This work is licensed under CC BY 4.0

Authentication

In a RESTful API



Attack Vectors – WEB APIs



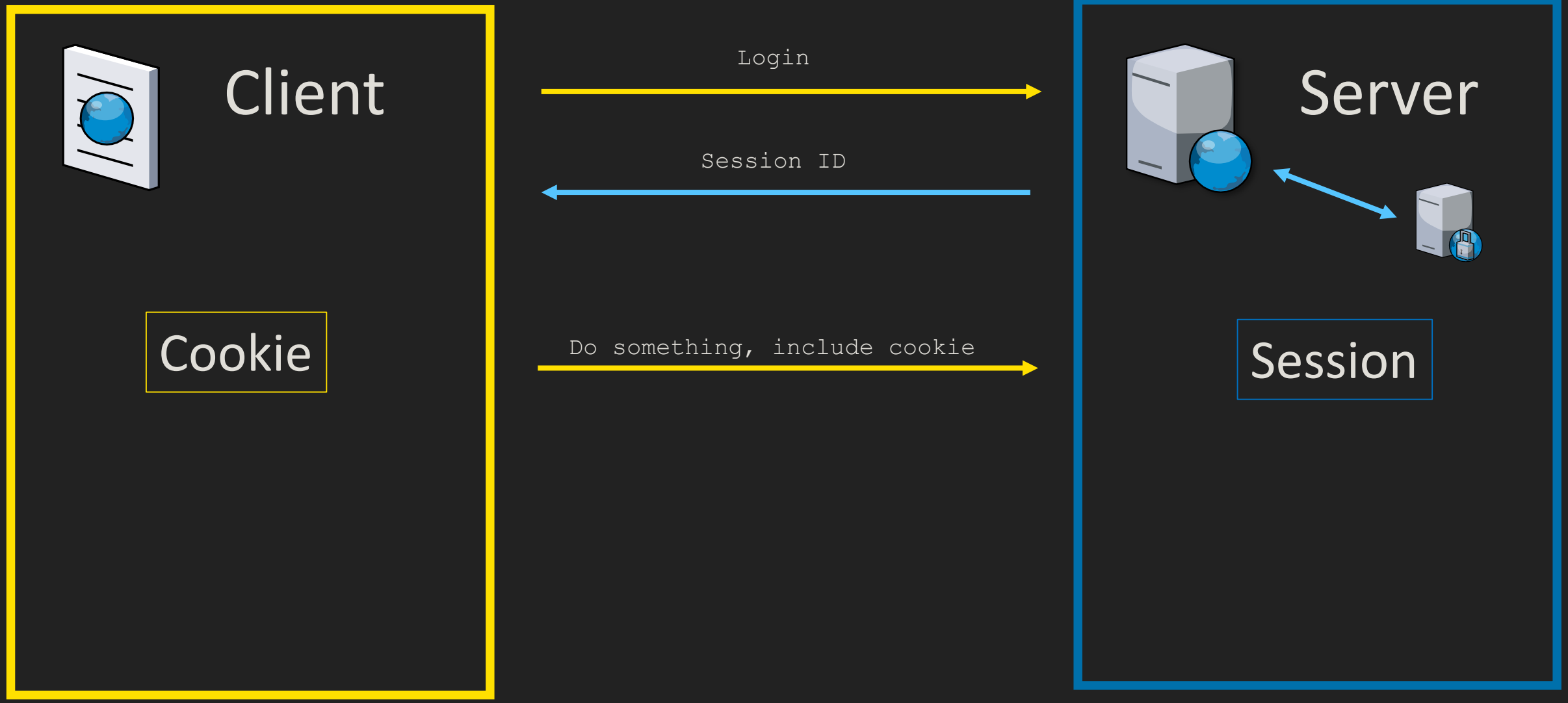
- New technology, old treats
  - Injections, XSS, CSRF… be suspicious!
    - FIEO, Filter input, escape output
- HTTPS everywhere!
- Long lived tokens/passwords/”sessions”
  - Replay attacks
- Authentication/Authorization
  - Protecting resources
  - Stateless? Sessions? Patterns?

Strategies – Authn, Authz

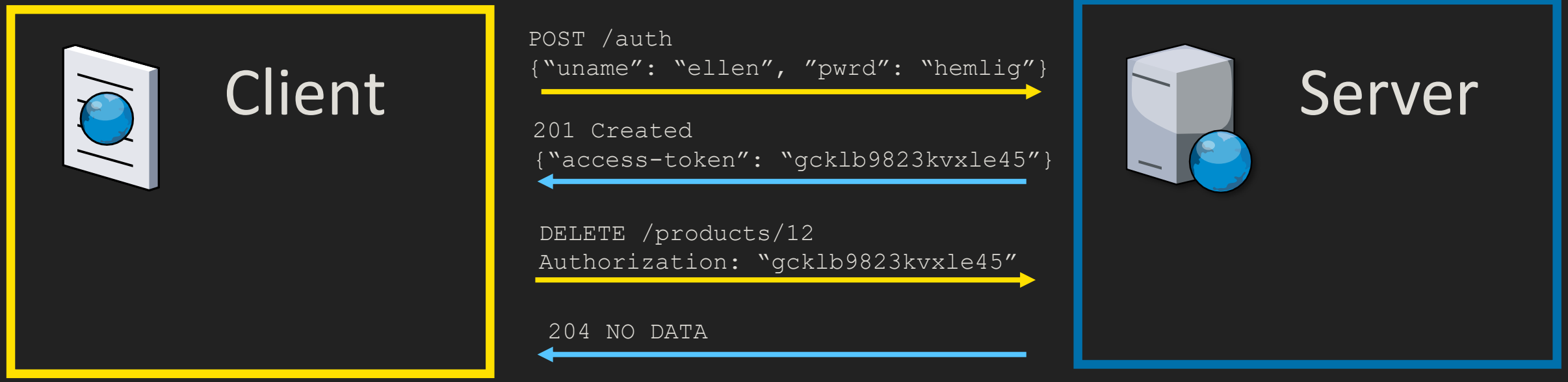


- ✓ Open and public! Read-only!
- ✓ HTTP Authentication
- ✓ API keys
- ✓ OAuth 2.0, OpenID Connect
- ✓ Token-based authentication
  - JWT
- ✓ (Kerberos, Public key, WS-Trust, SAML)

Session based authentication



Token based authentication



- ✓ Avoid sending in URL – There are logging proxies out there
- ✓ Use custom header or Authorization header (Bearer)
- ✓ Time limit
- ✓ Avoid resending username:password each time

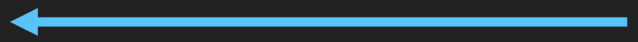
Base64 encoding?

```
This is an example with
text and other...
{
   "name": "Ellen",
   "age": 123
}
```

btoa('This is..')



atob('Vghpcy..')



```
VGhpcyBpcyBhbiBleGFtcGxlIHdpdGggdGV4dCBhbmQgb3RoZXIuLi4KewogICAibmFtZSI6ICJFbGxlbiIsCiAgICJhZ2UiOiAxMjMKfQo=
```

In node.js:

```
// Create a buffer
const buff = Buffer.from('This is an example...', 'utf-8')

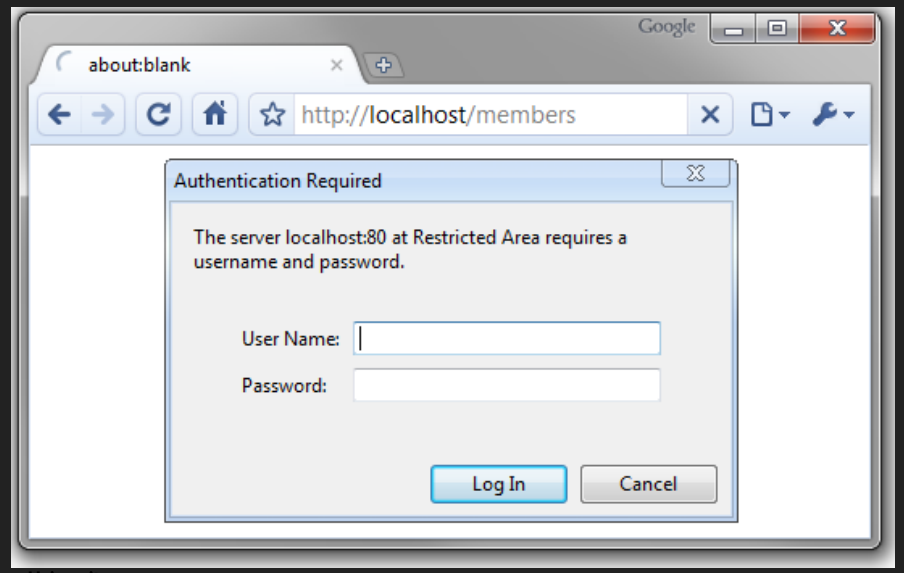
// Decode the buffer as Base64
const base64 = buff.toString('base64')
```

HTTP Basic Authentication

```
// Header from Server - A challenge 401 Unauthorized HTTP/1.1 WWW-
Authenticate: Basic realm="my api"

// Header from client
Authorization: Basic Kl52osuDS3DH6H12JDe543
```

base64-encoding of username:password
- not encrypted!!!

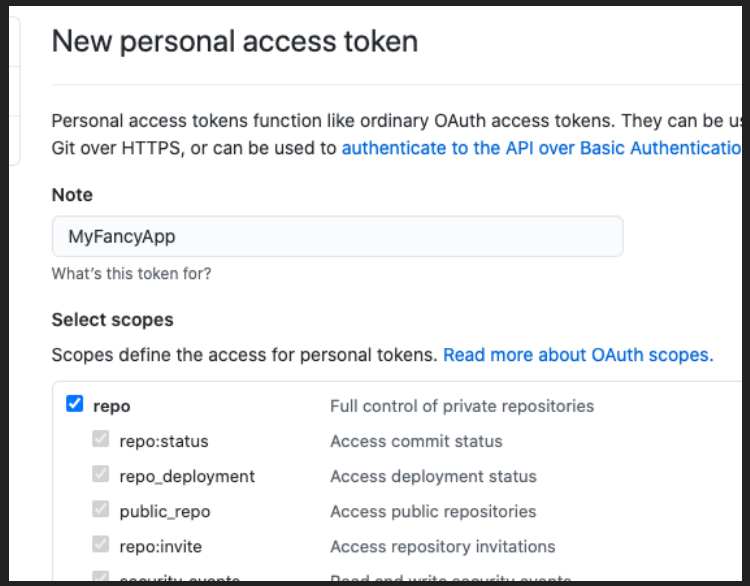


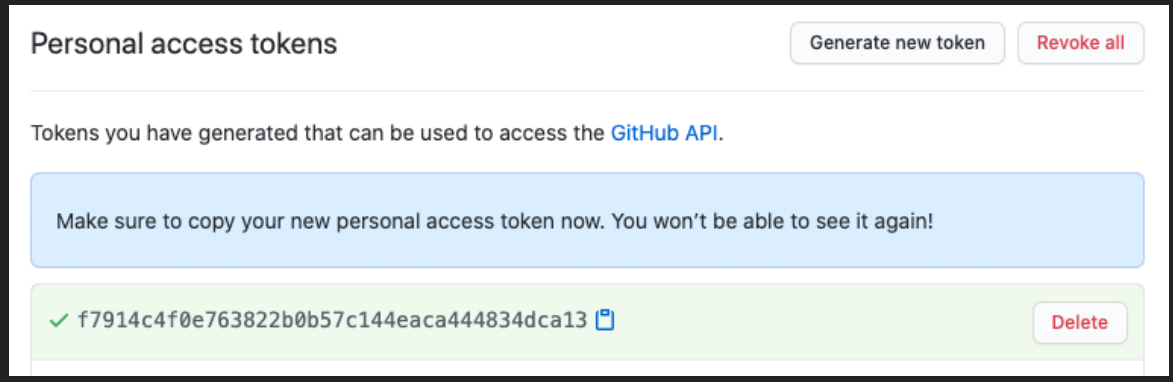
A more "secure" model was developed, "HTTP Digest Authentication" Uses MD5...

API-keys

- ✓ Authentication/Authorization
  - vs. username/password
  - long-lived?
  - Easy to leak to public?
  - “API keys are not API Security”

- ✓ Internal applications that only access a specific users data.



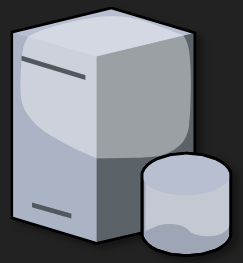


JWT [‘JOT’]

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdW
IiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lI
iwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT
4fwpMeJf36POk6yJV_adQssw5c
```

- ✓ JSON Web Token
- ✓ Industry Standard RFC 7519
- ✓ Part of “JSON Identity Suite”
  - JWE – Encryption
  - JWA – Algoritms
  - JWK – A JSON data structure that represents a cryptographic key
  - JWS – For signing arbritary data (inforamtion has not changed since signed).
- ✓ JWTs are self-contained – Carry all info in it self – Token by value
  - ”JSON”, greate language support
  - Every one can see the info (if not encrypted JSON Web Encryption)
    - Protected from manipulation
  - Could use multiple JWTs

JWT (Symmetric key)



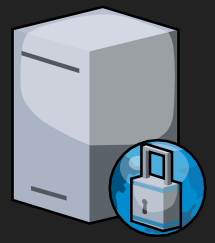
Client

Saves the JWT and uses it for future attempts to read resources

`POST /login/ Username/Password`

`JWT`

`Authorization: Bearer JWT…`



Auth server

Key



Resource server

Key

JWT

- ✓ Header eyJh….J9
- ✓ Payload Jpc3…19
- ✓ Signature S0o6….SY

First two is JSON! – but base64-encoded

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
Jpc3MiOiJsbnUuc2UvZWxsZW4iLCJzdWIiOiJzZ
XJ2aWNlIDEiLCJleHAiOiIxMzAwODE5MzgwIiwi
anRpIjoiMTIzMDAxMjMiLCJuYW1lIjoiSm9obiB
Eb2UiLCJhZG1pbiI6dHJ1ZSwic2NvcGUiOlsicH
JvZmlsZSIsImVtYWlsIiwicmVzb3VyY2VzIl19.
SOo6v-
ehINbYK4u98kIbZR87HDTFYlAMLZfeMFuLrSY
```

JWT - Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
Jpc3MiOiJsbnUuc2UvZWxsZW4iLCJzdWIiOiJzZ
XJ2aWNlIDEiLCJleHAiOiIxMzAwODE5MzgwIiwi
anRpIjoiMTIzMDAxMjMiLCJuYW1lIjoiSm9obiB
Eb2UiLCJhZG1pbiI6dHJ1ZSwic2NvcGUiOlsicH
JvZmlsZSIsImVtYWlsIiwicmVzb3VyY2VzIl19.
SOo6v-
ehINbYK4u98kIbZR87HDTFYlAMLZfeMFuLrSY
```

base64encode on the above will get you the first part of the JWT.

HS256: Symmetric encryption
RS256: Asymmetric encryption

JWT - Payload

```
{
  "iss": "lnu.se/ellen",
  "sub": "service 1",
  "exp": "1300819380",
  "jti": "12300123",
  "name": "ellen",
  "admin": true,
  "scope": ["profile", "email", "resources"]
}
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
Jpc3MiOiJsbnUuc2UvZWxsZW4iLCJzdWIiOiJzZ
XJ2aWNlIDEiLCJleHAiOiIxMzAwODE5MzgwIiwi
anRpIjoiMTIzMDAxMjMiLCJuYW1lIjoiSm9obiB
Eb2UiLCJhZG1pbiI6dHJ1ZSwic2NvcGUiOlsicH
JvZmlsZSIsImVtYWlsIiwicmVzb3VyY2VzIl19.
SOo6v-
ehINbYK4u98kIbZR87HDTFYlAMLZfeMFuLrSY
```

base64encode on the above will get you the middle part of the JWT.
Can be read by the client! No sensitive data please!

JWT – Payload – Registered Claims

- ✓ iss
  - Issuer. Identifies the party that issued the JWT
- ✓ sub
  - Subject. Identifies the party that this JWT carries information about
- ✓ aud
  - Audience. Identifies for which this JWT is for
- ✓ exp
  - Expire. Number with specific timestamp as definiet by POSIX-UNIX-date (sec since 1970-01-01 00:00)
- ✓ nbf
  - Not before. Timestamp from when to be valid
- ✓ iat
  - issued at. When the JWT was issued
- ✓ jti
  - JWT ID. A unique identifier for this JWT

JWT - Signature

```
let encodedString = base64UrlEncode(header) + '.' + base64UrlEncode(payload)
let signature = HMACSHA256(encodedString, 'secret key')
```



https://jwt.io/

- ✓ Secret only known by the server (so it could validate the content)

Example application

```
import jwt from 'jsonwebtoken'
```

```
async login (req, res, next) {
  try {
    const user = await User.authenticate(req.body.email, req.body.password)

    const payload = {
      first_name: user.firstName,
      last_name: user.lastName,
      email: user.email,
      permission_level: user.permissionLevel
    }

    // Create the access token with the shorter lifespan.
    const accessToken = jwt.sign(payload, process.env.ACCESS_TOKEN_SECRET, {
      algorithm: 'HS256',
      expiresIn: '1h' // process.env.ACCESS_TOKEN_LIFE
    })
```

sign

.example.env

```
# ------------------------------------------------------------------------
# NOTE! Never push the .env file to a repository. In this solution,
#       this example has been added to document the environment variables.
# ------------------------------------------------------------------------
DB_CONNECTION_STRING={ The MongoDB connection string. }
PORT={ the port that the server listens for after connections. }
ACCESS_TOKEN_SECRET=swsh23hjddnns
ACCESS_TOKEN_LIFE=12000
REFRESH_TOKEN_SECRET=dhw782wujnd99ahmmakhanjkajikhiwn2n
REFRESH_TOKEN_LIFE=8640000
```

```
const authenticateJWT = (req, res, next) => {
  const authorization = req.headers.authorization?.split(' ')

  if (authorization?.[0] !== 'Bearer') {
    next(createError(401))
    return
  }

  try {
    req.jwt = jwt.verify(authorization[1], process.env.ACCESS_TOKEN_SECRET)
    req.user = {
      email: req.jwt.email,
      firstName: req.jwt.first_name,
      lastName: req.jwt.first_name,
      permissionLevel: req.jwt.permission_level
    }
```

verify

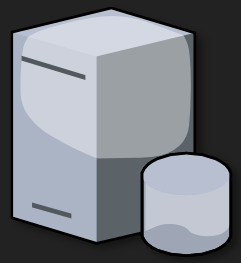
Recall an JWT?

If an JWT expires in two years? How to revoke it?

JWT with Refresh token (Symmetric key)

Ev. store refresh token state



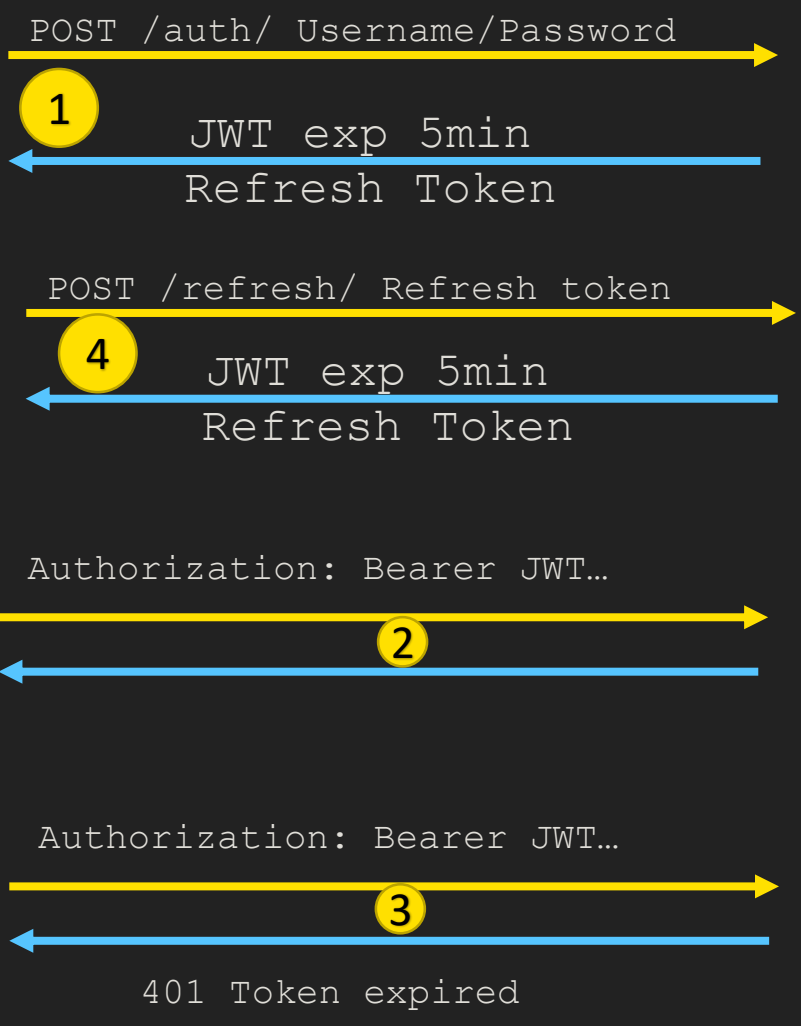


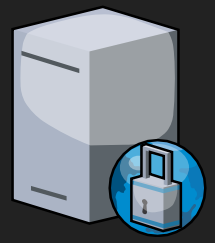
Client

Saves the JWT and uses it for future attempts to read resources.

Stores the refresh token to use when JWT expires.

Could refresh JWT prematurely





Auth server

Key

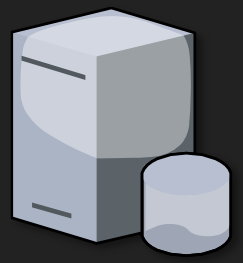


Resource server

Key

JWT (Asymmetric key)

```
{
  "alg": "RS256",
  "typ": "JWT"
}
```



Client

Notning different for the client

`POST /login/ Username/Password` →

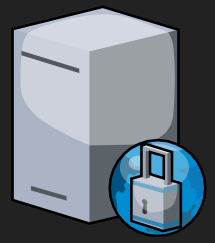
← `JWT`

`Authorization: Bearer JWT…` →

←

`Authorization: Bearer JWT…` →

←



Auth server

Private key



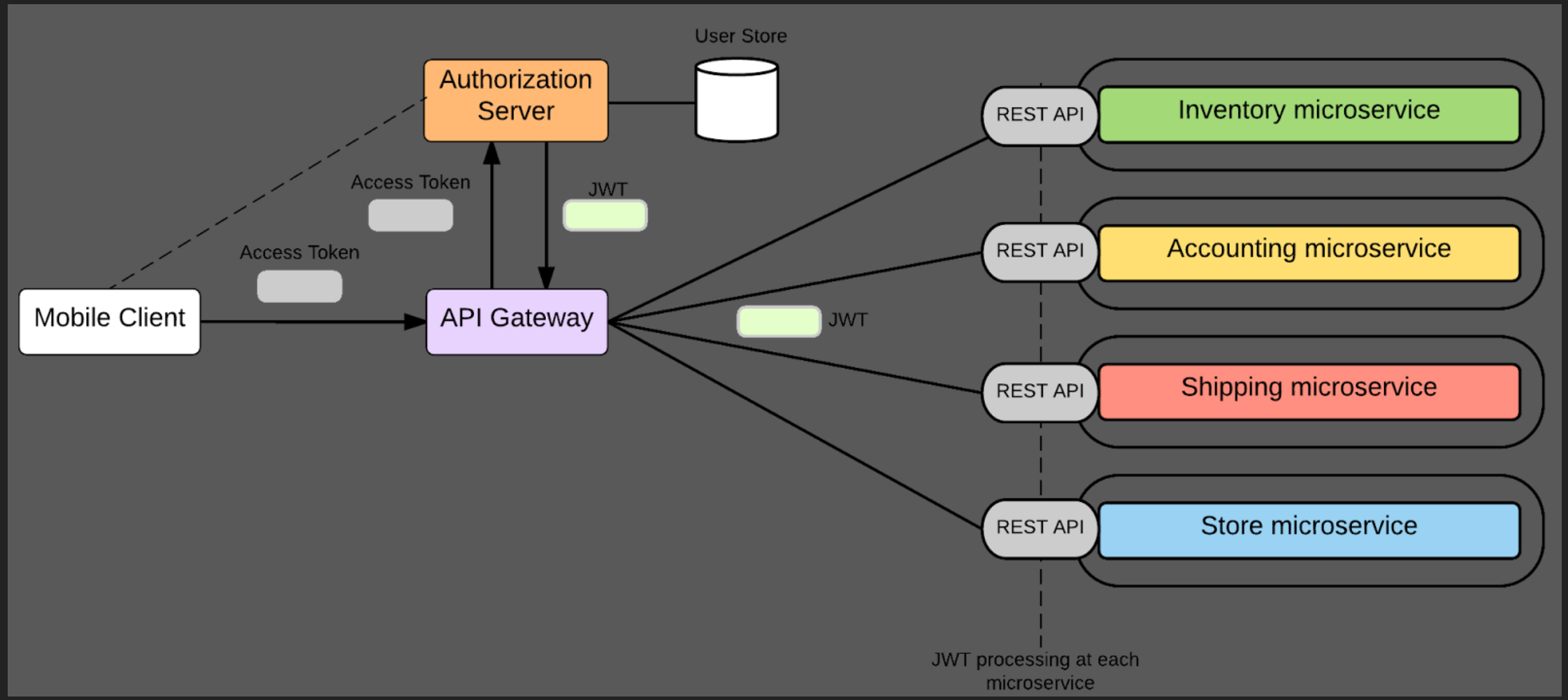
Resource server

Public key



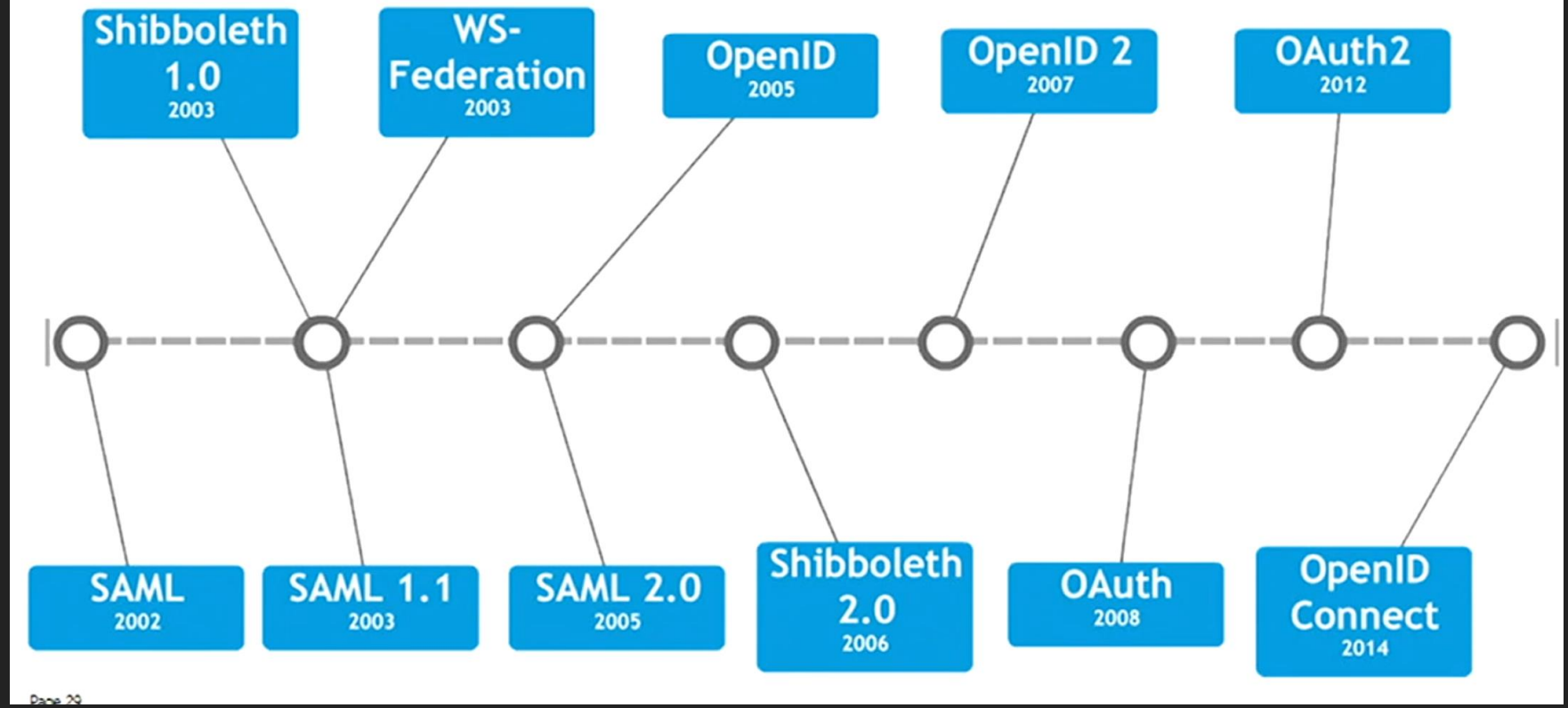
External server

Public key



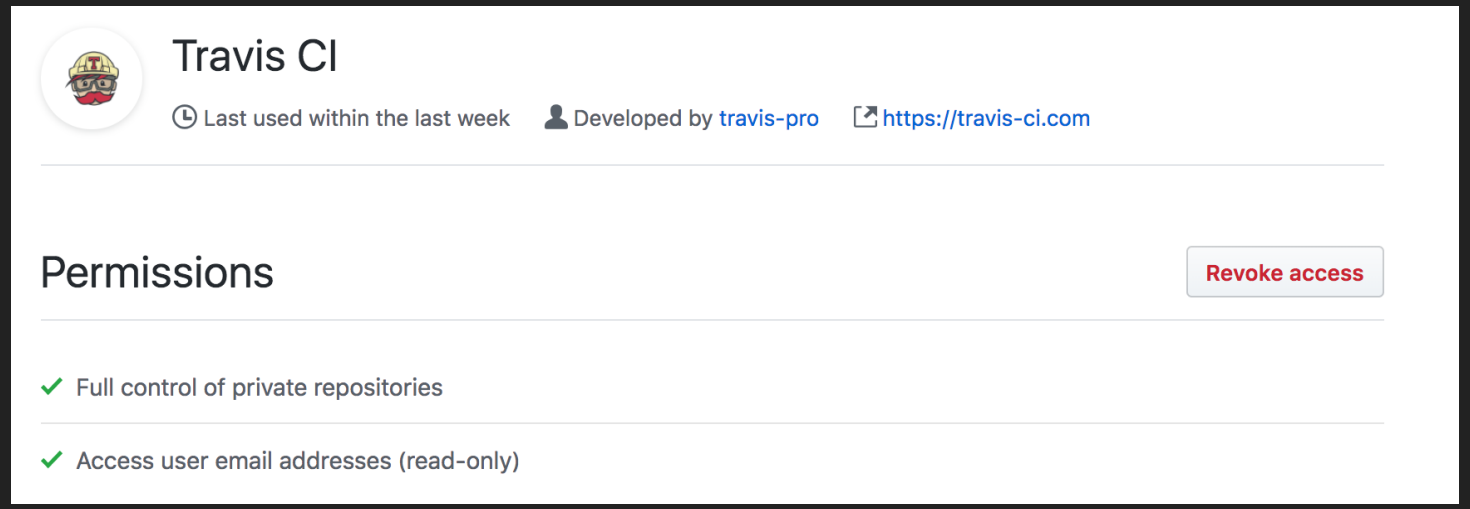
Token by reference - Contains no information outside the network
Token by value - Contains all necessary information

Authentication Protocol History



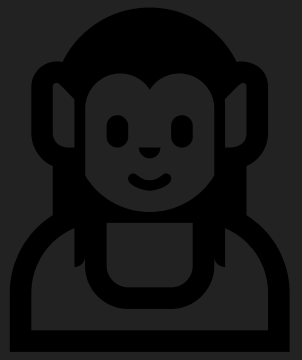
OAuth 2

- ✓ OAuth is not for Authorization
- ✓ OAuth is not for Authentication
- ✓ OAuth is a access delegation framework
  - Reduction of sharing credentals with third parts
  - Revoke of access on application level
  - Scopes
  - Acces token, refresh token

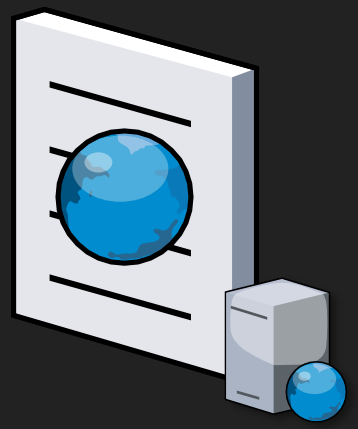


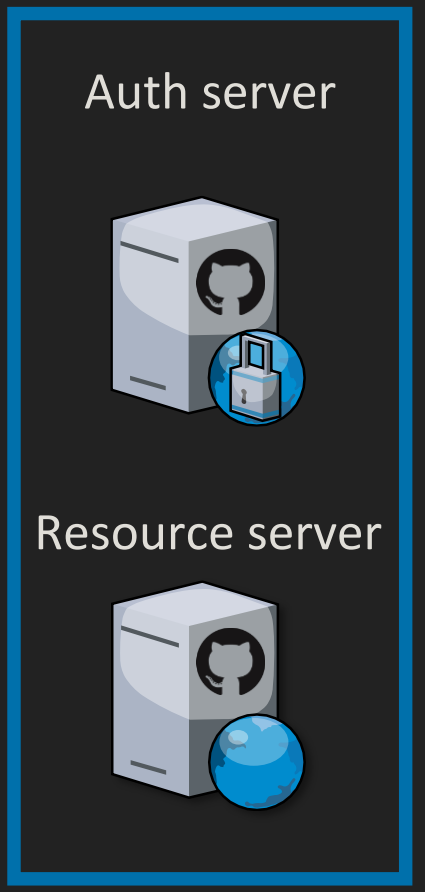
OAuth 2 - Parts

Resource Owner

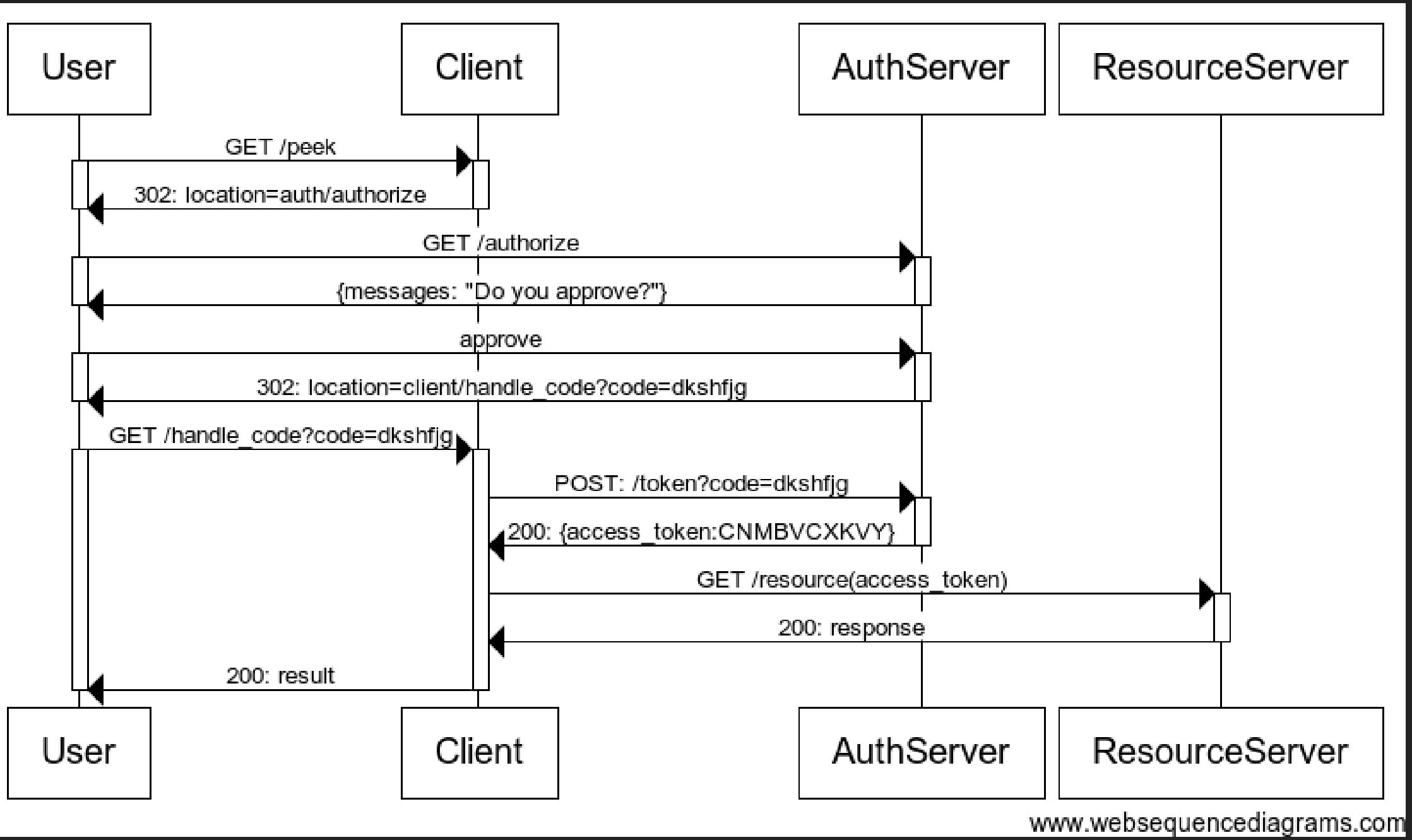


Client Applicationcv

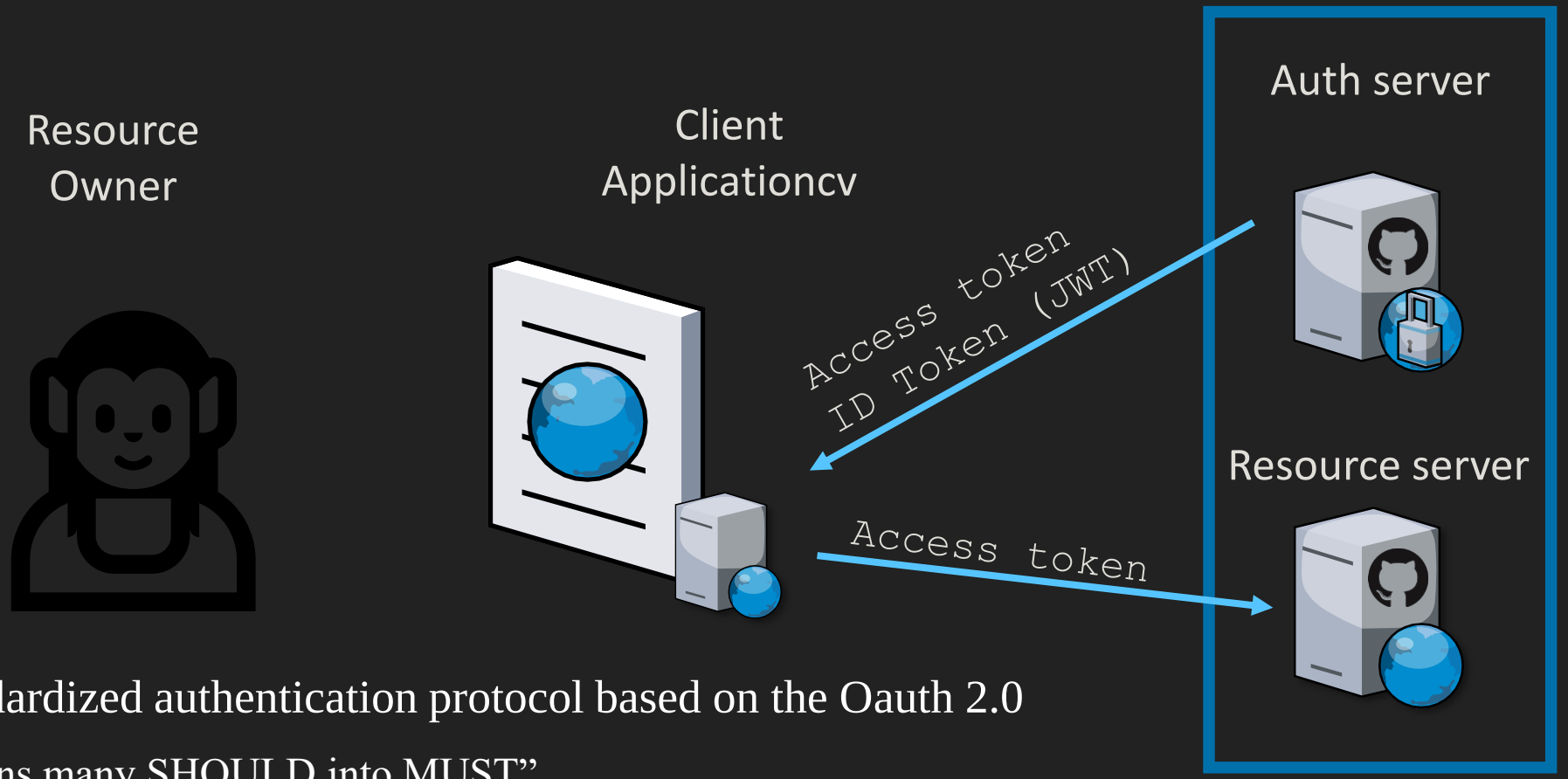




- ✓ Resource Owner - User
- ✓ Client Application - Third-party application using the API on behalf of the user.
- ✓ Resource Server - The API where the data lives
- ✓ Autorization Server - Where user credentials live


User
Client
AuthServer
ResourceServer
GET /peek
302: location=auth/authorize
GET /authorize
{messages: "Do you approve?"}
approve
302: location=client/handle_code?code=dkshfjg
GET /handle_code?code=dkshfjg
POST: /token?code=dkshfjg
200: {access_token:CNMBVCXKVY}
GET /resource(access_token)
200: response
200: result
User
Client
AuthServer
ResourceServer
www.websequencediagrams.com

OpenID Connect (OIDC)



- ✓ A standardized authentication protocol based on the Oauth 2.0
  - ”Turns many SHOULD into MUST”
- ✓ Also get an ID Token (JWT) with info about the identity of the end-user
  - When you want to know more about the user, customize the user experience

