Spring Cloud Security offers a set of primitives for building secure applications and services with minimum fuss. ... Building on Spring Boot and Spring Security OAuth2 we can quickly create systems that implement common patterns like single sign on, token relay and token exchange.

Spring Cloud Security

Outline

Spring Cloud Security using OAuth2
by Pratap Kumar

Building Java microservices with Spring Cloud

- In this module , we will look into how to build secure services using OAuth 2

Content

Spring Cloud Security using OAuth2
by Pratap Kumar

- The Role of Security in microservices
- Problem with status quo
- What is Spring (Cloud) Security?
- What is OAuth 2.0?
- OAuth 2.0 Authorization code grant type
- Creating a Resource Server
- OAuth 2.0 password credential grant type
- Creating a Custom Authorization Server
- OAuth 2.0 client credential grant type
- Adding method access rules
- Advanced Token options
- Summary

The Role of Security in microservices

Spring Cloud Security using OAuth2
by Pratap Kumar

- **User Authentication / Authorization**
  - Ideally each microservices is not doing its Authentication that would be bit of oppressive , but you do need to think of Authorization,
  - The User Authentication and Authorization comes into play in microservice application.
  - How do you work in this kind of low trust environment where services that spanning machines , network , even location boundary , your private and public cloud.
  - How to keep the wrong people from accessing the right resource.
  - I need a way to manage credential , I need to check the access of the requester and I need to make sure people are doing what they are suppose to.

The Role of Security in microservices

Spring Cloud Security using OAuth2
by Pratap Kumar

- **Single Sign-On**
  - Single sign-on comes to play, because you are potentially chaining a lot of services together,
  - So the idea of single sing-on is that I am not authenticating on every request and grinding that request to a halt
  - Instead being able to authenticate once lets me sign into with number of services and authenticate with them.
- **Data Security**
  - Security in transit at rest , This is not just the data going in and out of services but even configuration.
- **Interoperability**
  - MicroServices Architecture is often categorized by team shipping at different time even with different programming language,
  - So any Authentication / Authorization scheme should work across platform so that all these different microservices can use same short of protocol

Problem with Status Quo

- Credentials embedded in applications
- Unnecessary Permissions
- Differentiating Users and Machines
- Not optimized for diverse client

Spring (Cloud) Security

- **Service Authorization powered by OAuth 2. 0**

What is OAuth 2.0?

- Protocol for conveying authorization
- Provides authorization flow for various clients
- Obtain limited access to user accounts
- Separates idea of user and client
- Access Token carries more than identity
- Not an Authentication scheme

Actors in OAuth 2.0 Scenario

- **Resource Owner**
  - Entity that grant access to a resource, Usually you!
- **Resource Server**
  - Server hosting the protected resources
- **Client**
  - App making protected resource request on behalf of resource owner
- **Authorization Server**
  - Server Issuing access tokens to clients

OAuth 2.0 Terms

- Access Token
- Refresh Token
- Scope
- Client ID / Secret
- OpenID Connect
- JWT

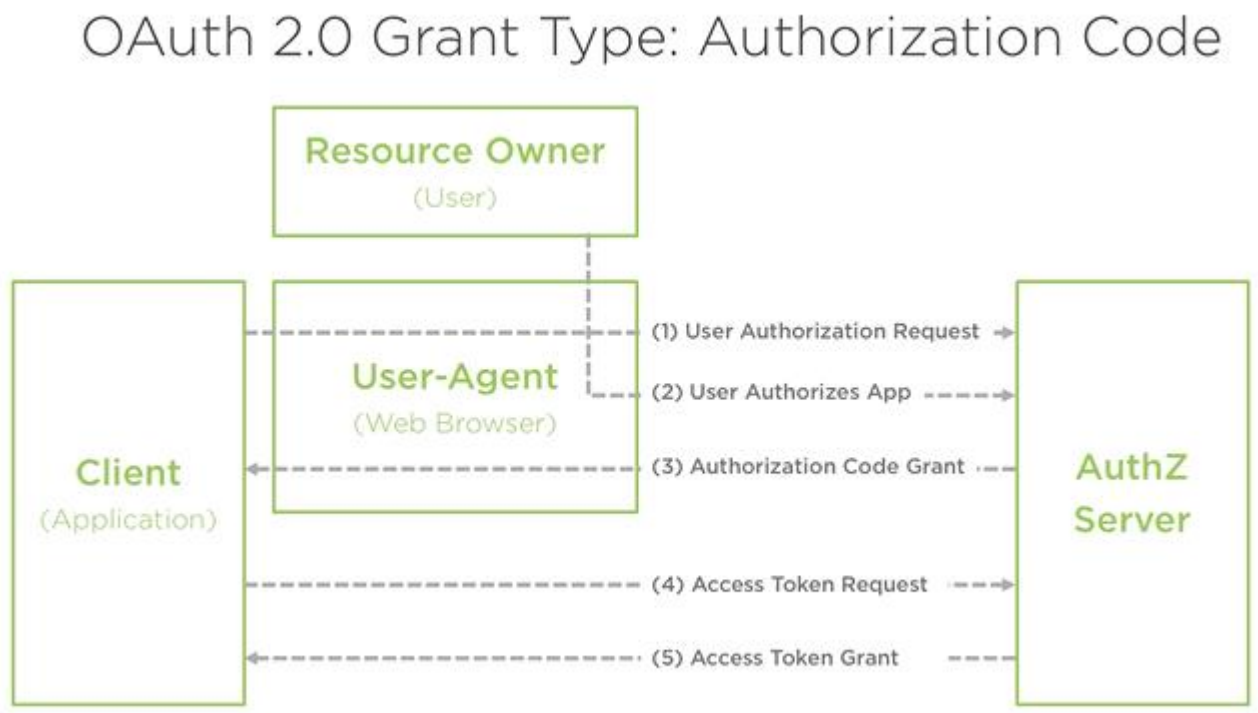
How Spring Support OAuth 2.0

- Code Annotation
  - @EnableOAuthSSO , @EnableResourceServer,
  - @EnableOauth2Client
- Token Storage Options
- OAuth 2.0 endpoints
  - /authorize , /token , /check , /error , /tokenkey
- Numerous extensibility endpoints

OAuth Flow

Spring Cloud Security using OAuth2
by Pratap Kumar

Authorization code flow



Spring Cloud Security using OAuth2
by Pratap Kumar

Demo

- Build back-office Toll reporting site
- Add OAuth2 annotations to controller
- Authenticate and authorize via GitHub
- Watch redirects during flow
- Choose which pages to protect

Demo

Spring Cloud Security using OAuth2
by Pratap Kumar

- **Building back-office Toll reporting site**

- new > spring starter project

  > spring-cloud-secureui

> Dependencies

  Web

  Cloud Oauth2

  Cloud Security

  Thymeleaf

Routing Tokens

Spring Cloud Security using OAuth2
by Pratap Kumar

Creating a Resource Server and Routing Tokens to Downstream Services

- How to flow credentials downstream from a front end controller and i want to call a secure service?
- How do i take that token and pass that down the chain?
  - Use @EnableResourceServer annotation
  - Can combine Authz and Resource servers
  - user-info-uri and token-info-uri properties
  - OAuth2RestTemplate bean
    - (fetches all the user details from the Authorization server , It injects right things into the header)

Demo

Spring Cloud Security using OAuth2
by Pratap Kumar

- Create resource server to return toll data
- Add @EnableResourceServer annotation
- Update application.properties
- Test via HTTP calls
- Update web client to securely call service

Demo

Spring Cloud Security using OAuth2
by Pratap Kumar

- **Creating a Resource Server**

- New > spring starter project

  > spring-cloud-secureservice

- Dependencies:
  - Web
  - Cloud-Security
  - Cloud-OAuth2

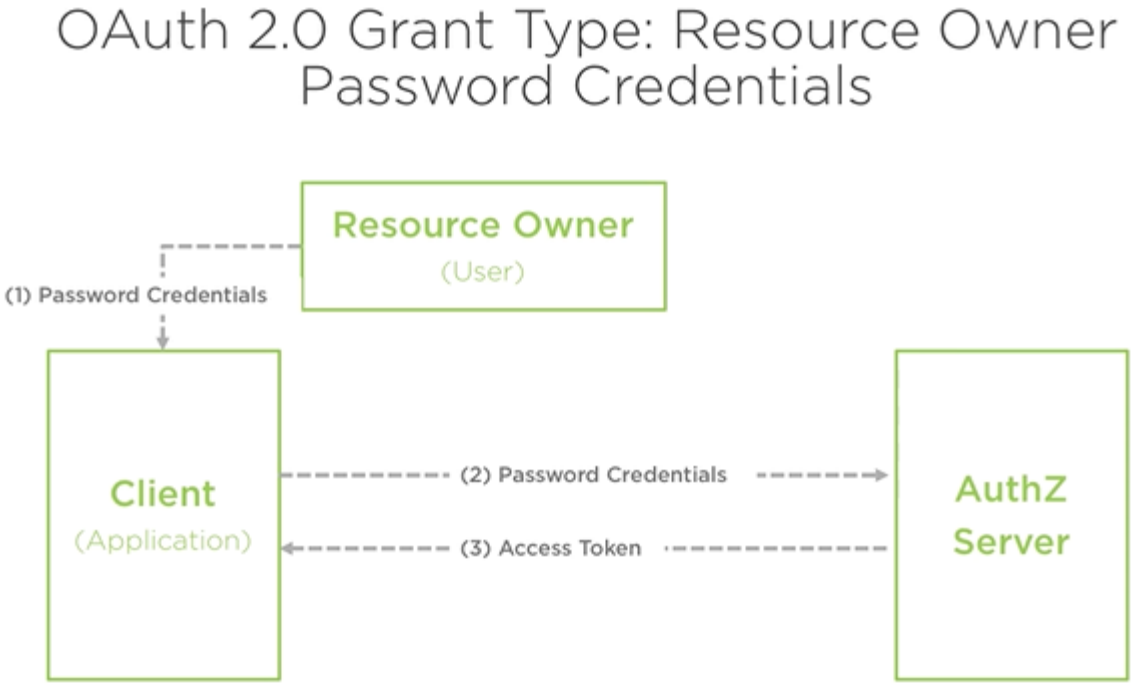
OAuth2 Grant type

Spring Cloud Security using OAuth2
by Pratap Kumar

Resource Owner Password Credentials

- The user provide their service credentials such as username, password directly to the application which then use those credentials to get the access token.
- This is not the ideal flow, This type should be used if the other flows are not viable.
- This grant type should be used for trusted client application.
- This grant type is little less secure as you are providing the credentials to the app.

OAuth2 Grant type:
Resource Owner Password Credentials



Spring Cloud Security using OAuth2
by Pratap Kumar

Authz Server

Spring Cloud Security using OAuth2
by Pratap Kumar

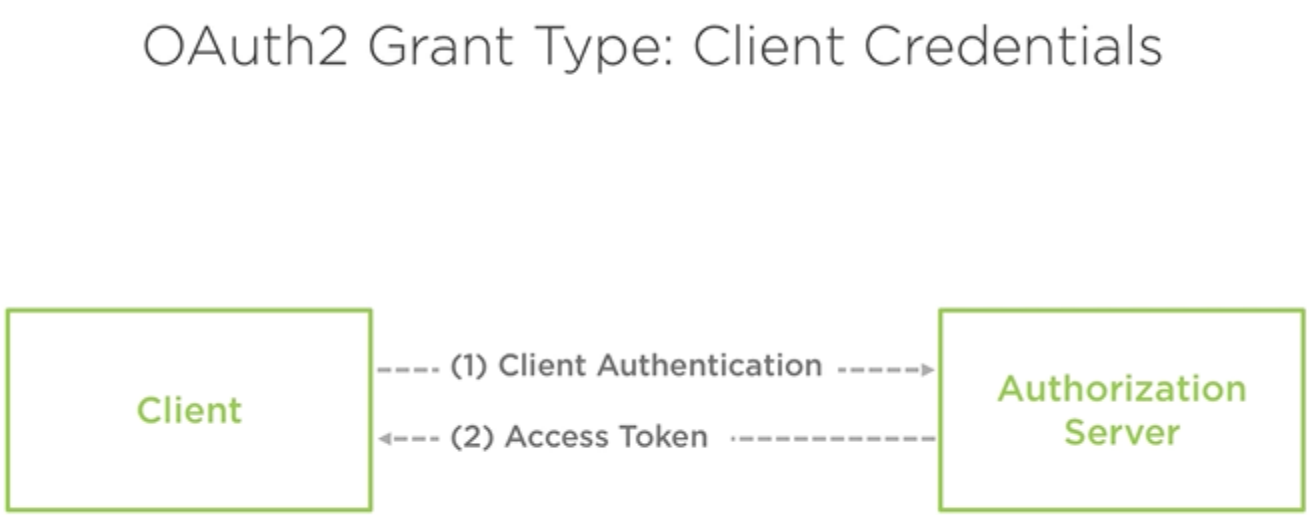
- **Creating a Custom Authorization Server**
  - Lit up with @EnableAuthorizationServer
  - Set of endpoints provided
  - Property settings drive behavior
  - Secured with Basic Authentication

Demo

Spring Cloud Security using OAuth2
by Pratap Kumar

- Create new Spring app
- Add Authorization Server annotation
- Set properties for the server
- Test token generation with API calls
- Update resource server to use custom authorization server
- Test resource server with API calls
- Create application that relies on system user
- Call resource server from client app with password credentials

OAuth2 Grant type: Client Credentials



Spring Cloud Security using OAuth2
by Pratap Kumar

Client Credentials

Spring Cloud Security using OAuth2
by Pratap Kumar

- This one is a super basic, barely used.
- This is used, where the client is the resource owner.
- There is no authorization to be obtained from the end user.
- The clientId and clientSecret is enough to get the simple token.
- This could be useful when the client itself is the authorization server.
- Not used in realistic scenario.
- However you could test it by specifying the grant_type=clientcredential

Access Rules

- Adding Access rules to our resource server
- What if i want to do some security a little more fine grained.
- We can use @PreAuthorize to control
  - Can a method be called?
  - Check OAuth2 scope
  - Check user role

```
@PreAuthorize("#oauth2.hasScope('READ')
                    and hasAuthority('ROLE_ADMIN')")
@RequestMapping("/")
public String secureCall(){
        return "success!";
}
```

Demo

Spring Cloud Security using OAuth2
by Pratap Kumar

- Create extension class to get OAuth2 filters
- When you authenticate with OAuth2, Values do get added to your spring security session, but by default those classes have no concept of OAuth.
- So we have biuild a extension to get those OAuth filters
- we will add those role filters to our method and test.
- Add OAuth2 filter to method and test.
- Apply both filter to method and test

res

- https://www.baeldung.com/spring-cloud-security
- https://www.youtube.com/watch?v=USMl2GNg2ro