Authorization Services with Keycloak

MSc. Carlos Avendaño

OAuth2 - Check In at a hotel

Aaron Parecki - Senior Security at Okta

You go to the front desk, you give that person your ID and your credit card. They give you back a hotel key card. You take that key card and you swipe it on the door and the door lets you in.

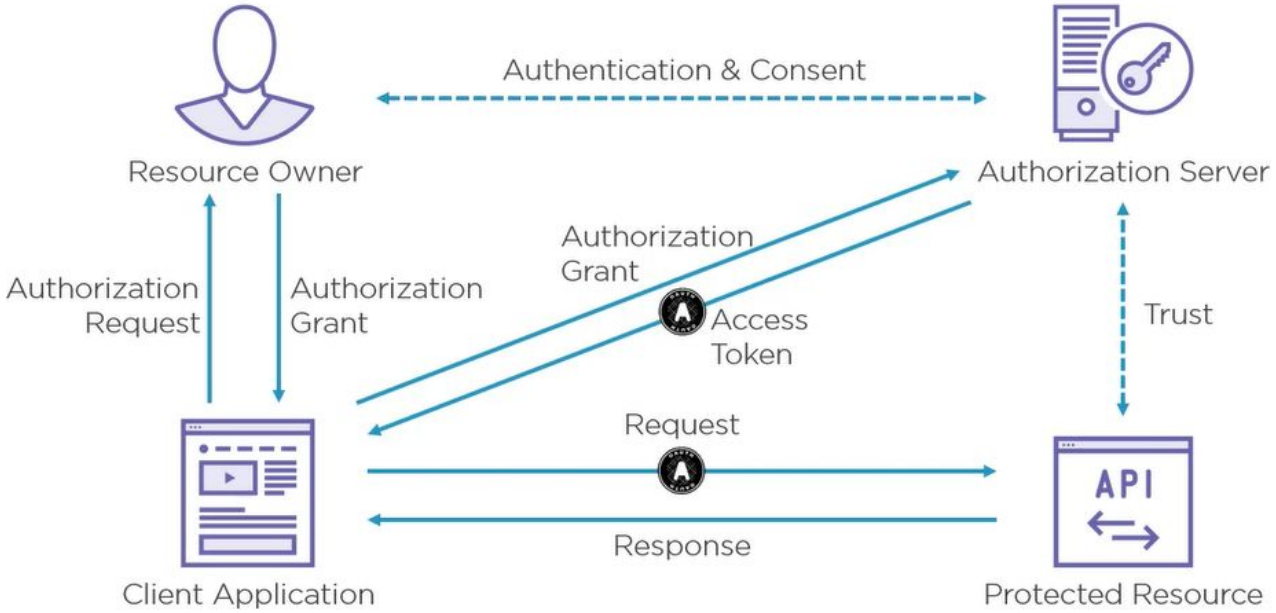
Note. The key card does not represent you as a person. It just needs to represent that you have access to this door. So the door does not care you about who you are. The door just cares about whether this key card has access or not.

The person at the front desk -> **Authorization Server**

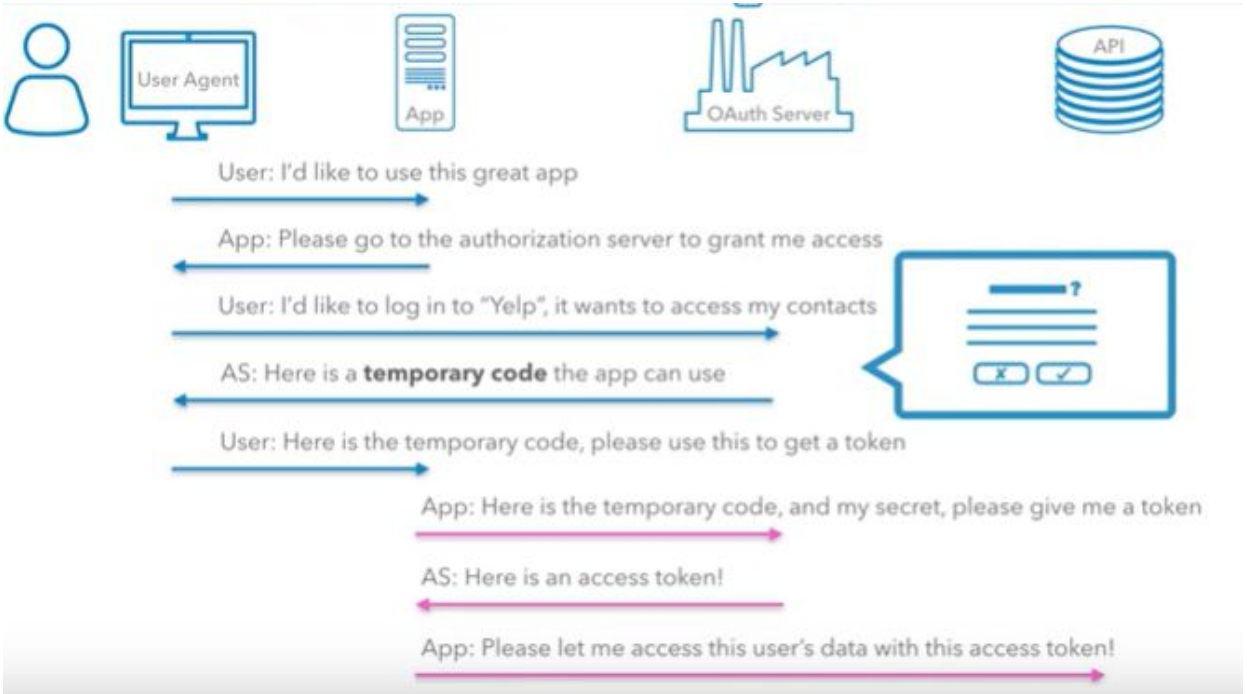
Key Card -> **Token**

Door -> **API**

OAuth2. The Authorization Code Flow



OAuth2. The Authorization Code Flow



OAuth Scope

A permission to do something within a protected resource on behalf of the resource owner.

“What those permissions are and how coarse or fine-grained they are is Up To You”

We could have a scope that represents an API in its entirety or it could be a particular type of access within that API, or it could be a particular piece of functionality.

Scopes

- media.management_api
- media.management_api.read
- media.management_api.feed

Front Channel: Authorization Request

```
https://authorization-server.com/auth
?response_type=code
&client_id=29352735982374239857
&redirect_uri=https://client.example.com/callback
&state=xcoivjuywkdkhvusuye3kch
&scope=resource1.create resource1.delete
```

Front Channel: Authorization Response

```
https://client.example.com/callback
?code=g0ZGZmNjVmOWI&state=dkZmYxMzE2
&state=xcoivjuywkdkhvusuye3kch
&scope=resource1.create resource1.delete
```

Back Channel: Token Request

```
POST /token HTTP/1.1
    Host: authorization-server.com
    Content-type: application/x-www-form-urlencoded
    Authorization: Basic Y2F2ZW5kYW5vYToxMjM0NTY=

    grant_type= authorization_code
    code=g0ZGZmNjVmOWI&state=dkZmYxMzE2
    client_id=29352735982374239857
    redirect_uri=https:/client.example.com/cb
```

Back Channel: Token Response

```
HTTP /1.1 200 OK
    Content-type: application/json
    {
        “access_token” : “AYjcyMzY3ZDhiNmJkNTY”,
        “Refresh_token” : “RjY2NjM5NzA2OWJjuE7c”,
        “token_type” : “Bearer”,
        “expires_in” : 3600,
        “scope”: “resource1.create resource1.delete”
    }
```

OpenID Connect

OAuth is not authentication. It does not give the client application any sort of indication of who the user is and how they authenticated.

OpenID Connect not only adds identity to OAuth, but it also formalize some of the OAuth ambiguity by defining token types and standardizing cryptography and validation procedures. It does not change OAuth in any way. It only adds to it.

By using OpenID Connect, your Authorization Server also acts as an identity provider.

OAuth Services

Open Source

- Glewlwyd
- Keycloak
- OAuth.io
- ORY Hydra
- SimpleLogin
- SSQ signon

Commercial

- Auth0
- Curity Identity Server
- FusionAuth
- Okta
- Red Hat Single Sign-On

https://oauth.net/code/

Keycloak - The Authorization Process

- Resource Management
- Permission and Policy Management
- Policy Enforcement

Resource Management

It involves all the necessary steps to define what is being protected. What you are looking to protect which usually represents a web application or a set of one or more services.

Scopes usually represents the actions that can be performed on a resource, but they are not limited to do that. You can also use scopes to represent one or more attributes within a resource.

Permission and Policy Management

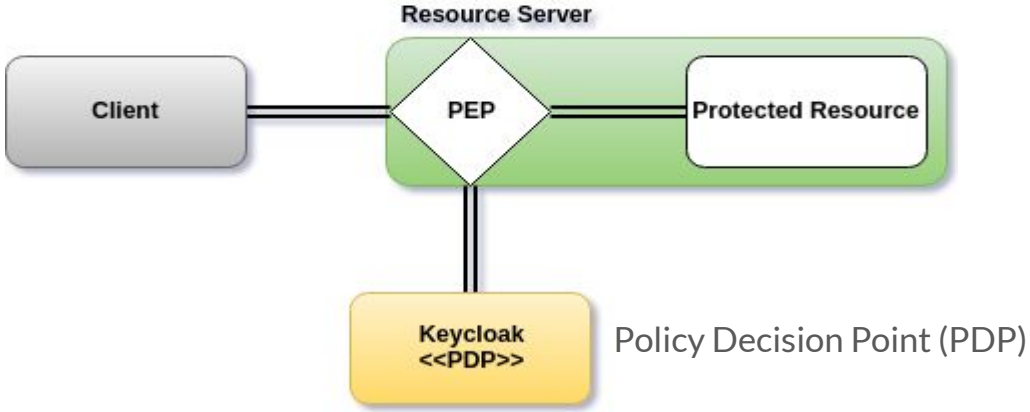
This process involves all the necessary steps to actually define the security and access requirements that govern your resources.

Policies define the conditions that must be satisfied to access or perform operations on something (resource or scoped), BUT they are not tied to what they are protecting. They are GENERIC and can be reused to build permissions or even more complex policies.

Permissions are coupled with the resource they are protecting. Here you specify what you want to proyect (resource or scoped) and the policies that must be satisfied to grant or deny permissions.

Policy Enforcement

It involves the necessary steps to actually enforce **authorization decisions** to a resource server. This is achieved by enabling a **Policy Enforcement Point** (PEP) at the resource server that is capable of communicating with the authorization server, ask for authorization data and control access to protected resources based on decisions and permissions returned by the server.



Policy Decision Point (PDP)

Keycloak - Example

Users(roles)

alice(user)
jdoe(user, user-premium)
cavendanoa(user)

Resources and Scopes

Default Resource
Premium Resource (view, delete)
Admin Resource

Policies - Type

Only User Policy - role
Only Premium User Policy - role
Admin-Policy-Based-User - user

Permissions- Type

Default Resource Permission - resource
Premium Resource Scope View Permission - scope
Premium Resource Scope Delete Permissions - scope
Admin Resource Permission - resource

Policy Enforcement Configuration

```
{
 "policy-enforcer": {
        "user-managed-access" : {},
        "enforcement-mode" :
"ENFORCING"
        "paths": [
        {
        "path" : "/someUri/*",
        "methods" : [
        {
        "method": "GET",
        "scopes" :
["urn:app.com:scopes:view"]
        },
        {
        "method": "POST",
        "scopes" :
["urn:app.com:scopes:create"]
        }
        ]
        },
        {
        "name" : "Some Resource",
        "path" : "/usingPattern/{id}",
        "methods" : [
        {
        "method": "DELETE",
        "scopes" :
["urn:app.com:scopes:delete"]
        }]},
        {
        "path" : "/exactMatch"
        },
        {
        "name" : "Admin Resources",
        "path" : "/usingWildCards/*"
        }
        ]
}
}
```

RBAC vs ABAC

Characteristic	RBAC	ABAC
Flexibility	✓ (For small and medium-sized organizations)	✓
Scalability	–	✓
Simplicity	Easy to establish roles and permissions for a small company, hard to maintain the system for a big company	Hard to establish all the policies at the start, easy to maintain and support
Support for simple rules	✓	✓
Support for complex rules	✓	✓
Support for rules with dynamic parameters	–	✓
Customizing user permissions	– (Every customization requires creating a new role)	✓
Granularity	Low	High

References

1. https://oauth.net/2/
2. https://app.pluralsight.com/ Course: Getting Started with OAuth2.0
3. Keycloak - Authorization Services Guide
   https://www.keycloak.org/docs/latest/authorization_services/index.html
4. https://www.ekransystem.com/en/blog/rbac-vs-abac
5. RedHat Policy Enforcement
   https://access.redhat.com/documentation/en-us/red_hat_single_sign-on/7.1/html/authorization_services_guide/enforcer_overview