Web Application Security

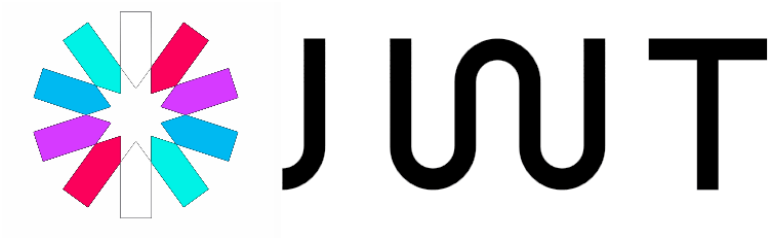
Outline

1. Token based security (JWT)
2. Authorization for Node.js & React
3. Delegated Authentication (OpenID Connect)
4. Delegated Authorization (OAuth2)

Web Security Aspects

- **Authentication (Identity verification)**:
  - Verify the identity of the user given the credentials received
  - Making sure the user is who he claims to be
- **Authorization**:
  - Determine if the user should be granted access to a particular resource.
- **Confidentiality**:
  - Encrypt sensitive data to prevent unauthorized access in transit or in storage
- **Data Integrality**:
  - Sign sensitive data to prevent the content from being tampered (e.g., changed in transit)

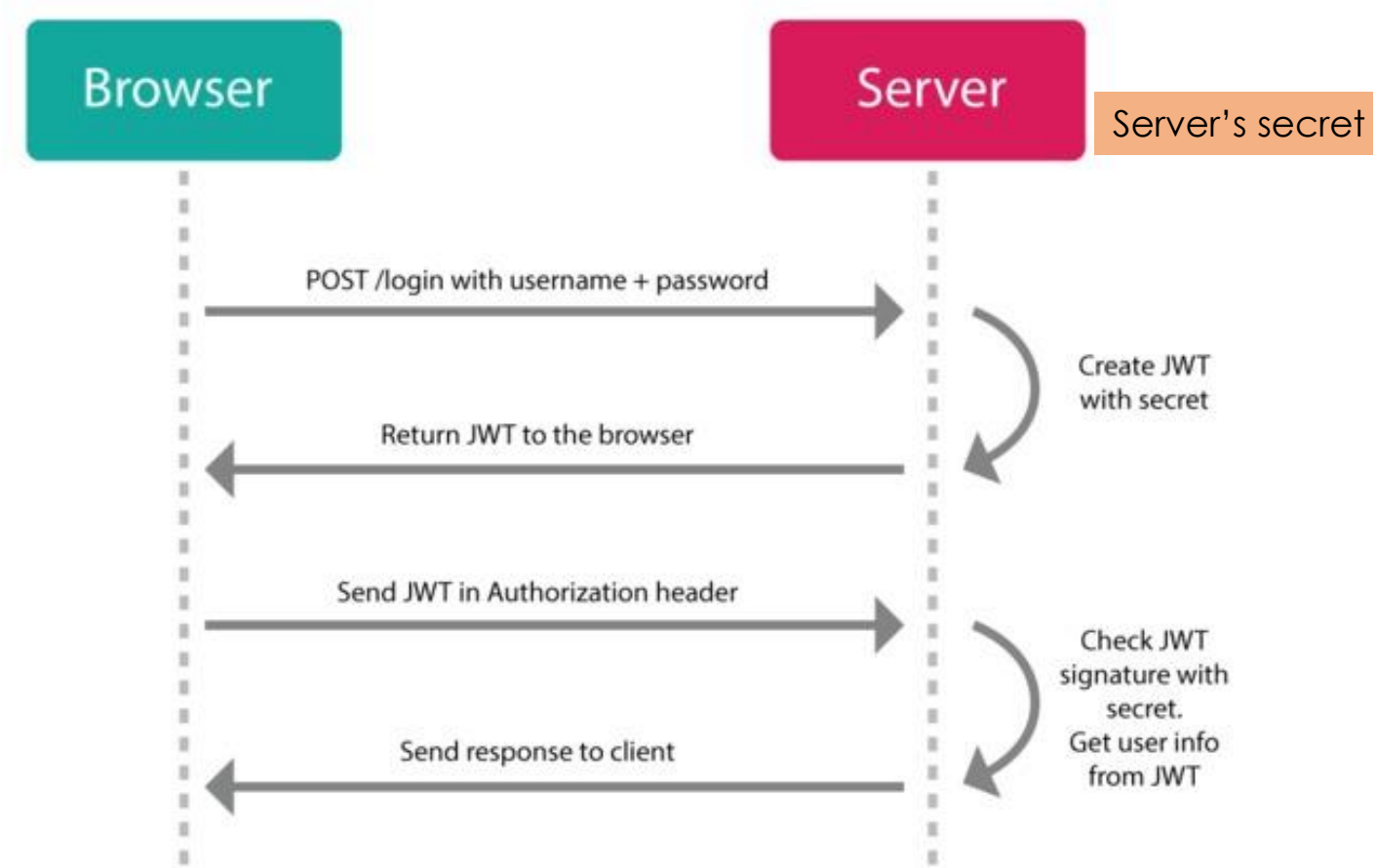
Token based security



Token based security

- After a successful authentication a **JSON Web Token (JWT)** is issued by the server and communicated to the client
- JWT token is a signed json object
  - contain information about issuer and subject (claims)
  - signed (tamper proof & authenticity)
  - typically contain an expiration time
- JWT is added to the HTTP header of subsequent requests to Web API
- Web API (i.e., a resource) validates a token

JSON Web Token (JWT)



- Every request to a Web API must include a **JW**
- Web API checks that the JWT token is valid
- Web API uses info in the token (e.g., **role**) to make authorization decisions

Token based security advantages

- A primary reason for using token-based authentication is that it is **stateless** and **scalable** authentication mechanism
  - It is suitable for SPA, Web APIs, and mobile apps
  - The token is stored on the client-side
  - The claims in a JWT are encoded as a **JSON** object that contains information that is useful for making authorization decisions
  - JWT is a simple and widely useful security token format with libraries available in most programming languages
- Can be used for **Single Sign-On:**
  - Sharing the JWT between different applications

JWT Structure

Header

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

Claims

```
{
    role: "Admin",
    given_name: "Abdelkarim",
    family_name: "Erradi",
    name: "erradi",
    email: "erradi@jwt.org",
    iat: 1526597430,
    exp: 1526604630
}
```

eyJhbGciOiJub25lIn0.eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMD.4MTkzODAsDQogImh0dHA6Ly9leGFt

Header **Claims** **Signature**

Sign-Up Example

- Sign up @ http://localhost:3040/auth/signup

Try it with Postman

POST http://localhost:3040/auth/signup

Authorization | Headers (1) | Body | Pre-request Script | Tests

form-data | x-www-form-urlencoded | raw | binary | JSON (application/json)

```
{
    "given_name": "Abdelkarim",
    "family_name": "Erradi",
    "name": "erradi",
    "email": "erradi@jwt.org",
    "password":"secret"
}
```

Body | Cookies | Headers (7) | Test Results

Pretty | Raw | Preview | JSON

```
{
    "success": "New user has been created"
}
```

Successful Login using JWT

- Sign in @ http://localhost:3040/auth/login

POST http://localhost:3040/auth/login

Authorization | Headers (1) | Body | Pre-request Script | Tests

form-data | x-www-form-urlencoded | raw | binary | JSON (application/json)

```
1 {
2     "name": "erradi",
3     "password":"secret"
4 }
```

Body | Cookies | Headers (7) | Test Results

Pretty | Raw | Preview | JSON

```
1 {
2     "idToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJvaWRQcm92aWRlciI6ImxvY2Fs
          oiZXJyYWRpIiwiZW1haWwiOiJlcnJhZGlAand0Lm9yZyIsIl9fdiI6MCwiaWF0IjoxNTI2NTk
3 }
```

Use JWT to Access Protected Resource

- Get users http://localhost:3040/auth/users

GET http://localhost:3040/auth/users

Authorization | Headers (1) | Body | Pre-request Script | Tests

Key	Value
Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJvaWRQcm92aWRlciI6ImxvY2FsIiwi
New key	Valu

Body | Cookies | Headers (7) | Test R

Pretty | Raw | Preview | JSON


```
[
    {
        "oidProvider": "local",
        "role": "Visitor",
        "defaultUrl": "/",
        "_id": "5afe02cbac0f4c4260622e46",
        "given_name": "Abdelkarim",
        "family_name": "Erradi",
        "name": "erradi",
        "email": "erradi@jwt.org",
        "password": "$2b$10$I/DbUjlOeja.dMnekCLZTOqnBi4xyI9zPCsE3e48xn5Dwpcs45NWe"
    }
]
```


Add the JWT token to standard **Authorization** header of HTTP requests to allow the Web API to verify it and allow access to resources

Storing JWT in Browser Local Storage

Local Storage allows storing a set of name value pairs directly accessible with **client-side** JavaScript

- **Store**

```
localStorage.idToken = "eyJhbnR5cCI…."
```

- **Retrieve**

```
Console.log(localStorage.idToken)
```

- **Remove**

```
delete localStorage.idToken
```

- **Remove all saved data**

```
localStorage.clear();
```

https://jwtinspector.io/

JWT Inspector is a chrome extension that lets you **decode** and **inspect** JWT in requests, and local storage

Overview (0) | Debug

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJvaWRQcm92aWRlciI6ImxvY2FsIiwicm9sZSI6IlZpc2l0b3IiLCJkZWZhdWx0VXJsIjoiLyIsIl9pZCI6IjVhZmUwMmNiYWMwZjRjNDI2MDYyMmU0NiIsImdpdmVuX25hbWUiOiJBYmRlbGthcmltIiwiZmFtaWx5X25hbWUiOiJFcnJhZGkiLCJuYW1lIjoiZXJyYWRpIiwiZW1haWwiOiJlcnJhZGlAand0Lm9yZyIsIl9fdiI6MCwiaWF0IjoxNTI2NTk2NDQ1LCJleHAiOjE1MjY2MDM2NDV9.fwG_o7zbvdEIRnNifQ5Bj8sZ5Q4VxtaC5c6VPbxaDLc

Header


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


Payload


```
{
  oidProvider: "local",
  role: "Visitor",
  defaultUrl: "/",
  _id: "5afe02cbac0f4c4260622e46",
  given_name: "Abdelkarim",
  family_name: "Erradi",
  name: "erradi",
  email: "erradi@jwt.org",
  __v: 0,
  iat: 1526596445,
  exp: 1526603645
}
```


Signature

fwG_o7zbvdEIRnNifQ5Bj8sZ5Q4VxtaC5c6VPbxaDLc

401 vs. 403

- ***401 Unauthorized***

- Should be returned in case of failed authentication

- ***403 Forbidden***

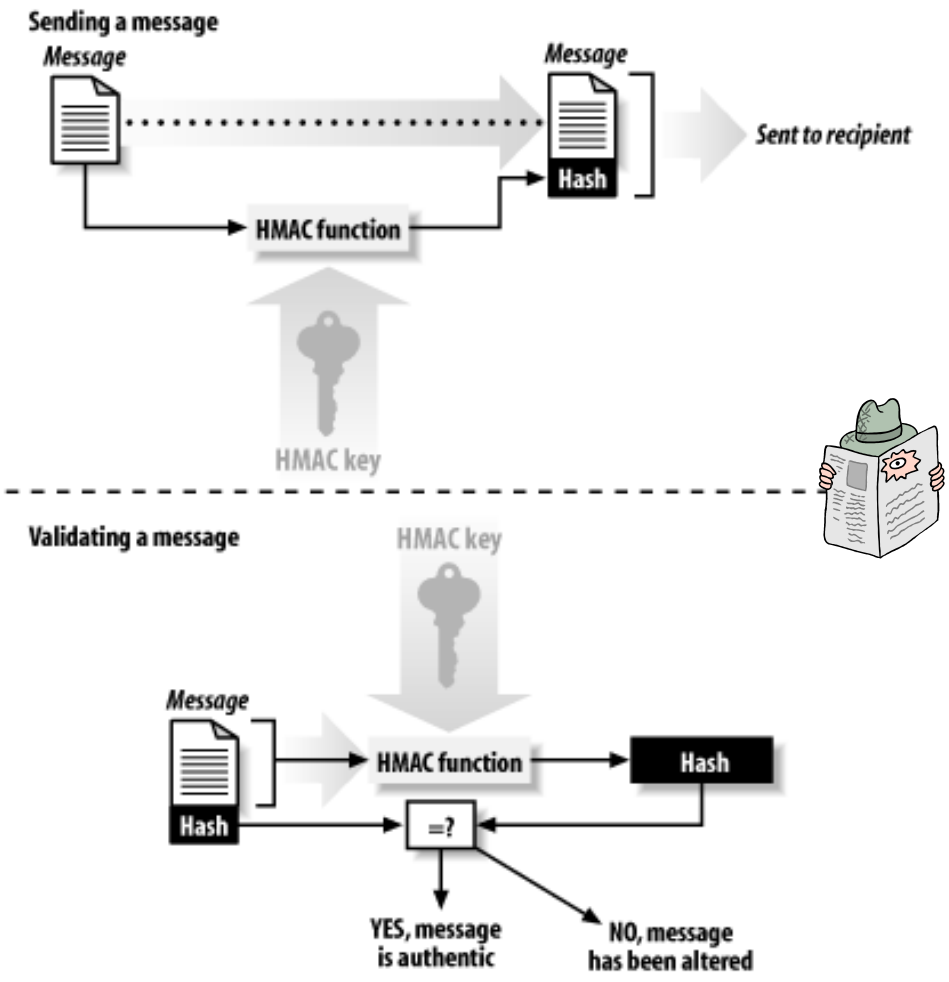
- Should be returned in case of failed authorization

- The user is authenticated but not authorized to perform the requested operation on the given resource

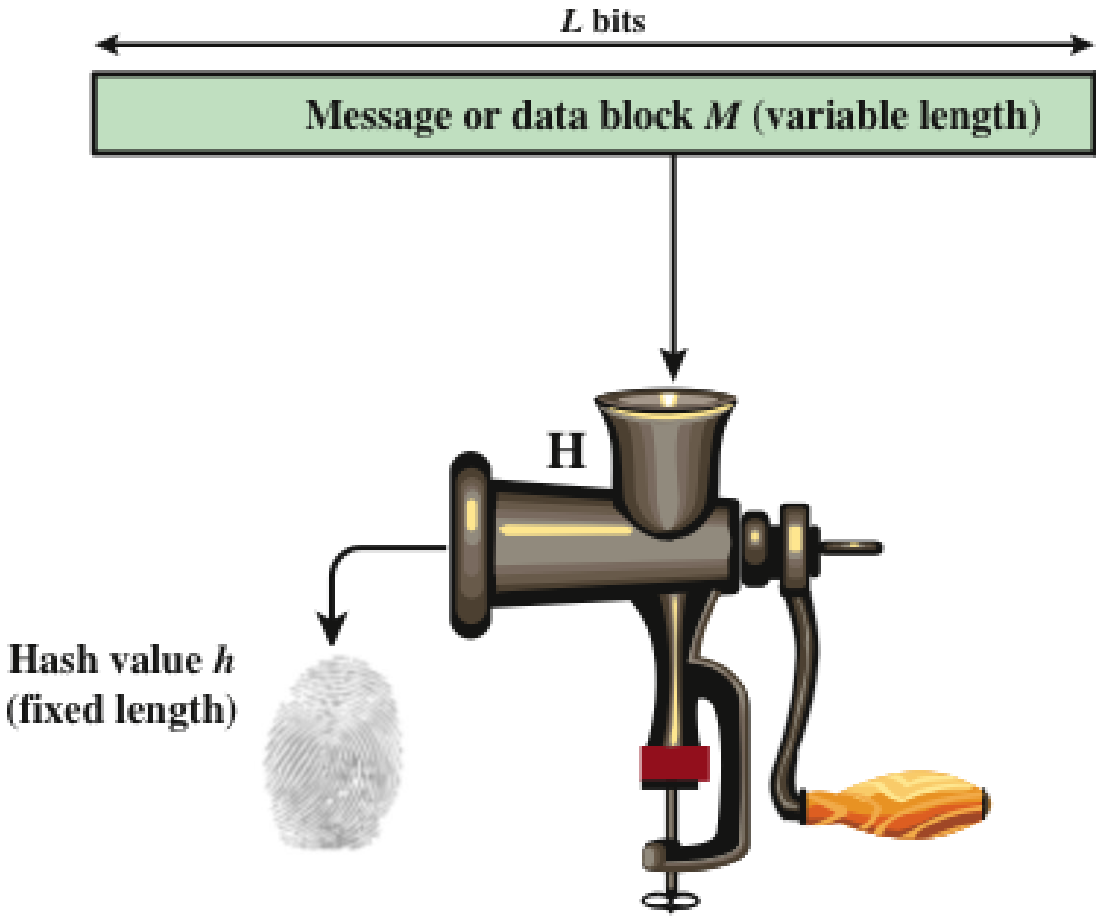
Hash-based Message Authentication Code (HMAC)

- **HMAC-SHA256** is often used for **signing JWT** to ensure its integrity
- HMAC-SHA256 is a *cryptographic hash function* that uses SHA256 hashing and a *secret key to generate a MAC (i.e., JWT signature)*
- The MAC is appended to the message sent
- MAC provides **message integrity**: Any manipulations of the message during transit will be detected by the receiver

(An attacker who alters the message will be unable to alter the associated MAC value without knowledge of the secret key)



Hashing



Hash functions are used to compute a digest of a message. Its takes a variable size input, produce fixed size pseudorandom output

Authorization for Node.js & React

Node.js Middleware to Check Authorization

- Use route middleware function to check if the user is authenticated and authorized before handling their request

```
async function isAuthenticated(req, res, next) {
    let idToken = req.headers.authorization;
    try {
        if (idToken) {
            idToken = idToken.split(" ")[1];
            //Decode and verify jwt token using the secret key
            const decodedToken = await jwt.verify(idToken, keys.jwt.secret);
            req.user = decodedToken;
            next();
        }
        else {
            res.status(401).json({error: "Unauthorized. Missing JWT Token"});
        }
    } catch (error) {
        res.status(403).json({error});
    }
}

router.get('/users', isAuthenticated, async function (req, res) {
    if (req.user.role == 'Admin') {
        const users = await userRepository.getUsers();
        res.json(users);
    } else {
        res.status(403).json({ error: "Access denied" });
    }
});
```

React Protected Routes

- For protected React routes, we need to use a **Custom Route** function to check if the user

(1) is authenticated

(2) is authorized to access a particular route based on the user's role

- See ***ProtectedRoute.js*** and ***App.js*** example

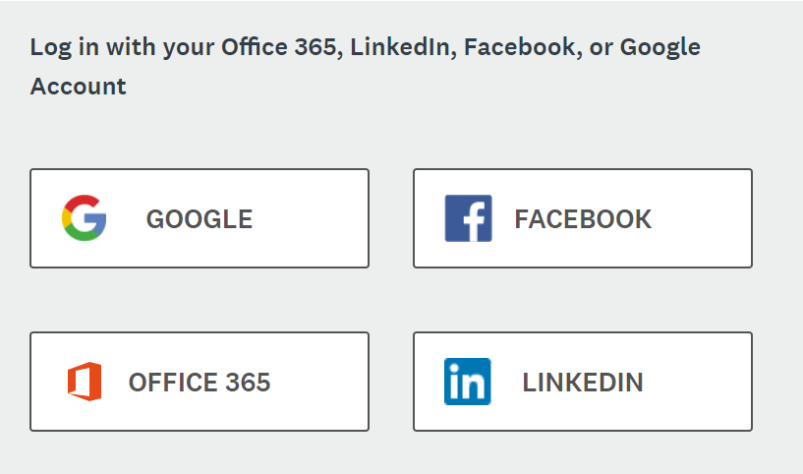
Delegated Authentication

Authentication is hard

- Trying to write your own login system is difficult:
  - Need to save passwords securely
  - Provide recovery of forgotten passwords
  - Make sure users set a good password
  - Detect logins from suspecious locations or new devices
  - etc.
- Luckily, **you don't have to build your own authentication!**
- You can use **OpenID Connect** to delegate login to an **Identity Provider** and get the user's profile

OpenID Connect

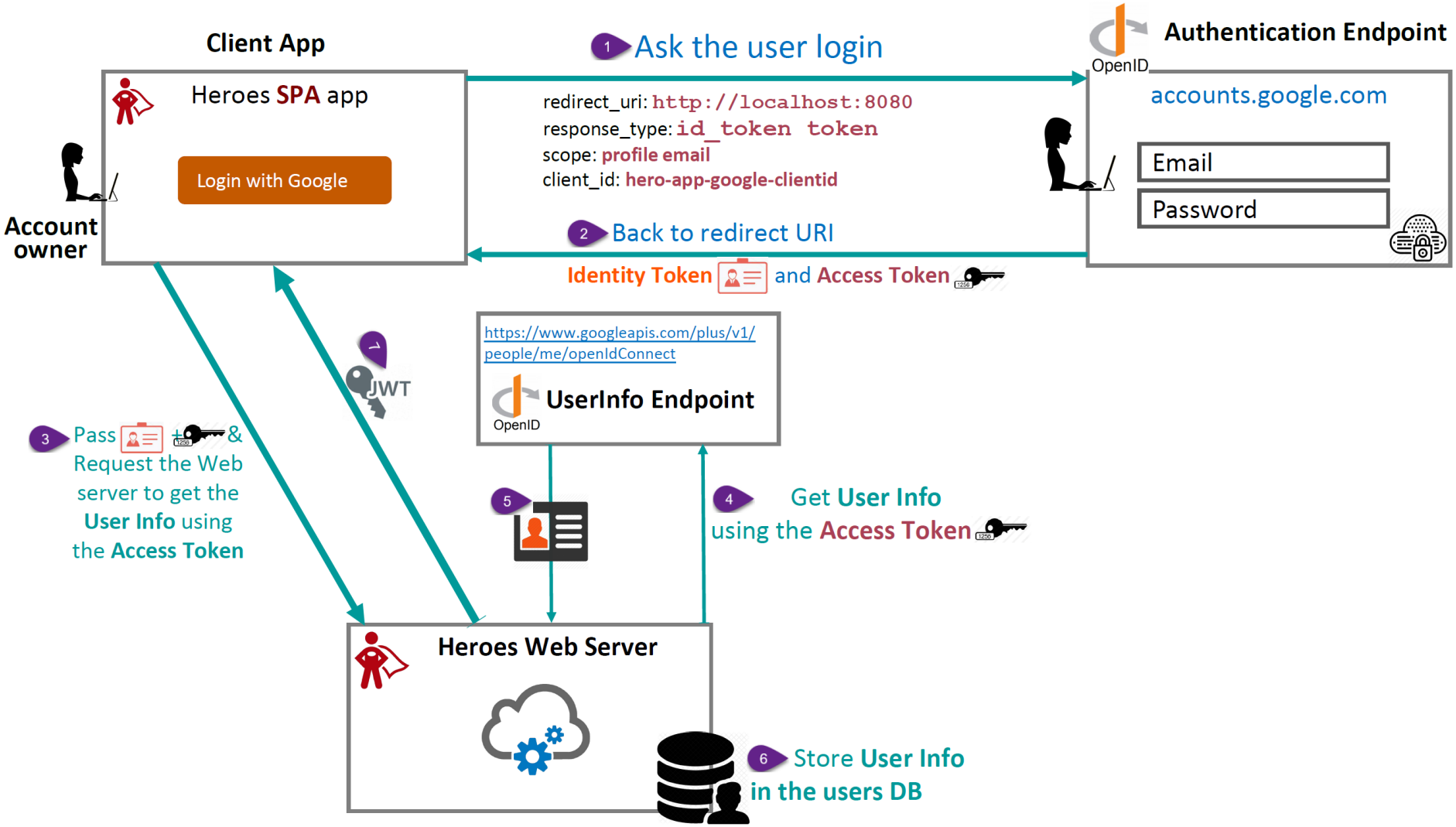
- **OpenID Connect** is a standard for user authentication
  - For users:
    - It allows a user to log into a website like AirBnB via some other service, like Google or Facebook
  - For developers:
    - It lets you authenticate a user without having to implement log in
  - Examples: "Log in with Facebook"



OpenID Connect APIs

- Companies like Google, Facebook, Twitter, and GitHub offer OpenID Connect APIs:
  - Google Sign-in API
  - Facebook Login API
  - Twitter Login API
  - GitHub Apps/Integrations
  - OpenID Connect is standardized, but the API that these services provide are slightly different
  - You must read the documentation to understand how to connect via their API
- After the user logins, you will get the user profile such name, email, etc.

OpenID Connect Authentication Flow



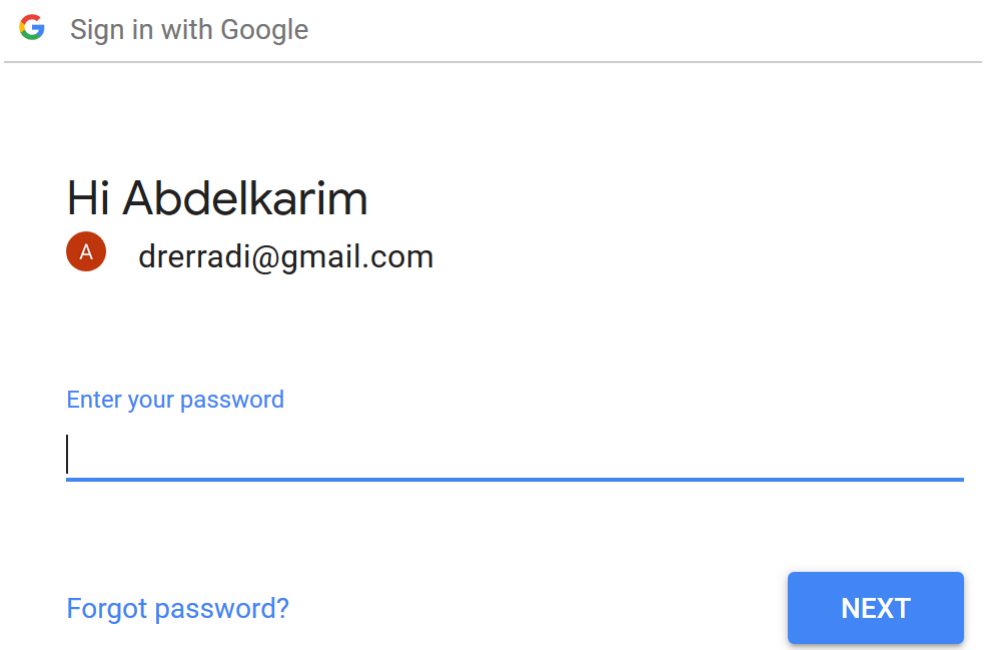
Authenticating via a SPA App

- **User** starts the flow by visiting a SPA App
- **Client** sends authentication request with `profile` scope via browser redirect to the **Authorization** endpoint
- **User** authenticates and consents to **Client** to access user's identity
- **ID Token** and **Access Token** is returned to **Client** via browser redirect
- **Client** optionally fetches additional user info with the **Access Token** from **UserInfo** endpoint

Authorization Request

- Ask the user to login via browser redirect to the Authentication Endpoint

https://accounts.google.com/o/oauth2/auth



- This will return an **Access Token** to the client to allow it to request the user's profile from the UserInfo Endpoint

Authentication Parameters

GET https://accounts.google.com/o/oauth2/auth?scope=profile%20email%20phone&client_... Params Send

Key	Value
scope	profile%20email%20phone
client_id	866457396346-piq09ek9kiofq9uspsnjuv...
response_type	id_token%20token
redirect_uri	http://localhost:8080

Scope = what user info the client needs access to?

Need to register and get **client_id** from https://console.developers.google.com/apis/credentials

What is the desired response type to get from the Authentication EndPoint?

- **id_token**: jwt of the authentication user
- **token**: access-token to be able to access the UserInfo endpoint

Redirect_udrl = callback address google will use to deliver to access_token and id_token

Body Cookies (2) Headers (15) Test Results Status: 200 OK Time: 461 ms Size: 72 KB

Google

One account. All of Google.

Sign in with your Google Account

Email or phone

Next

ID Token

- JWT representing logged-in user
- Claims:
  - `iss` - Issuer
  - `sub` - User Identifier
  - `aud` - Audience for ID Token
  - `exp` - Expiration time
  - `iat` - Time token was issued

Example ID Token from Google

```
{
  iss: "accounts.google.com",
  aud: "lv1muk.apps.googleusercontent.com",
  sub: "111893194175723488203",
  email: "karimerradi@gmail.com",
  email_verified: true,
  exp: 1526656174,
  iat: 1526652574
}
```

Scopes for Identify Claim Requests

- Scopes = what user info you need access to?
- Standard scopes:

  `openid` – JWT representing logged-in user

  `profile` – Profile info

  `email` – Email address & verification status

  `address` – Postal address

  `phone` – Phone number & verification status

Calling the UserInfo Endpoint

- Get the user's profile from the UserInfo Endpoint

GET https://www.googleapis.com/plus/v1/people/me/openIdConnect Params Send

Authorization Headers (1) Body Pre-request Script Tests

Key	Value	Description
Authorization	Bearer ya29.Gly_BcvwEQdaZgHKJIk2nB7N2g3falZmpCxp3NXM7UjoWxou_1Jp4v...	
New key	Value	Description

Body Cookies Headers (14) Test Results Status: 200 OK Time: 663 ms

Pretty Raw Preview JSON


```
{
    "kind": "plus#personOpenIdConnect",
    "gender": "male",
    "sub": "111893194175723488203",
    "name": "Erradi",
    "given_name": "Erradi",
    "family_name": "",
    "profile": "https://plus.google.com/111893194175723488203",
    "picture": "https://lh6.googleusercontent.com/-iuZD8qYF0xQ/AAAAAAAAAAI/AAAAAAAAGIM/ll35MtiUkJ8/photo.jpg?sz=50",
    "email": "karimerradi@gmail.com",
    "email_verified": "true",
    "locale": "en"
}
```


Send the **access token** received after the authentication. Add it to the **Authorization** header.

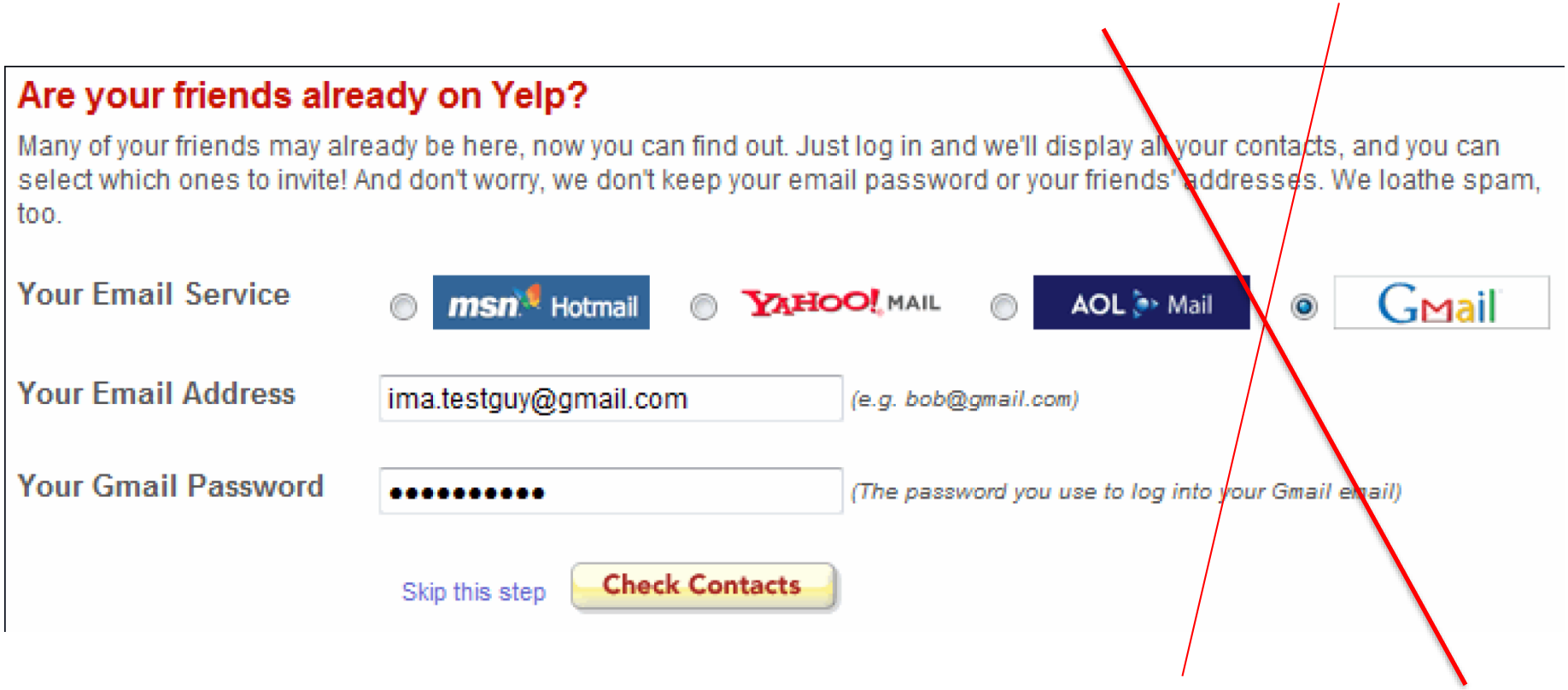
UserInfo Claims

- sub
- name
- given_name
- family_name
- middle_name
- nickname
- preferred_username
- profile
- picture
- website
- gender
- birthdate
- locale
- zoneinfo
- updated_at
- email
- email_verified
- phone_number
- phone_number_verified
- address

Delegated Authorization

The delegated authorization problem

- How can I let a Web/Mobile App access my Data without giving it my password?
- Don't do it this way!



Hotel Key Cards, but for Apps

OAuth Authorization Server

Access Token

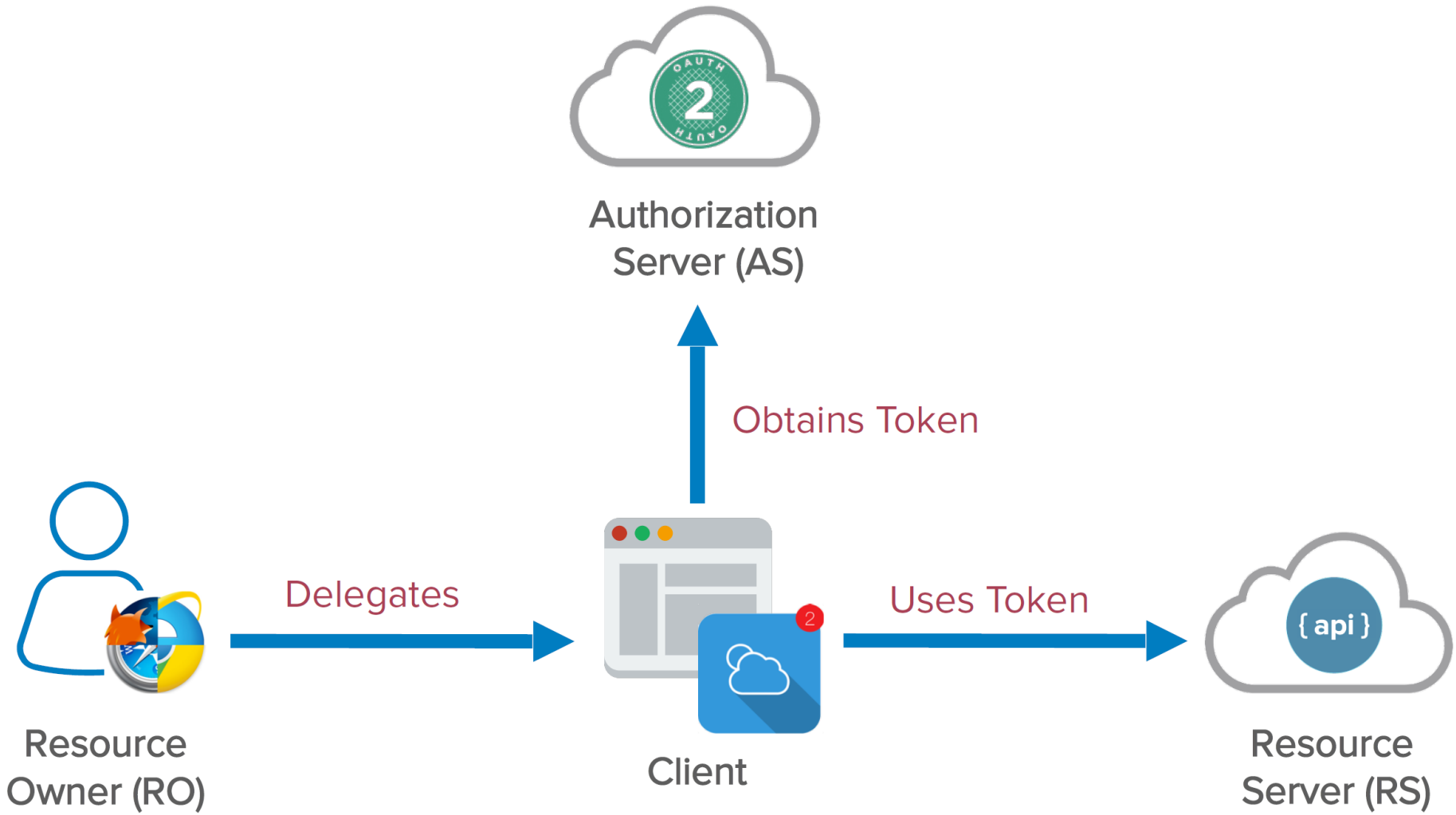
Resource (API)

OAuth 2.0

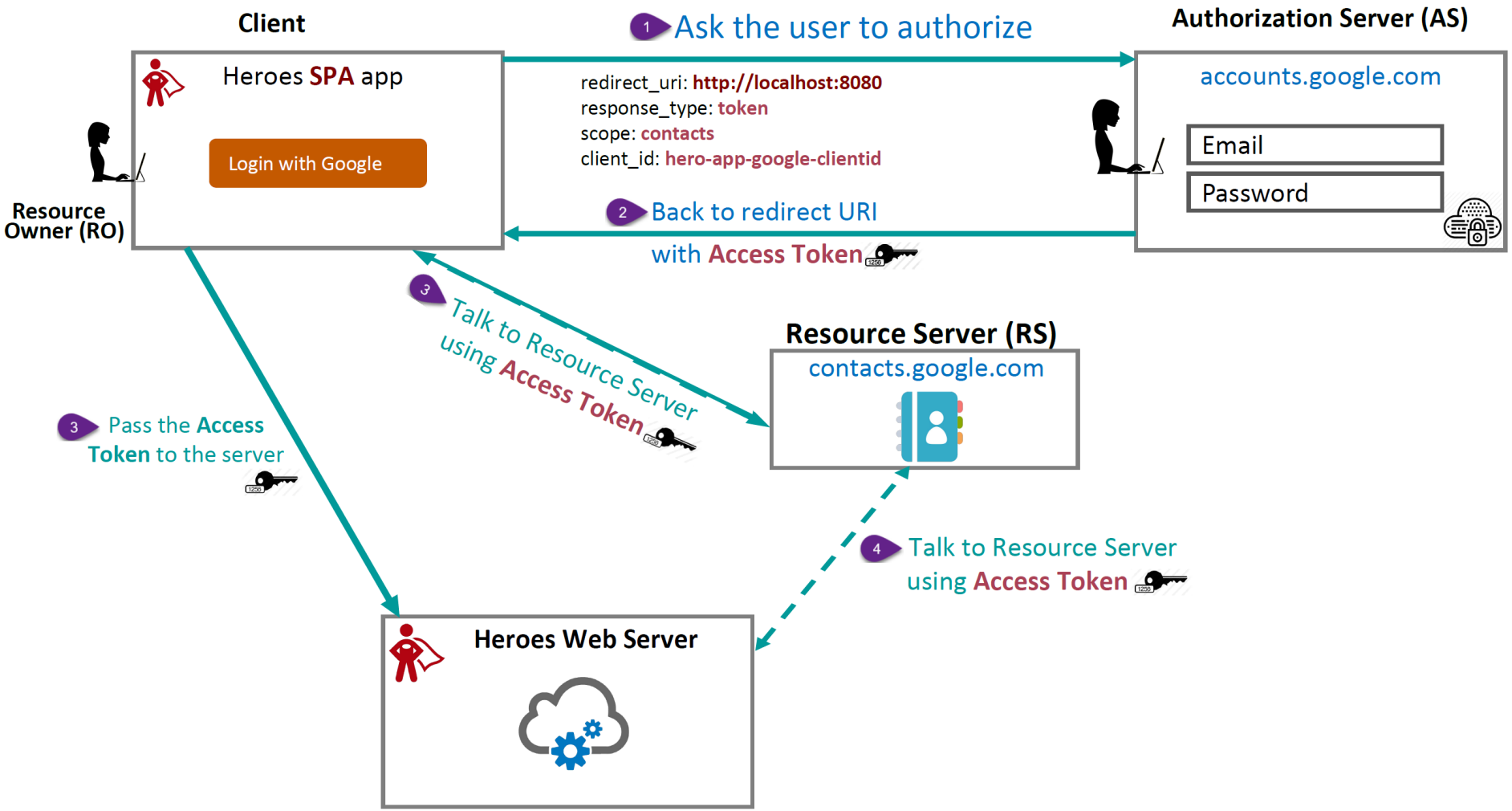
- OAuth is used for Delegation of Authorization (i.e., Access Granting Protocol)
  - App gets the permission to access data on the user's behalf

1. **App** requests authorization from **User**
2. **User** authorizes **App** and delivers proof
3. **App** presents proof of authorization to server to get a **Token**
4. **Token** is restricted to only access what the **User** authorized for the specific **App**

OAuth 2.0 Actors

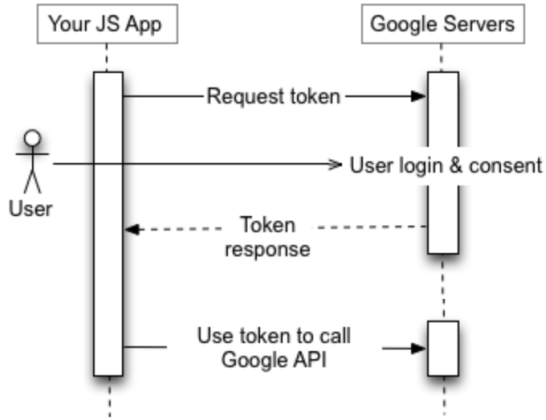


OAuth 2.0 Authorization Flow



OAuth 2.0 terminology

- Protected resource
  - Data to be protected by OAuth
- Resource owner = User
  - User granting access to protected resource
- Resource server
  - Server hosting protected resources accessible by access tokens
- Client = App
  - Application accessing protected resources based on resource owner authorization
- Authorization server
  - Server issuing acccess tokens to client based on authenticated resource owner and its authorization
- Tokens
  - Used instead of user credentials to access protected resources



Google OpenId Connect

- To access google API first register and create a project @ https://console.developers.google.com/apis
- Get the clientId and clientSecret @

https://console.developers.google.com/apis/credentials

- Before accessing any API you must enable it on your Google project e.g.,

https://console.developers.google.com/apis/library/people.googleapis.com/?project=**qu-oauth-demo**

The steps are very similar for other services such as Twitter and Microsoft

First ask the user to Authorize

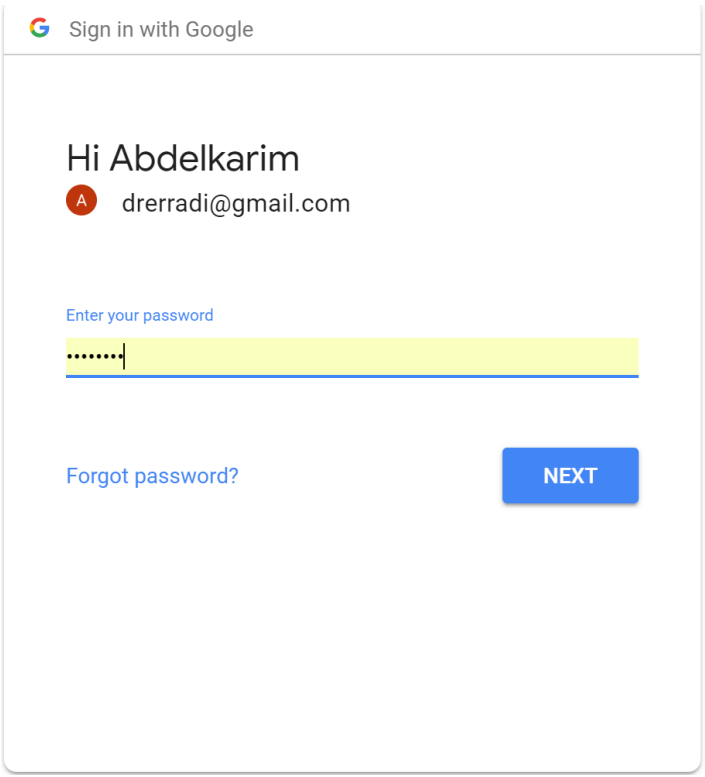
```
https://accounts.google.com/o/oauth2/v2/auth?
client_id=abc123&
redirect_uri=http://localhost/heroes/callback&
scope=contacts&
response_type=token
```

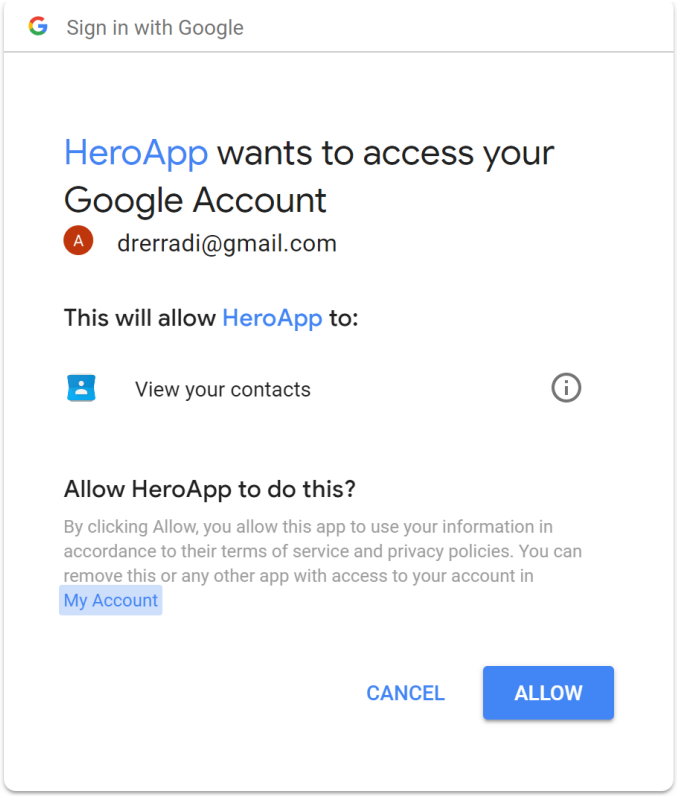
User Authenticates and Grants Authorization,
This will return an **Access Token** to the client to allow it to access the protect resource

```
http://localhost/heroes/callback?
access_token=oMsCeLvIaQm6bTrgtp7
```

User Authenticates and Grants Authorization

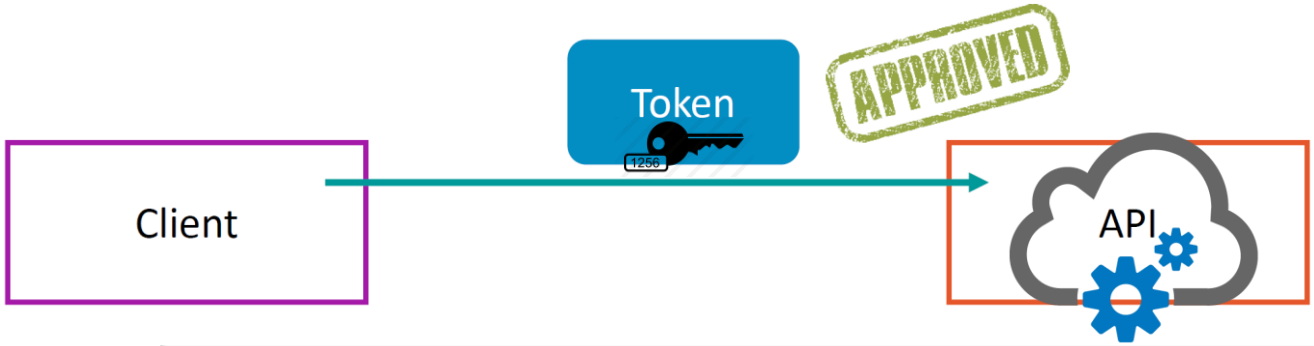
Authenticate & Grant Authorization





Note that users can revoke the permission @ https://myaccount.google.com/permissions

Use the Access Token to access the Resource



- Validate token
- Grant access to the resource

GET https://people.googleapis.com/v1/people/me/connections?personFields=names,emailAddresses,phoneNum... Params Send

Key	Value	Description
personFields	names,emailAddresses,phoneNumbers,addresses,pho...	

Authorization Headers (1) Body Pre-request Script Tests

Key	Value	Description
Authorization	Bearer ya29.GlvABUS5jPiHTOywJTpKr6cCwUEsACshu3...	

Body Cookies (1) Headers (13) Test Results Status: 200 OK Time: 253 ms

Pretty Raw Preview JSON


```
{
    "connections": [
        {
            "resourceName": "people/c3945308633452445077",
            "etag": "%EggBAgMJCxA3LhoMAQIDBAUGBwgJCgsMIgxuWmJUQU5BT3FKOD0=",
            "names": [
                {
                    "metadata": {
                        "primary": true,
                        "source": {
                            "type": "CONTACT",
                            "id": "36c08d4c8a36fd95"
                        }
                    },
                    "displayName": "Qatar University",
                    "familyName": "University",
                    "givenName": "Qatar",
                    "displayNameLastFirst": "University, Qatar"
                }
            ],
```

Summary

- JWT is easy to create, transmit and validate to protect Web API in a scalable way
- Use OpenID Connect for **Authentication** scenarios to:
  - Log in users
  - Making your accounts available in other systems
- Use OAuth 2.0 for **Authorization** scenarios to:
  - Grant access to Web API
  - Get access to user data in other systems

Resources

- JWT Handbook

https://auth0.com/resources/ebooks/jwt-handbook

- Authentication Survival Guide

https://auth0.com/resources/ebooks/authentication-survival-guide

- OAuth 2

https://www.oauth.com/

- Good resource to learn about JWT

https://jwt.io/

- RBAC

https://www.npmjs.com/package/easy-rbac

OAuth 2 and OpenID Connect Videos

- OAuth 2.0 and OpenID Connect (in plain English)

https://www.youtube.com/watch?v=996OiexHze0

- What the Heck is OpenID Connect?

https://www.youtube.com/watch?v=6ypYXxRPKgk

- How Google is using OAuth?

https://www.youtube.com/watch?v=fxRXLbgX53A