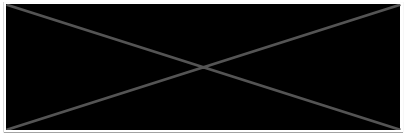
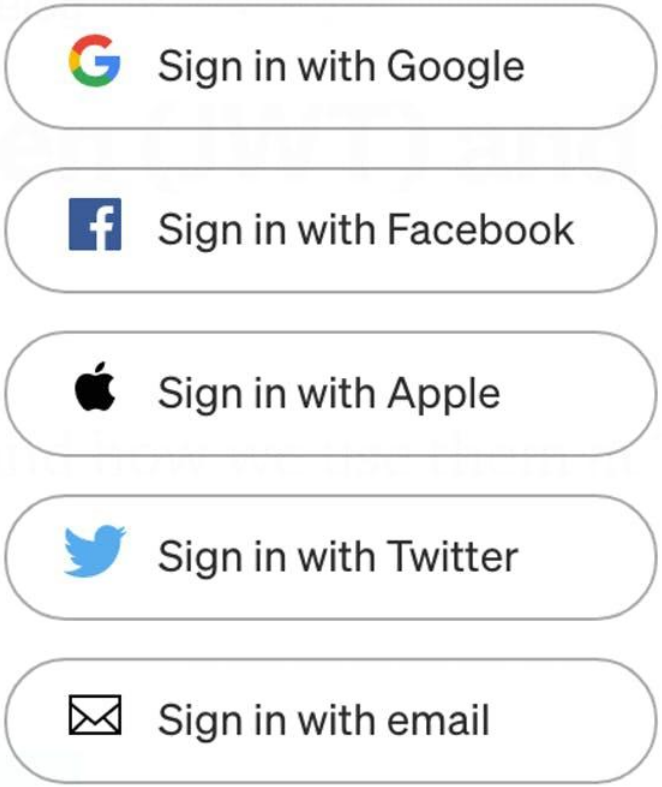
Programming 6

Security



Question


Sign in with Google
Sign in with Facebook
Sign in with Apple
Sign in with Twitter
Sign in with email

Ubiquitous language security

- https://openid.net/connect/

OpenID Connect (OIDC) is a identity layer on top of OAuth 2.0 protocol

What does it do?

*It lets app and site developers **authenticate** users **without** taking on the **responsibility** of storing and managing passwords in the face of an Internet that is **well-populated with people trying to compromise your users' accounts** for their own gain.*

Ubiquitous language

- Authentication & Authorization
- **IDP** - Identity Provider - A party that offers authentication as a service
- **RP** - Relying Party - A party that outsources its user authentication to an IDP (also called SP - Service Provider)
- **Resource Owner** - An entity that grants access to protected resources (a user, microservice, device,...)
- **Resource Server** - Server that holds the resources, something the resource owner needs access to
- **Client** - an application that requests access to a resource
- **Authorization Server** - accepts credentials from the Resource Owner, checking if they are authorized to access the resource

Multiple flows

There are multiple authorization flows possible:

- For specific use cases
- Sometimes with specific tradeoffs

Flows:

- Authorization Code Flow
- Client Credentials Flow
- Resource Owner Password Flow
- Implicit Flow with Form Post
- Hybrid Flow
- Device Authorization Flow
- **Authorization Code Flow with PKCE**

OAuth2 uses JWT Tokens

JWT token: in industry pronounced as “jot”.

JWT Token

Self-contained way for securely transferring information between parties. JWT is a standard token, but not all tokens are JWT's ;-)

JWT (Json Web Token) is a stanard token, but not every token is per se a JWT-Token. (cfr Opaque tokens)

- **JOSE Header**: contains metadata about the type of token and the cryptographic algorithms used to secure its contents.
- **JWS payload** (set of claims): contains verifiable security statements, such as the identity of the user and the permissions they are allowed.
- **JWS signature**: used to validate that the token is trustworthy and has not been tampered with. When you use a JWT, you **must** check its signature before storing and using it.

Self describing:

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

-

Tokens

identity token

A token that provides identity information about the user. Part of the OpenID Connect specification.

access token

A token that can be provided as part of an HTTP request that grants access to the service being invoked on. This is part of the OpenID Connect and OAuth 2.0 specification.

refresh token

A special token that can be provided and enables a client to retrieve new access tokens without requiring the user to perform a complete new login.

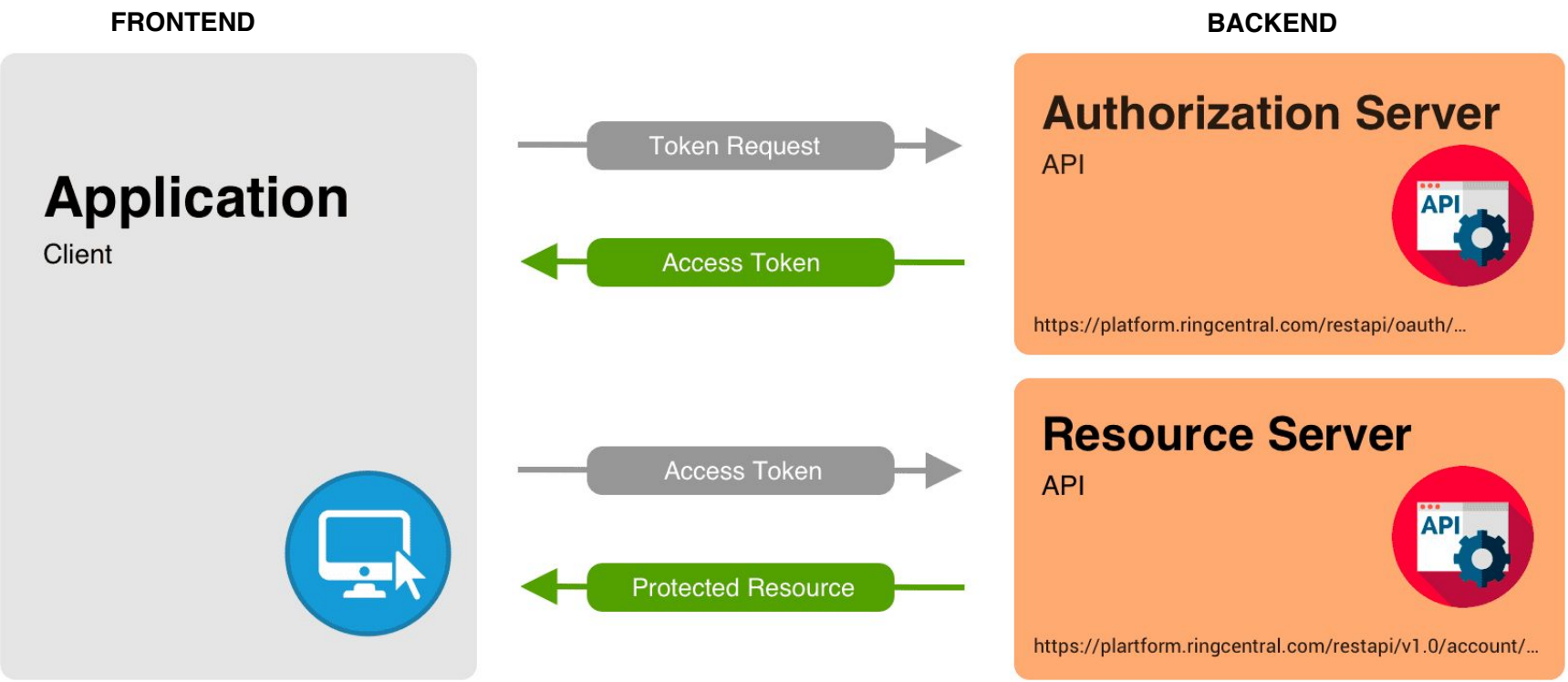
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJnT2daN0VFMVprWGwyQm1DbWl2WWpycWdq
RTk2WXBEZkFnMmVrcUtPMFo4In0.eyJleHAiOjE2OTE5MzA3NzQsImlhdCI6MTY5MTkzMDQ3NCwianRp
IjoiZjhhYWY2MmYtYmU2NC00ZjBmLTliNmYtZjIwZmVjNDI1M2YyIiwiaXNzIjoiaHR0cDovL2xvY2FsaG9
zdDo4MTgwL2F1dGgvcmVhbG1zL2RldmVsb3BtZW50IiwiYXVkIjoiYWNjb3VudCIsInN1YiI6ImQ2MzJlNz
JkLTFmOGEtNGE3ZC1hNzMzLTExZmQ4Y2YzODU2MiIsInR5cCI6IkJlYXJlciIsImF6cCI6ImJtYWNkZXYtY
XV0aCIsInNlc3Npb25fc3RhdGUiOiIyMjk4ZTM2NC1mYzllLTQzNDktOTFjMC00NmYzMmRhOWRmNzciL
CJhY3IiOiIxIiwicmVhbG1fYWNjZXNzIjp7InJvbGVzIjpbIm9mZmxpbmVfYWNjZXNzIiwiYXBpLXVzZXIiL
CJkZWZhdWx0LXJvbGVzLWJtYWMiLCJ1bWFfYXV0aG9yaXphdGlvbiJdfSwicmVzb3VyY2VfYWNjZXNzIj
p7ImFjY291bnQiOnsicm9sZXMiOlsibWFuYWdlLWFjY291bnQiLCJtYW5hZ2UtYWNjb3VudC1saW5rcyIsI
nZpZXctcHJvZmlsZSJdfX0sInNjb3BlIjoicHJvZmlsZSBlbWFpbCIsInNpZCI6IjIyOThlMzY0LWZjOWUtND
M0OS05MWMwLTQ2ZjMyZGE5ZGY3NyIsImVtYWlsX3ZlcmlmaWVkIjp0cnVlLCJuYW1lIjoiQi1NYUMgRG
V2ZWxvcGVyIiwicHJlZmVycmVkX3VzZXJuYW1lIjoiZGV2ZWxvcGVyIiwiZ2l2ZW5fbmFtZSI6IkItTWFDIi
wiZmFtaWx5X25hbWUiOiJEZXZlbG9wZXIiLCJlbWFpbCI6ImRldmVsb3BlckBibWFjLmNvbSJ9.errEKfztw
kgzYQa7w0yDGIDyAskGGfl24ATS9N-r4-BrkgX0q_OHEORfpvYhdF31CTDN9flfYMFiCImH_9qTGtMrtftU
Oe_Qy2gaBq5bmsAsIyBruhrzEgP5ScTQkdj5pD8s67G91TUhJsYepVCYM8GRnRJztu3ZwJw-mWcvbFYz
UernrylhzovJV11QO5PUIbJKNsIqMHSC3WkFytqH7eAnwkRyUitSZLndYmcQzoETWKKu7j-gmlb8Nr6o_
NoquGRYt91wS3L-XT89Ox7CsuIT_prwLXxwWzTE1Icvt1EsY6QaFraMPxI4PbXhIgiTQ6X0TkcWSnSYD1
HFjMhPXw

https://jwt.io/

Different flows

Resource Owner Password Flow

This solution is not likeable he said, but the easiest one to use.
It is easy because one call is used. password grant.

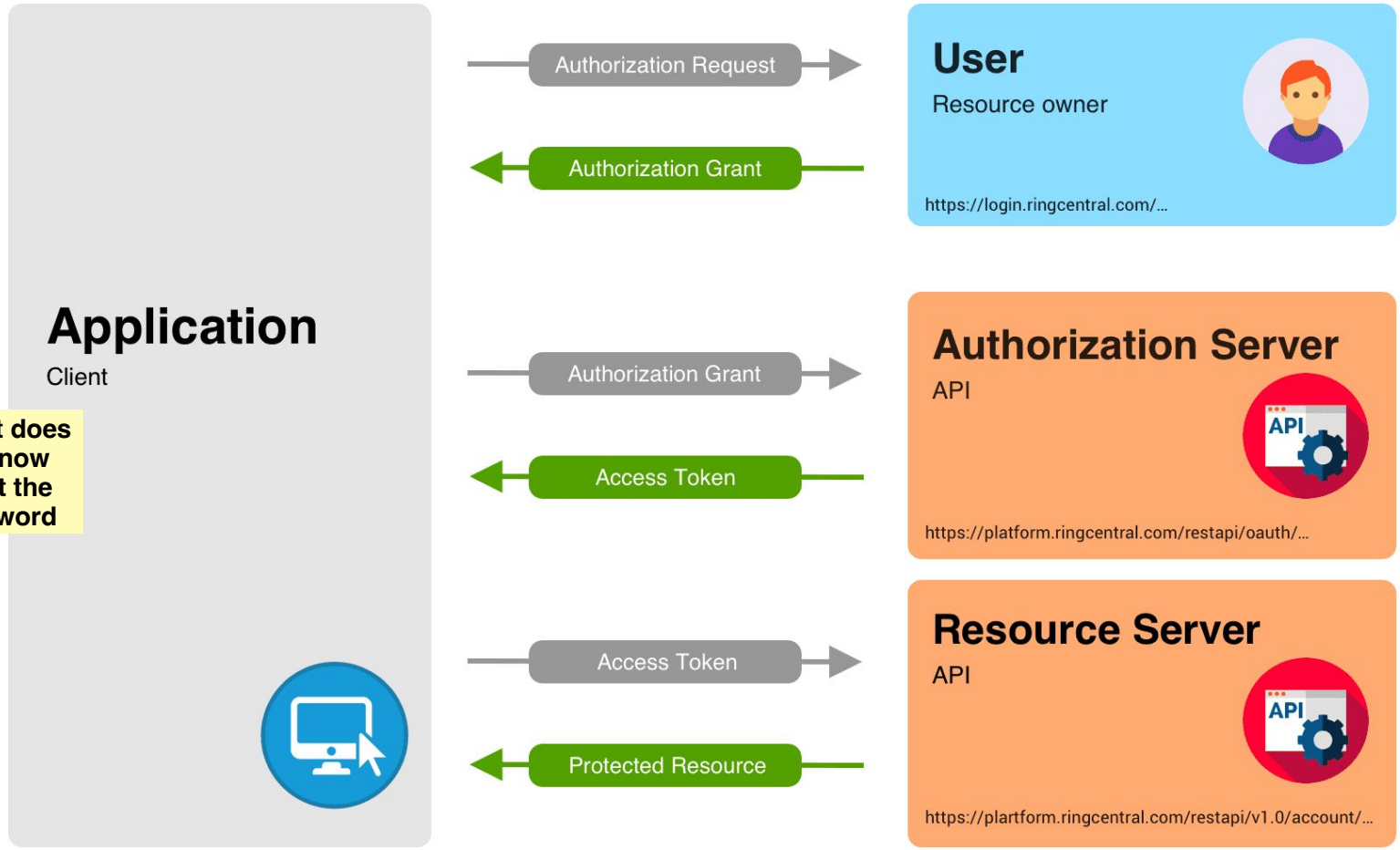


What could go wrong?

Different flows

this is like logging to ur student account, it redirects it to another source.
it gets one time code, says you can exchange this authorization code for token.
you can only use it once.

Authorization Code Flow

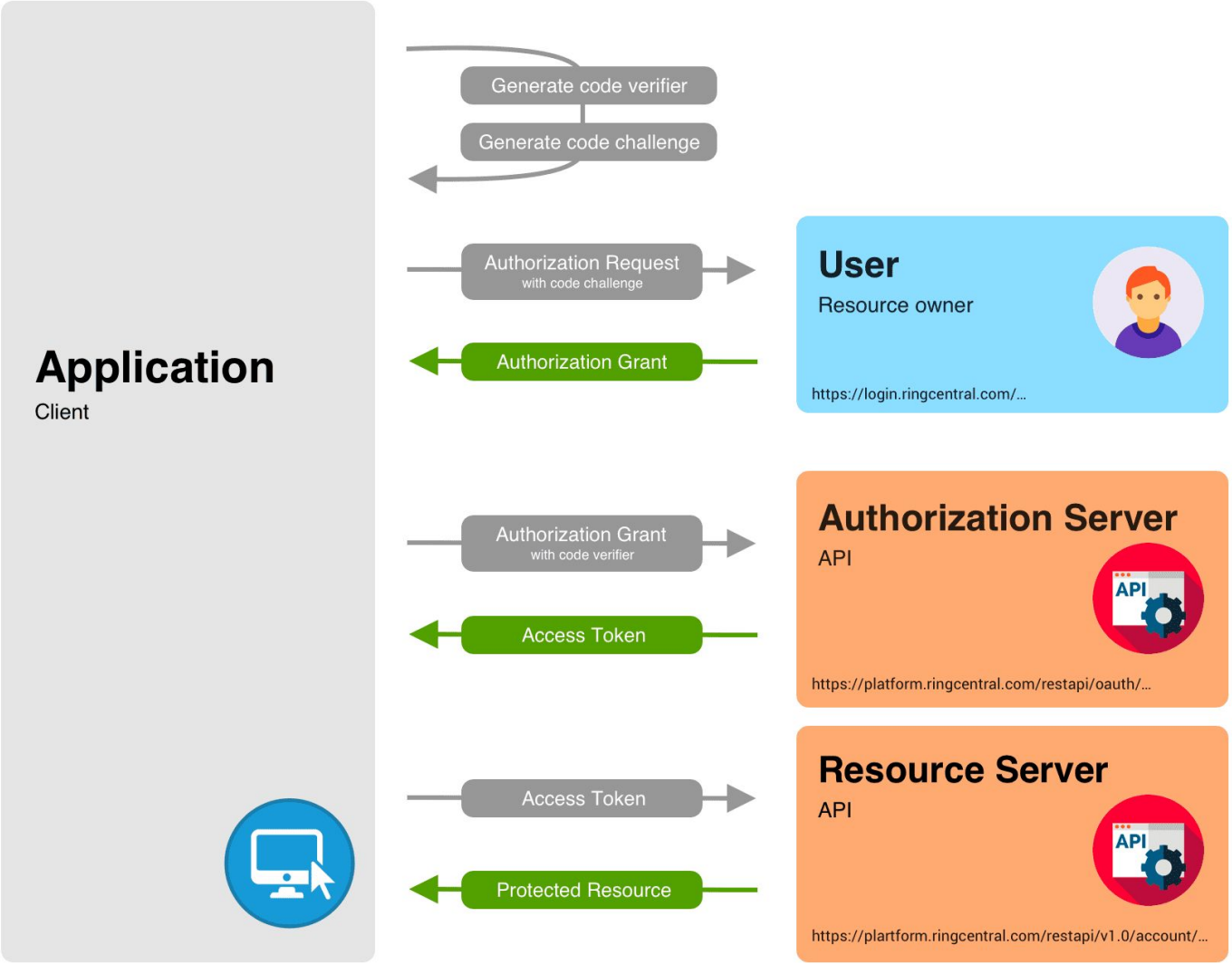


client does not know about the password

Different flows

we are not gonna use this, but you have to understand it

Authorization Code Flow with PKCE



Code what and code what?

The code challenge is a hashed and encoded version of the code verifier, which can be verified by the authorization server using the same hashing and encoding method.

The hashing and encoding method must be agreed upon in advance!

Let’s dive into a IAM

IAM - Identity and access management system

We will be using keycloak!

Keycloak

From the fine manual:

realms

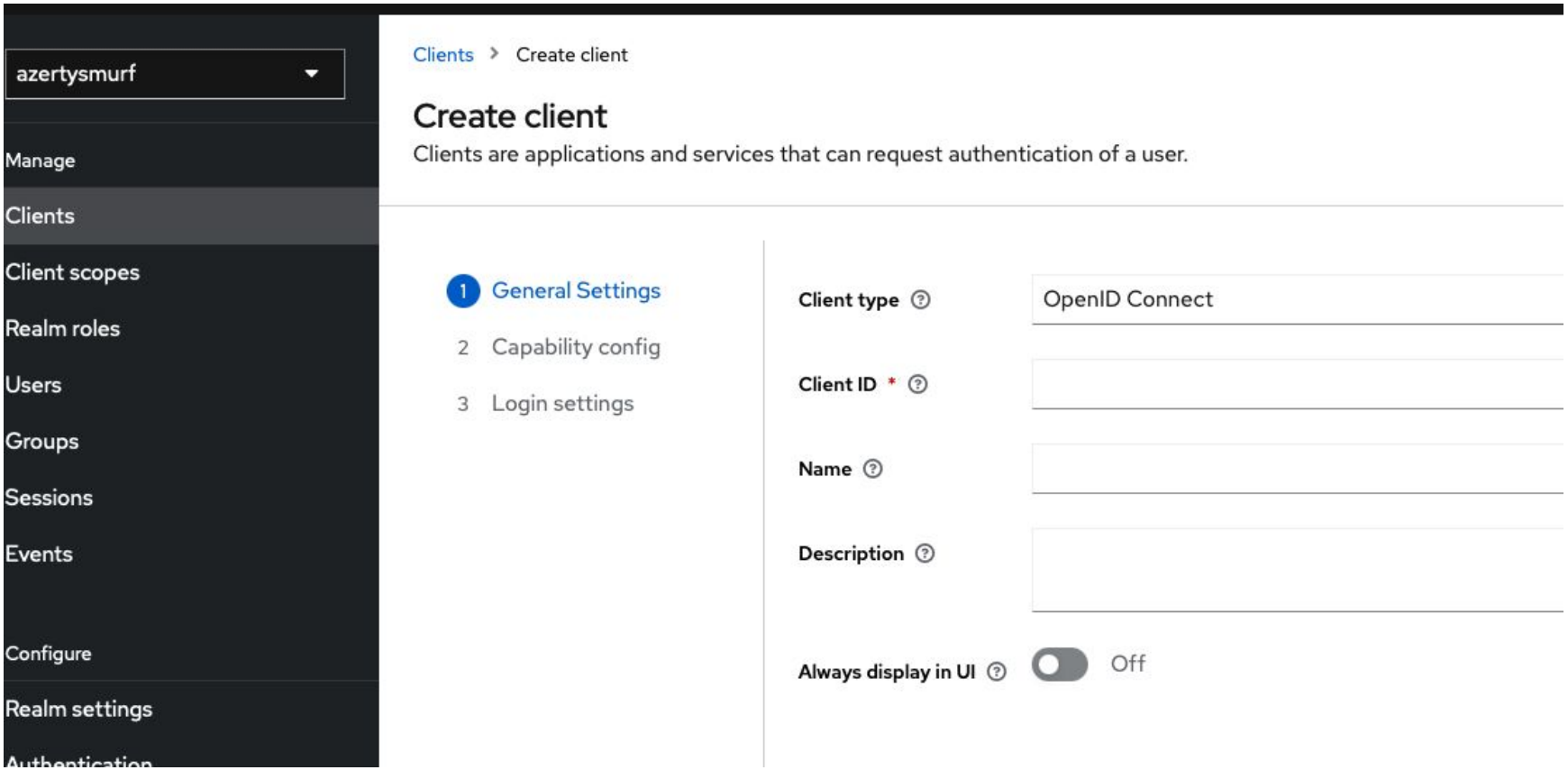
A realm manages a set of users, credentials, roles, and groups. A user belongs to and logs into a realm. Realms are isolated from one another and can only manage and authenticate the users that they control.

clients

Clients are entities that can request Keycloak to authenticate a user.

Create a realm

Add a client



Pick the right flows

Create client

Clients are applications and services that can request authentication of a user.

1 General Settings

2 Capability config

3 Login settings

Client authentication On

Authorization On

Authentication flow

- [x] Standard flow
- [x] Direct access grants
- [ ] Implicit flow
- [x] Service accounts roles
- [ ] OAuth 2.0 Device Authorization Grant
- [ ] OIDC CIBA Grant

put every redirection

Create client

Clients are applications and services that can request authentication of a user.

1 General Settings

2 Capability config

3 Login settings

Root URL

Home URL

Valid redirect URIs *

Add valid redirect URIs

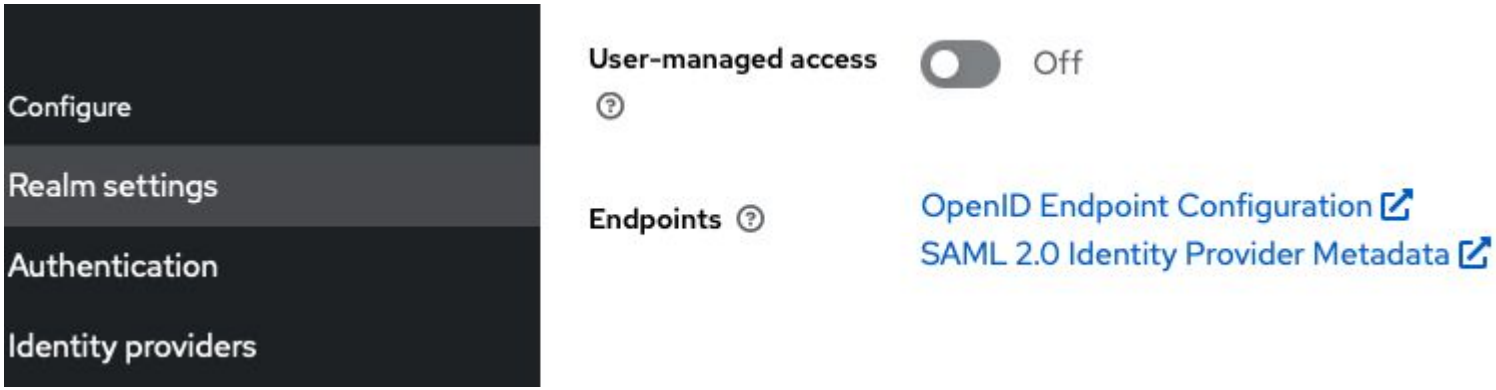
Valid post logout redirect URIs *

Add valid post logout redirect URIs

Web origins *

Add web origins

Important url!



for project:
ROLE MAPPING
pick one user per role

Search Postman | Sign In | Create Account

Import | POST https://lemur-9.cloud-ia

https://lemur-9.cloud-iam.com/auth/realms/azertysmurf/protocol/openid-connect/token | Save

POST https://lemur-9.cloud-iam.com/auth/realms/azertysmurf/protocol/openid-connect/token | Send

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings | Cookies

none | form-data | x-www-form-urlencoded | raw | binary

Key	Value	Bulk Edit
client_id	test	
client_secret	TNU[redacted]	
username	kevin	
password	kevin	
grant_type	password	
Key	Value	

Body | Cookies | Headers (16) | Test Results | Status: 200 OK | Time: 283 ms | Size: 2.92 KB | Save Response

Pretty | Raw | Preview | Visualize | JSON


```
{
    "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICIwSnkxVjJGcnEySU1rN0Zmb3Q4bVlnaEdsNzUtaGZHOFZPZlhYRU1kTGdvIn0.
        eyJleHAiOjE2OTQ2MTY3OTAsImlhdCI6MTY5NDYxNjQ5MCwianRpIjoiNDZiYjRiMjgtNWVmYy00ZjcwLTk1MDMtMDc3MTFmZWJhZTJhIiwiaXNzIjoiaHR0cHM6Ly9sZW11ci05Lm
        Nsb3VkLWlhbS5jb20vYXV0aC9yZWFsbXMvYXplcnR5c211cmYiLCJhdWQiOiJhY2NvdW50Iiwic3ViIjoiNGU0YzFmNWEtMTg0Ny00NGZlLWI4YzEtNTQ1MDc5ZTFlNGZmIiwidHlw
        IjoiQmVhcmVyIiwiYXpwIjoidGVzdCIsInNlc3Npb25fc3RhdGUiOiI2Mjc3YjcxMy00N2Y4LTRiOWMtYmUyZS01NDJiODVjMTY2NWQiLCJhY3IiOiIxIiwiYWxsb3dlZC1vcmlnaW
        5zIjpbIioiXSwicmVhbG1fYWNjZXNzIjp7InJvbGVzIjpbImRlZmF1bHQtcm9sZXMtYXplcnR5c211cmYiLCJvZmZsaW5lX2FjY2VzcyIsInVtYV9hdXRob3JpemF0aW9uIl19LCJy
        ZXNvdXJjZV9hY2Nlc3MiOnsiYWNjb3VudCI6eyJyb2xlcyI6WyJtYW5hZ2UtYWNjb3VudCIsIm1hbmFnZS1hY2NvdW50LWxpbmtzIiwidmlldy1wcm9maWxlIl19fSwic2NvcGUiOi
        Jwcm9maWxlIGVtYWlsIiwic2lkIjoiNjI3N2I3MTMtNDdmOC00YjljLWJlMmUtNTQyYjg1YzE2NjVkIiwiZW1haWxfdmVyaWZpZWQiOmZhbHNlLCJuYW1lIjoia2V2aW4iLCJwcmVm
        ZXJyZWRfdXNlcm5hbWUiOiJrZXZpbiIsImdpdmVuX25hbWUiOiJrZXZpbiIsImZhbWlseV9uYW1lIjoiIn0.
        oX6TlUf4k1OyE54BJ2cNmxyfbexcnvXNBljLdlSIq98ZUaTEi3F49_rO-LlebV-590lDJRVwoi-KIgoiuJuOOOftlDpH5-5L_p4IjOVwfiIu8vviR2ac54PJTF5NVHPpZFsOG0SJFx
        oAwaKgjZ31J-oR3AjrHXH8M_CE6mJSVQrAA-_N08zD9MTcPaGfDdcly_jcfMhRgqEuGYVmd9vqnOUBL8hpo1AoN8CBCzGPJ96HQuidQmhvxDcBbenLBzZUmvq0ZGjjOq7KZLvCb-ay
        e43buIcxSQXo6liDn9CDbNLZYjQ5i7gIyD1QBzN3XlJ62Y_H_G3JAm3DuawCgD7TdA",
    "expires_in": 300,
    "refresh_expires_in": 1800,
    "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICI1NmZmYjEwMS04YTExLTRhZDEtOWY1Ni1iMTcwYWU1OTU2ZjQifQ.
        eyJleHAiOjE2OTQ2MTgyOTAsImlhdCI6MTY5NDYxNjQ5MCwianRpIjoiODhmYWJlOTUtMGM0NC00M2VmLTlmZTItOWZlZmQ3N2RjNDgzIiwiaXNzIjoiaHR0cHM6Ly9sZW11ci05Lm
        Nsb3VkLWlhbS5jb20vYXV0aC9yZWFsbXMvYXplcnR5c211cmYiLCJhdWQiOiJodHRwczovL2xlbXVyLTkuY2xvdWQtaWFtLmNvbS9hdXRoL3JlYWxtcy9hemVydHlzbXVyZiIsInN1
        YiI6IjRlNGMxZjVhLTE4NDctNDRmZS1iOGMxLTU0NTA3OWUxZTRmZiIsInR5cCI6IlJlZnJlc2giLCJhenAiOiJ0ZXN0Iiwic2Vzc2lvbl9zdGF0ZSI6IjYyNzdiNzEzLTQ3ZjgtNG
```

Got our tokens, but what now?

This token can be used as a a “Bearer token”.

What’s a bearer token?

A Bearer token means that the bearer (who holds the access token) can access resource servers without further identification.
Of course, the bearer token must be valid and the resource server must have the ability to validate it!
That is why it is important that bearer tokens are protected.

Window of opportunity

How?

The access token has a limited valid time -> default 5 minutes.

The refresh token has a default validity time of 1 hour.

The token is signed and should be able to check if the signature is valid and will check who issued the token.

The token consists of so called “claims”, these can then be interpreted by the resource server. In openid there are 7 reserved claims

- `iss` (issuer): Issuer of the JWT
- `sub` (subject): Subject of the JWT (the user)
- `aud` (audience): Recipient for which the JWT is intended
- `exp` (expiration time): Time after which the JWT expires
- `nbf` (not before time): Time before which the JWT must not be accepted for processing
- `iat` (issued at time): Time at which the JWT was issued; can be used to determine age of the JWT
- `jti` (JWT ID): Unique identifier; can be used to prevent the JWT from being replayed (allows a token to be used only once)

Spring and the resource server

Spring always tries to configure your application depending on the dependencies you have added and your configuration.

What does the spring need besides the dependency?

The issuer URL

The certificates url

Audience (optional)

Now we can check whether the access token is valid, but not yet whether we have the correct “claims”.

Where can I find my role (-> extra claims)

```
"realm_access": {
   "roles": [
     "default-roles-test",
     "ThisIsARole",
     "offline_access",
     "uma_authorization"
   ]
  },
```

-> read the claims and use these to map to authorities/roles…

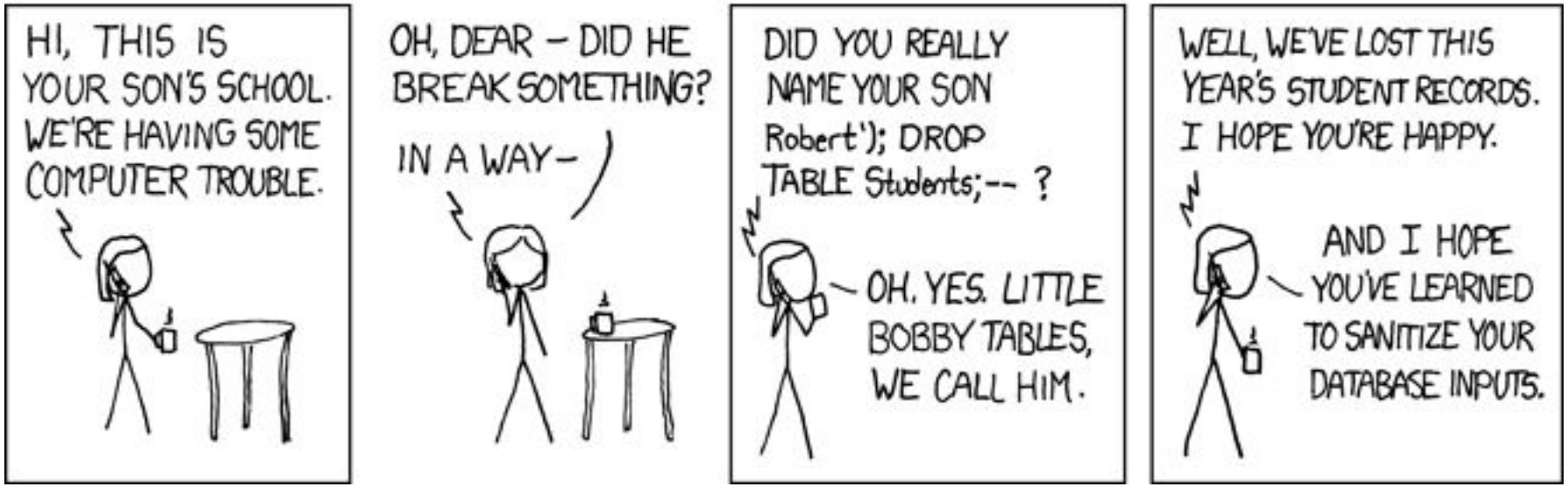
You should configure spring in order to do this mapping.

Our application is secure!

OWASP

https://owasp.org/API-Security/editions/2023/en/0x11-t10/

Input sanitation



Passwords in GIT?

We've found 4,232 code results

Sort: **Recently indexed**

Canbing007/sec-main-dashboard – sftp-config.json JSON

Showing the top match. Last indexed 2 hours ago.

```
18      "user": "root",
19      "password": "111111",
20      //"port": "22",
21
22      "remote_path": "/usr/local/nginx/nginx/html/test/",
```

nishun0419/node-test – sftp-config.json JSON

Showing the top match. Last indexed 4 hours ago.

```
18      "user": "vagrant",
19      "password": "vagrant",
20      "port": "2222",
21
22      "remote_path": "/home/vagrant/personal-development",
```

Arnavgupta7/Virtual-machine – sftp-config.json JSON

Showing the top match. Last indexed 10 hours ago.

```
18      "user": "arnav7",
19      "password": "myakanksha7119sis27",
20      // "port": "20",
21
22      "remote_path": "/home/arnav7/ECS150/HW4",
```

Vault

https://spring.io/projects/spring-vault

Copy pasted code from stackoverflow/chatgpt

https://arxiv.org/abs/1910.01321

arXiv > cs > arXiv:1910.01321

Search...
Help

Computer Science > Software Engineering

[Submitted on 3 Oct 2019 (v1), last revised 19 Jan 2021 (this version, v2)]

An Empirical Study of C++ Vulnerabilities in Crowd-Sourced Code Examples

Morteza Verdi, Ashkan Sami, Jafar Akhondali, Foutse Khomh, Gias Uddin, Alireza Karami Motlagh

Software developers share programming solutions in Q&A sites like Stack Overflow. The reuse of crowd-sourced code snippets can facilitate rapid prototyping. However, recent research shows that the shared code snippets may be of low quality and can even contain vulnerabilities. This paper aims to understand the nature and the prevale of security vulnerabilities in crowd-sourced code examples. To achieve this goal, we investigate security vulnerabilities in the C++ code snippets shared on Stack Overflo over a period of 10 years. In collaborative sessions involving multiple human coders, we manually assessed each code snippet for security vulnerabilities following CWE (Common Weakness Enumeration) guidelines. From the 72,483 reviewed code snippets used in at least one project hosted on GitHub, we found a total of 69 vulnerable co snippets categorized into 29 types. Many of the investigated code snippets are still not corrected on Stack Overflow. The 69 vulnerable code snippets found in Stack Over were reused in a total of 2859 GitHub projects. To help improve the quality of code snippets shared on Stack Overflow, we developed a browser extension that allow Stack Overflow users to check for vulnerabilities in code snippets when they upload them on the platform.

OWASP dependencies

```
25
26      //MAJOR VULNERABILITY
27      //CVE-2021-44228 -> Base Score: CRITICAL (10.0)
28      implementation 'org.apache.logging.log4j:log4j-api:2.13.3'
29      implementation 'org.apache.logging.log4j:log4j-core:2.13.3'
30  }
31
```

Published Vulnera

CVE-2021-44228

CISA Known Expl

- Product: Apache Log4j2
- Name: Apache Log4j2 Remote Code Execu
- Date Added: 2021-12-10
- Description: Apache Log4j2 contains a vuln
- Required Action: For all affected software a the measures provided at https://www.cisa.
- Due Date: 2021-12-24

Apache Log4j2 2.0-beta9 through 2.15.0 (excludin endpoints. An attacker who can control log messa by default. From version 2.16.0 (along with 2.12.2, Services projects.

CWE-400 Uncontrolled Resource Consumption, C

CVSSv2:

- Base Score: HIGH (9.3)
- Vector: /AV:N/AC:M/Au:N/C:C/I:C/A:C

CVSSv3:

- Base Score: CRITICAL (10.0)
- Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:

owasp dependency-check

- dependencyCheckAggregate
- dependencyCheckAnalyze
- dependencyC
- dependencyCheckUpdate

Identifies and reports

Questions?