

**IETF 116**  
**Yokohama**  
**March 2023**

**Aaron Parecki**  
**Mike Jones**  
**Ben Schwartz**

# **OAuth 2.0 Protected Resource Metadata**

## **Authorization Server Discovery**

<https://datatracker.ietf.org/doc/html/draft-jones-oauth-resource-metadata/>  
<https://datatracker.ietf.org/doc/html/draft-parecki-oauth-authorization-server-discovery/>

# Two drafts with a subset of overlapping functionality

- Commonality
  - Both enable Resource Servers to identify Authorization Servers for a Client to use
- Differences
  - draft-jones-oauth-resource-metadata
    - .well-known container for extensible set of metadata about the resource
    - Analogous to Authorization Server Metadata data structures defined by RFC 8414
    - `authorization_servers` metadata value is an array of Authorization Server issuers
    - Can also publish keys, algorithms, documentation, etc.
  - draft-parecki-oauth-authorization-server-discovery
    - Provides exactly one piece of metadata - an Authorization Server issuer value
    - Provides it in `WWW-Authenticate` response with `issuer` parameter
- Mike will describe first approach - Aaron will describe second
- *Thanks to Aaron for suggesting combining discussions!*

# OAuth 2.0 Protected Resource Metadata

# Example Protected Resource Metadata Request

GET /.well-known/oauth-protected-resource HTTP/1.1


Host: resource.example.com

# Example Protected Resource Metadata Response

HTTP/1.1 200 OK

Content-Type: application/json

```
{
  "resource":
    "https://resource.example.com",
  "authorization_servers":
    ["https://as1.example.com/",
     "https://as2.example.net/"],
  "bearer_methods_supported":
    ["header", "body"],
  "resource_documentation":
    "http://resource.example.com/resource_documentation.html"
}
```



This is the metadata element that tells Clients what Authorization Server issuer URLs they can use with this Protected Resource

# History

- 2016: Protected Resource Metadata draft created in parallel with Authorization Server Metadata draft (which became RFC 8414)
  - AS Metadata was in use at the time and progressed by the WG
  - Protected Resource Metadata was not in use, and was not adopted
- 2022: Protected Resource Metadata reference added to OpenID Connect Federation specification
- 2023: PR Metadata in production use in Italian Federation deployments
  - Required by [https://italia.github.io/spid-cie-oidc-docs/en/metadata\\_aa.html](https://italia.github.io/spid-cie-oidc-docs/en/metadata_aa.html)
- This week - new draft published incorporating IANA feedback

# Authorization Server Discovery

# Who is this for?

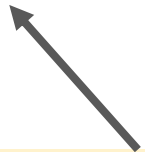
- Calendar / email apps that work with many resource servers and authorization servers with no prior relationship to either



# Step 1: The Trigger

```
OPTIONS /home/bemasc/calendars HTTP/1.1
Host: cal.example.com

HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer issuer="https://authorization-server.com/" scope="read"
```



The specifics of this header are TBD, the important part is it has the full issuer URL of the authorization server.

Note: The authorization server URL could be under the control of the resource server or a completely unrelated server depending on how you want to deploy it.

## Step 2: Client Discovers AS Metadata


GET https://authorization-server.com/.well-known/oauth-authorization-server HTTP/1.1

HTTP/1.1 200 Ok

Content-Type: application/json

```
{
  "issuer": "https://authorization-server.com/",
  "authorization_endpoint": "https://authorization-server.com/authorize",
  "token_endpoint": "https://authorization-server.com/oauth/token",
  "registration_endpoint": "https://authorization-server.com/oauth/clients",
  "response_types_supported": "code",
  ...
}
```

Where to open the browser to



Where to get the tokens from

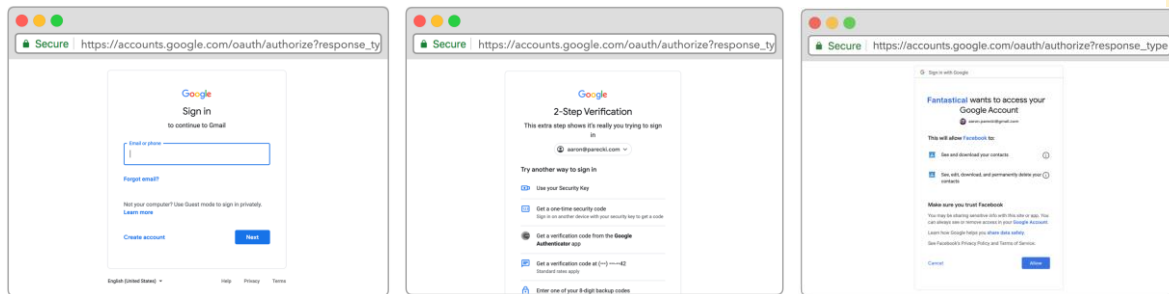


# Step 3: Initiate OAuth Flow

Client launches a browser to initiate the OAuth flow...

`https://authorization-server.com/authorize?client_id=***&redirect_uri=***&scope=read  
&code_challenge=XXXX&code_challenge_method=S256&state=XXX`

Normal OAuth flow proceeds,  
enabling strong MFA and  
passwordless, as well as SSO



Note: The client\_id could be:

- Pre-registered out of band
- Registered dynamically via RFC7591
- Provided as a URI according to a new specification

Note: The redirect\_uri could be

- Custom URL scheme
- localhost:port
- "out-of-band"

## Step 4: OAuth flow is complete

OAuth flow completes, authorization server redirects to `redirect_uri` with authorization code, client exchanges code for an access token

```
POST /oauth/token HTTP/1.1
Host: authorization-server.com
Content-type: application/x-www-form-urlencoded
```

```
grant_type=authorization_code
&client_id=***
&code_verifier=XXXX
```

```
HTTP/1.1 200 OK
Content-type: application/json
```

```
{
  "token_type": "Bearer",
  "expires_in": 86400,
  "access_token": "XXXXXXXXXX",
  "refresh_token": "YYYYYYYYYY",
  "scope": "read"
}
```

Note: Refresh token is up to the discretion of the AS, but can be used to get a new token when the current one expires if the AS doesn't need the user to re-authenticate themselves.

## Step 5: Resource request

Client uses access token to fetch data

```
GET /home/bemasc/calendars HTTP/1.1
Host: cal.example.com
Authorization: Bearer XXXXXXXXXX
```

CALENDAR DATA RESPONSE

...

Note: There are opportunities here to also leverage the new step-up OAuth draft as well, if the RS wants the user to come back with a new or different access token

Next Steps

# Possible Next Steps

- Deliberate on overlapping functionality between the two drafts
  - *The point of this combined presentation!*
  - Possibly combine approaches?
  - For instance, could add `WWW-Authenticate resource_metadata` response to `draft-jones-oauth-resource-metadata`
- Note temporal differences in mechanisms
  - `.well-known` operates at configuration/set-up time
  - `WWW-Authenticate` operates just-in-time at request time
  - For instance, calendar apps may have a configuration phase
- Working group adoption of `draft-jones-oauth-resource-metadata`?
  - *Because it's now in use*