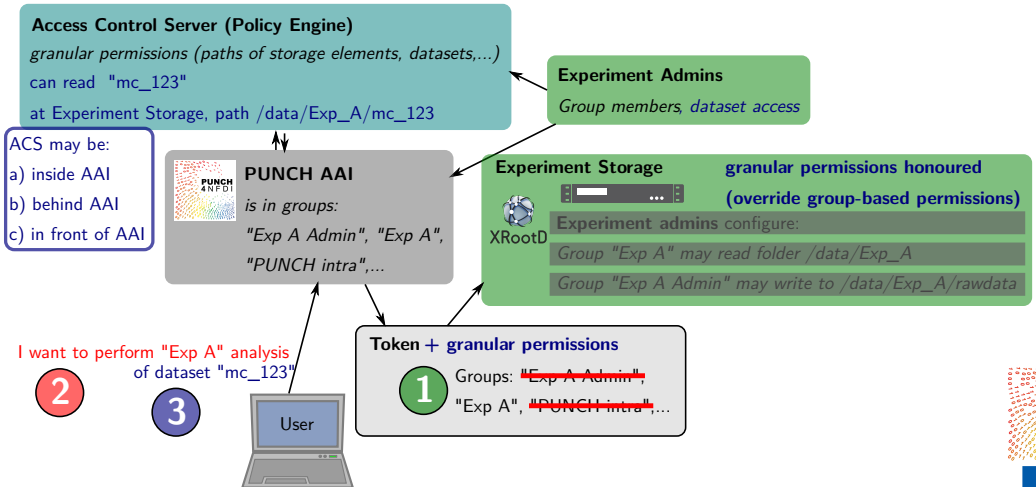
Visualization of Feature Requests



Visualization of Feature Requests



Visualization of Feature Requests



Status of Requests

- Request 1 ‘Group information in tokens’:
  ✓ Implemented by Unity before we made an official request
  ⇒ Still needs to be tested in full within PUNCH
- Request 2 ‘Claim filtering’ (i. e. not all groups in tokens):
  ✓ Implemented and tested (still need to test workflows)
- Request 3 ‘Granular permissions’
  ⌛ Description finalized, quotations for different possibilities

PUNCH 4NFDI

UNIVERSITÄT BONN

Granular Authorization

Storage

Scopes: `storage.read`, `storage.create`, `storage.modify`, `storage.write`

- Usually scoped for a specific path, i. e. `storage.read:/someexperiment/somefile.dat`
- Inspired by WLCG Common JWT Profiles which are inspired by SciTokens
- Used in the tools in Storage4PUNCH: dCache, XRootD
- May also be used inside Rucio / FTS (now or in the future)

Compute

Scopes: `compute.read`, `compute.modify`, `compute.create`, `compute.cancel`

- May be scoped to a specific compute resource
- Expected by Compute4PUNCH batch system (HTCondor)
- Inspired by WLCG Common JWT Profiles which are inspired by SciTokens

Technical Solution 'A': External Policy Service 'behind' AAI

- When a specific (custom) scope pattern is requested by the user (e. g. `storage.read:*`, an external service (with REST API) is queried by the AAI.
- The API of that external service is yet to be defined. A potential specification could be:
  - For `http_status==200`, output MUST be a single JSON object.
  - 'Permission denied' is also possible.

Example

Input AT for the user with audience being set to the external service, the triggering scope (`storage.read:/example/subdir/file`).

Output One single JSON object, the result of which is added as a sub-json to the JWT-AT which is generated for the user, Example:
`{["storage.read:/example/subdir/file"]}`

Technical Solution ‘A’: External Policy Service ‘behind’ AAI

Example JWT-AT generated by the AAI:


```
{
        "typ": "at+jwt",
        "alg": "RS256"
}
{
        "sub": "6c611e2a-2c1c-487f-9948-c058a36c8f0e",
        "aud": "public-oidc-agent",
        "scope": "openid offline_access storage.read:/example/subdir/file",
        "iss": "https://login.helmholtz.de/oauth2",
        "exp": 1683731886,
        "iat": 1683727886,
        "jti": "108ed829-9871-4f23-922b-be977be48476",
        "client_id": "public-oidc-agent"
        "storage": {
            [
                "storage.read:/example/subdir/file"
            ]
}
```

Technical Solution 'B': Token Exchange with Trusted Client

Enabling full support for token-exchange endpoints (see https://indigo-iam.github.io/v/v1.7.2/docs/reference/api/oauth-token-exchange/ and https://www.rfc-editor.org/rfc/rfc8693 allowing community-specific services, which act as registered and trusted clients of the AAI, to request access tokens with fine-grained additional scopes, e.g. scopes not present in the Bearer token and implementing capability-based authorization.
Users can then request access tokens through such a client, which decides (based on community-defined access policies) whether to perform the token exchange and to return the desired AT to the user.

Some ‘Pros’ and ‘Cons’

General

- Development of Policy Engine service with API needed (can reuse existing tools).
- We may need to adapt all our tools, middleware etc. to be able to use less generic scopes (e. g. `punch-storage.read:`) and get these changes upstream.

⇒ Independent of approach ‘A’ or ‘B’!
General goal: Try to stay close to WLCG workflows (to reuse tools).

Some ‘Pros’ and ‘Cons’

Variant A: Policy Engine ‘behind’ AAI

- 👍 Acts more like the AAI used in WLCG (Indigo), i. e. as if policy engine was embedded
- 👍 All clients can directly talk to the AAI (e. g. `oidc-agent`)
- 👍 Service needs only to be reachable from AAI
- 👎 Not a ‘standardized’ approach

Variant B: Trusted Client with Policy Engine in front of AAI

- 👍 Standardized approach
- 👎 Clients need to go via the trusted client service
- 👎 PUNCH needs to operate a world-wide reachable service exchanging tokens for users

Thank you

for your attention!

Embed group information in tokens

For now, special procedure with `oidc-agent` required (see documentation, simplification foreseen).

Access Token


```
{
 [...],
 "preferred_username": "o.freyermuth",
 "scope": "openid offline_access profile",
 "eduperson_entitlement": [
  "urn:mace:dir:entitlement:common-lib-terms",
  "urn:geant:dfn.de:nfdi.de:punch:group:PUNCH4NFDI:punch_intra#login.helmholtz.de",
  "urn:geant:h-df.de:group:HDF#login.helmholtz.de",
  "urn:geant:dfn.de:nfdi.de:punch:group:PUNCH4NFDI#login.helmholtz.de",
  "urn:geant:dfn.de:nfdi.de:punch:group:PUNCH4NFDI:elsa_one#login.helmholtz.de"
 ]
}
```

Reduced permissions in tokens

- Do not expose all groups to used service
- Work with reduced / minimal privileges

Token Request

```
scope=eduperson_entitlement:...:PUNCH4NFDI:elsa_one#login.helmholtz.de
```

Access Token

```
{
 [...],
 "preferred_username": "o.freyermuth",
 "scope": "openid offline_access profile
 ↪  eduperson_entitlement:...:PUNCH4NFDI:elsa_one#login.helmholtz.de",
 "eduperson_entitlement": [
  "urn:geant:dfn.de:nfdi.de:punch:group:PUNCH4NFDI:elsa_one#login.helmholtz.de"
 ]
}
```

Query PUNCH service for granular permissions

- Granular file access permissions require a scope policy system
- Extension in Unity not likely ⇒ external Access Control Server

Token Request

```
scope=storage.read:/example/subdir/file
```

Access Token

```
{
 [...],
 "preferred_username": "o.freyermuth",
 "scope": "openid offline_access profile storage.read:/example/subdir/file",
 "storage": {
          [
   "storage.read:/example/subdir/file"
  ]
 }
}
```