UNIVERSITY OF
Southampton

UNIVERSITY OF Southampton

Authentication and Authorisation

COMP3220 Web Infrastructure

Dr Nicholas Gibbins – nmg@ecs.soton.ac.uk

HTTP authentication

Simple authentication of user agent using HTTP headers

- Origin server sends `WWW-Authenticate:` header containing challenge (scheme and realm)
- User agent sends `Authorization:` header containing credentials

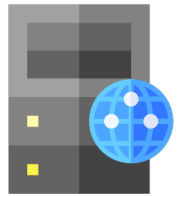
Authentication schemes:

- `Basic`
  User agent sends base64(username + ":" + password)
- `Bearer`
  OAuth 2.0 token

Basic authentication scheme

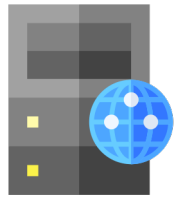
```
GET / HTTP/1.1
Host: example.org
```

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Access to Example"
```



```
GET / HTTP/1.1
Host: example.org
Authorization: Basic c3F1ZWFtaXNoOm9zc2lmcmFnZQo=
```

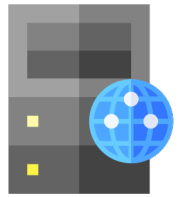
```
HTTP/1.1 200 OK

...
```



Basic authentication scheme

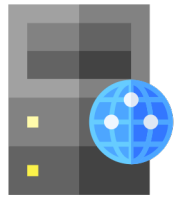
```
GET / HTTP/1.1
Host: example.org
```

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Access to Example"
```



```
GET / HTTP/1.1
Host: example.org
Authorization: Basic bmV2ZXJnb2luZ3RvZ2l2ZTp5b3V1cAo=
```

```
HTTP/1.1 403 Forbidden
```



Proxy authentication

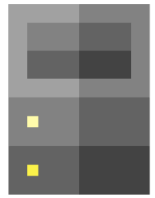
Similar challenge/response mechanism for authenticating with a proxy

- Proxy sends `Proxy-Authenticate:` header containing challenge
- User agent sends `Proxy-Authorization:` header containing credentials

Proxy authentication

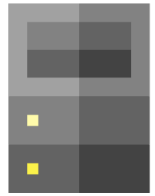
```
GET http://example.org/ HTTP/1.1
Host: example.org
```

```
HTTP/1.1 407 Proxy Authentication Required
Proxy-Authenticate: Basic realm="Access to Proxy"
```



```
GET http://example.org/ HTTP/1.1
Host: example.org
Proxy-Authorization: Basic c3F1ZWFtaXNoOm9zc2lmcmFnZQo=
```

```
HTTP/1.1 200 OK

...
```



OAuth 2.0

Modern Web applications are federations of interacting services

The password problem: how can we give an application access to our data held by a service (a "protected resource") without giving it our password for that service?

This is a problem of authorisation, rather than simply authentication

OAuth 2.0 roles

The resource owner

- An entity capable of granting access to a protected resource. May be a combination of a person (an end-user) and their user agent

The resource server

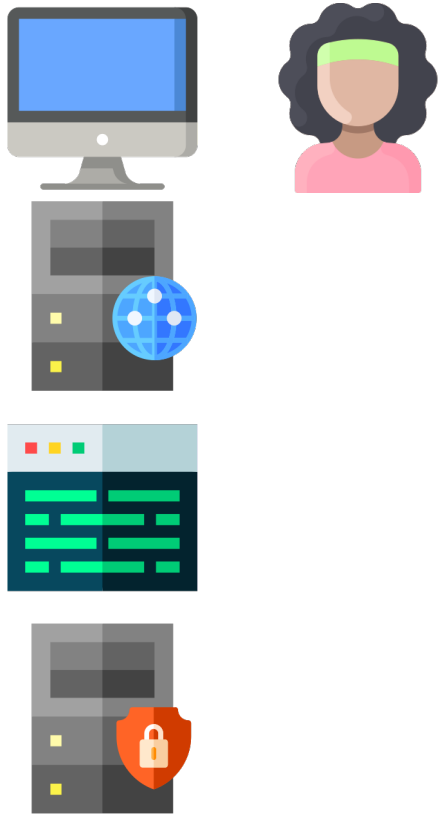
- The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens

The client

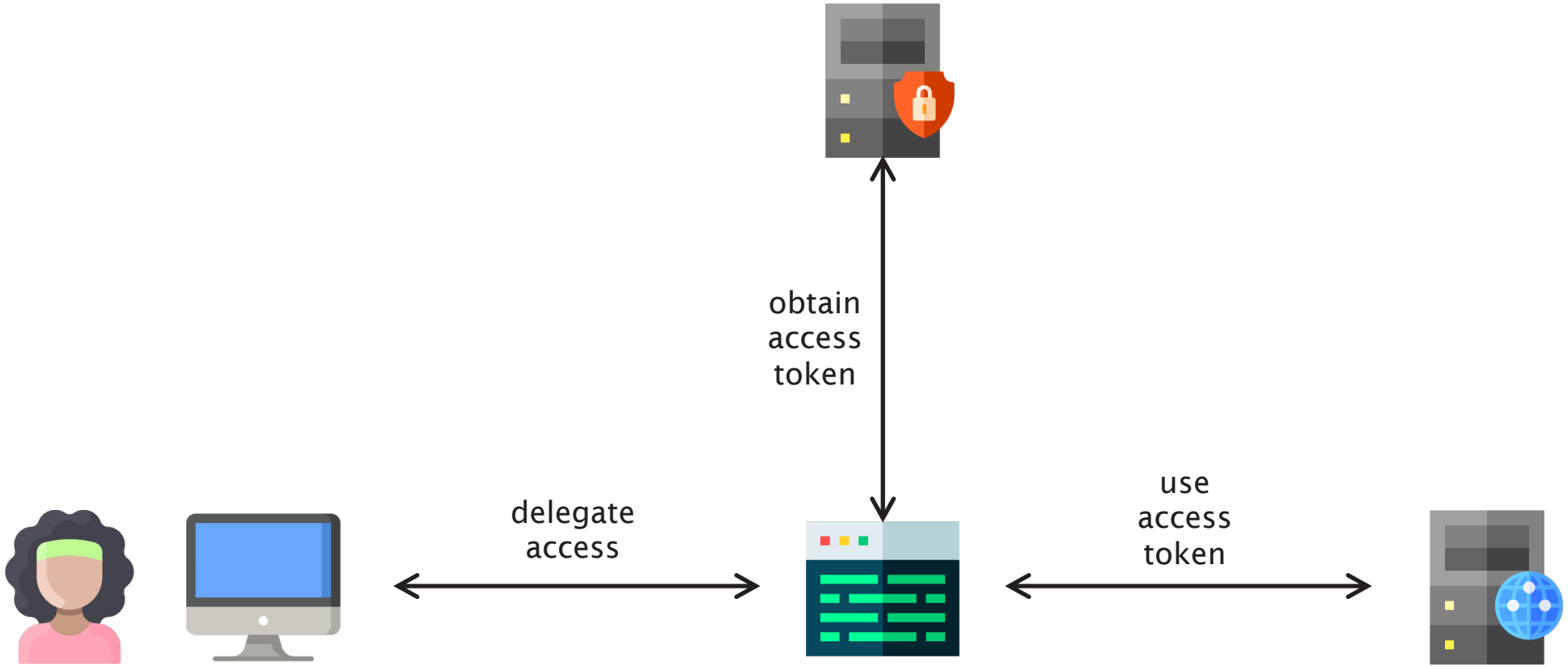
- An application making protected resource requests on behalf of the resource owner and with its authorisation

The authorisation server

- The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorisation.



Abstract protocol flow



OAuth 2.0 protocol flows

Authorisation Code Grant

- Client obtains access token directly from authorisation server ("3-legged")
- Flow relies on redirection via user agent

Implicit Grant

- All communication goes through user agent ("2-legged")
- Commonly used by single page applications

Resource Owner Password Credentials Grant

- Resource owner gives credentials to client (requires trust relationship)

Client Credential Grant

- Client authenticates directly with authorisation server

Registration

Client registers with the authorisation server before the protocol starts

- Client issued with a `client_id` – unique string (not a secret)

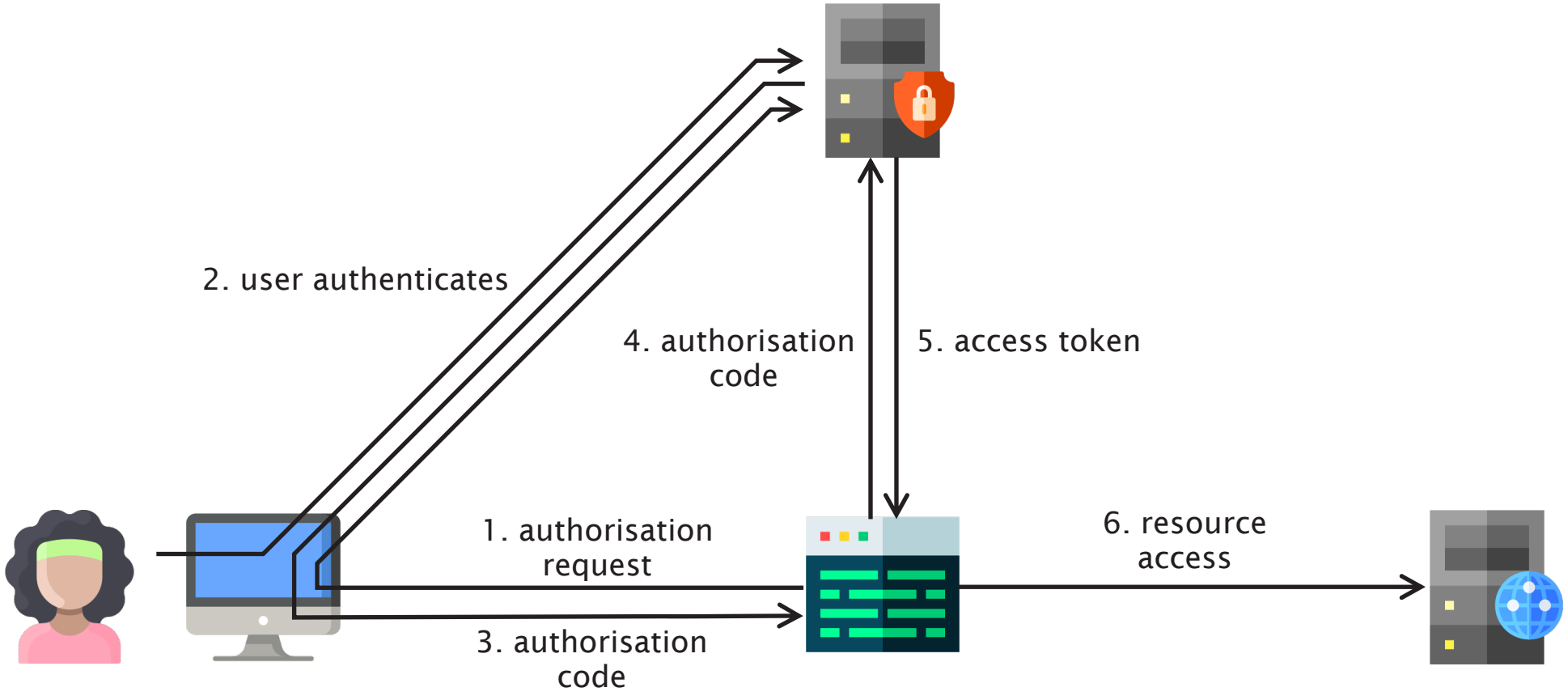
Client and authorisation server establish an authentication method

- Client password, use with HTTP Basic Authentication
- `client_id`, `client_secret` sent in request body (not recommended)

Endpoints

- Authorisation endpoint (e.g. `https://auth.org/auth`)
- Client redirection endpoint (e.g. `https://client.org/cb`)
- Token endpoint (e.g. `https://auth.org/token`)

Authorisation Code Grant



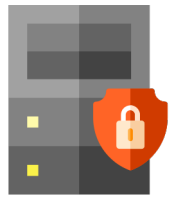

```
HTTP/1.1 302 Found
Location: https://auth.org/auth?
  response_type=code&
  client_id=s6BhdRkqt3&
  redirect_uri=https://client.org/cb
```

authorisation endpoint

issued during registration

redirection endpoint

```
GET /auth?response_type=code&client_id=s6BhdRkqt3&redirect_uri=
   https%3A%2F%2Fclient%2Eorg%2Fcb HTTP/1.1
Host: auth.org
```



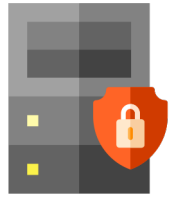
```
HTTP/1.1 302 Found
Location: https://client.org/cb?code=SplxlOBeZQQYbYS6WxSbIA
```

authorisation code

```
GET /cb?code=SplxlOBeZQQYbYS6WxSbIA HTTP/1.1
Host: client.org
```

4. authorisation code

5. access token



```
POST /token HTTP/1.1
Host: auth.org
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&
code=SplxlOBeZQQYbYS6WxSbIA&
redirect_uri=https://client.org/cb
```

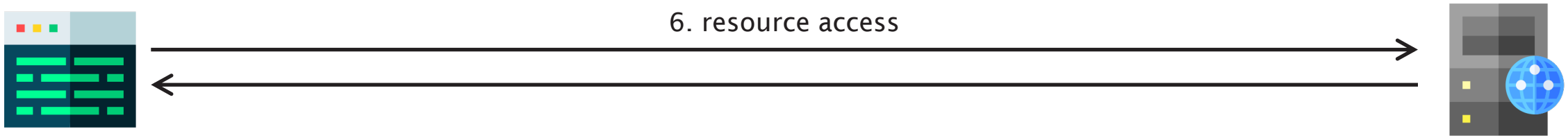
agreed during registration

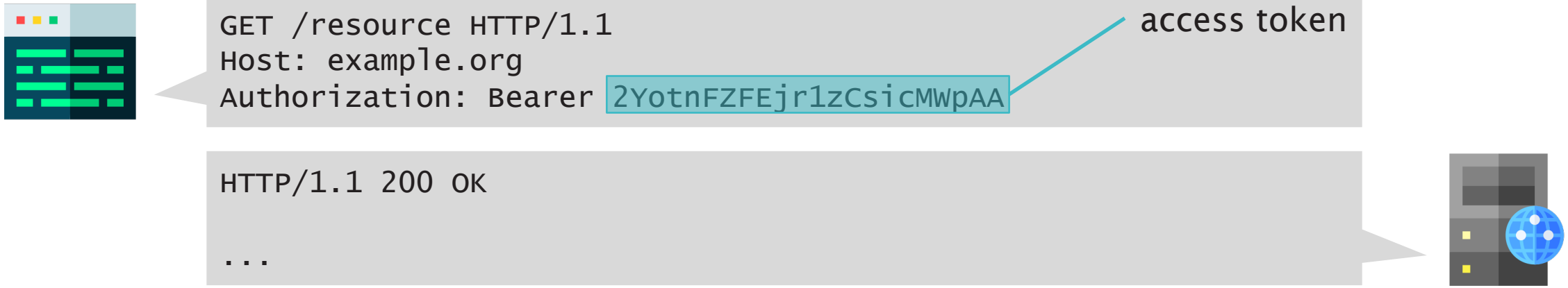
authorisation code

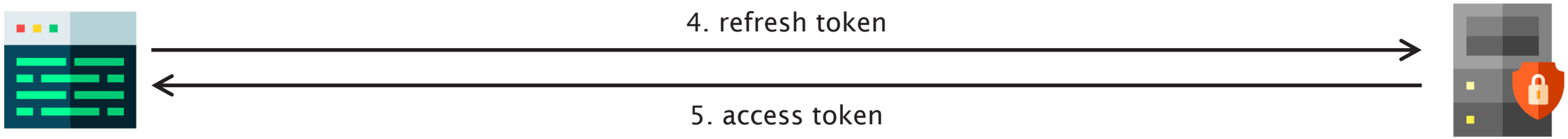
```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store

{ "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"bearer",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA" }
```

used to access resource

used to get new access token


6. resource access


GET /resource HTTP/1.1
Host: example.org
Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA
access token
HTTP/1.1 200 OK
...



```
POST /token HTTP/1.1
Host: auth.org
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&
refresh_token=tGzv3JOkF0XG5Qx2TlKWIA
```

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store

{ "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"bearer",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA" }
```

Further reading

Fielding, R. and Reschke, J. (2014) *Hypertext Transfer Protocol (HTTP/1.1): Authentication*. RFC7235.

https://tools.ietf.org/html/rfc7235

Reschke, J. (2015) *The 'Basic' HTTP authentication scheme*. RFC7617.

https://tools.ietf.org/html/rfc7617

Hardt, D. (2012) *The OAuth 2.0 authorization framework*. RFC6749.

https://tools.ietf.org/html/rfc6749

Jones, M. and Hardt, D. (2012) *The OAuth 2.0 Authorization Framework: Bearer Token Usage*. RFC6750

https://tools.ietf.org/html/rfc6750