<http://kimseunghyun76.tistory.com/403>

### [spring-security-oauth 심층 분석 - 1](http://kimseunghyun76.tistory.com/403)

멋진승현 2016.03.14 18:22

스프링 시큐리티 OAuth 에 대한 부분을 정리 합니다.

뭐, 그냥 레퍼런스와 Sample 예제를 보면서, 이런저런 내용을 정리하는데에 포인트를 맞춰볼 예정입니다.

"인증 토큰을 발행하고, 그 발행된 인증 토큰을 사용해서 API를 사용한다" 가 목표입니다.

뽀인트!!! 은 바로 , @EnableAuthorizationServer  와 @EnableResourceServer 이 두가지 어노테이션입니다.

토큰을 발행하고, 발행된 토큰을 검증하는 것의 역활은 @EnableAuthorizationServer 어노테이션이 하는 역활이고,

@EnableResourceServer 는 위의 토큰과 함께 호출하는 API에 대한 것을 검증하는 필터같은 역활을 하게 됩니다.

별다른 설정이 없으면, in-memory 형태로 토큰이 저장되는 점, 만약 databases 연동을 하고자 한다면,

TokenStore 에 datasource를 연결하면 됩니다.

대부분의 예제는 우선 인증 서버 와 리소스 서버(API)가 분리되어 있지 않고, 함께 구성되어 있습니다.

또한 발행된 Token을 저장은 H2을 사용하게 됩니다.

이에 따라, 인증과 리소스 서버를 분리하고, h2를 사용하지 않고 mysql을 사용할 예정입니다.

제공해주는 DB Table Schema를 모두 mysql 용으로 변경을 해야 겠네요.

우선 프로젝트를 하기 앞서서 ,

Spring Security OAuth 에서 테스트 코드를 먼저 분석을 하고 난 뒤에 , 이것을 커스텀마이징을 하겠습니다.

**A. Vanilla : a basic , no-frillsAuthorization Server and Resource Server.**

<https://github.com/spring-projects/spring-security-oauth/tree/master/tests/annotation/vanilla>

이 프로젝트는 인증서버와 리소스 서버를 최소 설정으로 셋업하는 것을 보여드립니다.

인증 서버에 대해서 여러분은 @EnableAuthorizationServer가 필요하고, 또한 최소한으로 하나의 클라이언트 등록 설정하는 것이 필요합니다.(OAuth2ClientDetails)

이것이 Application.java 의 벌크라는 것을 볼수 있습니다.

AuthenticationManager가 Spring Boot에 의해 생성됩니다.(이것은 application.yml 마다  "user" 라는 이름으로, "password"라는 패스워드와 함께 싱글유저를 가집니다. )

인증서버에서 Resource Owner Password 승인 타입을 위한 인증을 제공하기 위해 필요로 합니다.

리소스 서버가 필요로한 모든것은 @EnableResourceServer 어노테이션입니다.

기본적으로, 모든 명확하게 무시되지 않았거나, AuthentoizationEndpoint에 의해 표출되지 않은 리소스를 보호 합니다.

This project shows what you can do with the minimum configuration to set up an Authorization Server and Resource Server.

For the Authorization Server you need to @EnableAuthorizationServer and also configure at least one client registration (OAuth2ClientDetails).

You can see this is the bulk of  Application.java.

An AuthenticationManager is created by Spring Boot (it has a single user, named "user" , with password "password" , per application.yml)

It is needed in the Authorization Server to provide authentication for the Resource Owner Password grant type.

For the Resource Server all that is needed is the @EnableResourceServer annotation.

By default, it protected all resources that are not explicitly ignored and not exposed by the AuthorizationEndpoint (if there is an Authorization Server in the sample application.)

**b. jwt - uses Json Web Tokens as the token format**

Json Web Tokens를 이용한 인증 방식을 사용하면,

서버에서 발급한 암호화된 토큰 안에 인증정보와 부가 정보를 저장하고, 이를 헤더에 붙여서 요청하면 필터에서 검증을 하는 시스템이랍니다.

security-jwt 를 추가 한뒤에,

|  |
| --- |
| <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-jwt</artifactId>  </dependency> |

@EnableAuthorizationServer 상에,

|  |
| --- |
| @Bean public JwtAccessTokenConverter accessTokenConverter() {     return new JwtAccessTokenConverter(); }  @Override public void configure(AuthorizationServerSecurityConfigurer oauthServer) throws Exception {     oauthServer.tokenKeyAccess("isAnonymous() || hasAuthority('ROLE\_TRUSTED\_CLIENT')").checkTokenAccess(             "hasAuthority('ROLE\_TRUSTED\_CLIENT')"); }  @Override public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {     endpoints.authenticationManager(authenticationManager).accessTokenConverter(accessTokenConverter()); } |

이렇게 하면 끝!!! 그럼 Token 자체도,

{"access\_token":"eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOlsib2F1dGgyLXJlc291cmNlIl0sInNjb3BlIjpbInJlYWQiXSwiZXhwIjoxNDU3OTc3NTA5LCJhdXRob3JpdGllcyI6WyJST0xFX0NMSUVOVCJdLCJqdGkiOiI2NjFmNzEyYy1hMTJjLTQxMWMtYTYxNy0yMDFiZjI2OGExZGUiLCJjbGllbnRfaWQiOiJteS1jbGllbnQtd2l0aC1zZWNyZXQifQ.wpbFte1YLuRtaAQ-U2qdlrefCguPxUfl94TAFLs4ClA","token\_type":"bearer","expires\_in":43199,"scope":"read","jti":"661f712c-a12c-411c-a617-201bf268a1de"}

와 같이 나오게 됩니다.

**c. mappings - changes the default values for the endpoint paths and the protected resource paths**

기본값보다 조금 다른 앤드포인트 경로들을 사용하는 인증 서버와 리소스 서버를 최소한의 설정으로 셋업하는 것을 할수 있도록 보여줍니다.

인증서버에서는, 기본 'vanilla' 기능들에 추가적으로 , 우리는 "/oauth/token" 를 "/token" 으로의 매핑을 예로 추가합니다.

매핑의 타겟 값은 그것들 테스트 하기 쉽게 만들기 위해 @Value 를 사용해서 주입된다

여러분은 이것이 Application.java의 벌크인것을 볼수 있습니다.

리소스서버는 , 이 앱에서 우리는 기본 보호될 "/" 과 "/admin/beans" 의 리소스 패턴을  변경합니다.

이 앱의 나머지는 Spring Boot 자동 설정 기능으로 인해 , 기본으로 HTTP Basic security 에 의해 보호되어집니다.

(이것은 테스트 케이스인 ProtectedResourceTests 에서 검증됩니다.)

우리는 또한 접근 롤을 추가할 것입니다.(OAuth2 resources둘 다 scope = 'read' 을 요구합니다.)

This project show what you can do with the minimum configuration to set up an Authorization Server and Resource Server with different endpoint paths than the defaults.

For the Authorization Server, in addition to the basic "vanilla" features, we add mappings from "/oauth/token" to "/token" for instance.

The target values for the mappings are injected using @Value, largely to make it easier to test them.

You can see this is the bulk of Application.java

For the Resource Server, in this app we change the default protected resource patterns to "/" and "/admin/beans" .

The rest of the app is protected by HTTP Basic security by default because of the Spring Boot autoconfiguration features( this is verified in a test case ProtectedResourceTests).

We also add an access rule (scope='read' is required to access both OAuth2 resources).

여기서는 우선 application.yml 상에 경로를 변경 했습니다.

|  |
| --- |
| **oauth:  paths:  token:** /token  **authorize:** /authorize  **confirm:** /approve  **check\_token:** /decode  **token\_key:** /key |

|  |
| --- |
| @Configuration  @EnableResourceServer protected static class ResourceServer extends ResourceServerConfigurerAdapter {      @Override     public void configure(HttpSecurity http) throws Exception {         // @formatter:off         http                 // Just for laughs, apply OAuth protection to only 3 resources                 .requestMatchers().antMatchers("/","/admin/beans","/admin/health")                 .and()                 .authorizeRequests()                 .anyRequest().access("#oauth2.hasScope('read')");         // @formatter:on     }      @Override     public void configure(ResourceServerSecurityConfigurer resources) throws Exception {         resources.resourceId("sparklr");     }  }  @Configuration  @EnableAuthorizationServer protected static class OAuth2Config extends AuthorizationServerConfigurerAdapter {       */\*\*      \* Mappings      \*/* @Value("${oauth.paths.token:/oauth/authorize}")     private String tokenPath = "/oauth/token";      @Value("${oauth.paths.token\_key:/oauth/token\_key}")     private String tokenKeyPath = "/oauth/token\_key";      @Value("${oauth.paths.check\_token:/oauth/check\_token}")     private String checkTokenPath = "/oauth/check\_token";      @Value("${oauth.paths.authorize:/oauth/authorize}")     private String authorizePath = "/oauth/authorize";      @Value("${oauth.paths.confirm:/oauth/confirm\_access}")     private String confirmPath = "/oauth/confirm\_access";      @Autowired     private AuthenticationManager authenticationManager;      @Autowired     private ServerProperties server;      @Override     public void configure(AuthorizationServerSecurityConfigurer oauthServer) throws Exception {         oauthServer.checkTokenAccess("hasRole('ROLE\_TRUSTED\_CLIENT')");     }      @Override     public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {         String prefix = server.getServletPrefix();         endpoints.prefix(prefix);         // @formatter:off         endpoints.authenticationManager(authenticationManager)                 .pathMapping("/oauth/confirm\_access", confirmPath)                 .pathMapping("/oauth/token", tokenPath)                 .pathMapping("/oauth/check\_token", checkTokenPath)                 .pathMapping("/oauth/token\_key", tokenKeyPath)                 .pathMapping("/oauth/authorize", authorizePath);         // @formatter:on     }      @Override     public void configure(ClientDetailsServiceConfigurer clients) throws Exception {         // @formatter:off         clients.inMemory()                 .withClient("my-trusted-client")                 .authorizedGrantTypes("password", "authorization\_code", "refresh\_token", "implicit")                 .authorities("ROLE\_CLIENT", "ROLE\_TRUSTED\_CLIENT")                 .scopes("read", "write", "trust")                 .resourceIds("sparklr")                 .accessTokenValiditySeconds(60)                 .and()                 .withClient("my-client-with-registered-redirect")                 .authorizedGrantTypes("authorization\_code")                 .authorities("ROLE\_CLIENT")                 .scopes("read", "trust")                 .resourceIds("sparklr")                 .redirectUris("http://anywhere?key=value")                 .and()                 .withClient("my-client-with-secret")                 .authorizedGrantTypes("client\_credentials", "password")                 .authorities("ROLE\_CLIENT", "ROLE\_TRUSTED\_CLIENT")                 .scopes("read")                 .resourceIds("sparklr")                 .secret("secret");         // @formatter:on     } |

이렇게 하면, 위의 URL 매핑이 모두 변경이 된다라는 점이다.

**d. resource - a pure Resource Server (needs to be paired with an auth server and share a token store)**

이 프로젝트는 리소스 서버 세팅을 위한 최소 설정을 할수 있는 것을 보여 줍니다.

필요로한 모든 것은 @EnableResourceServer 와 TokenStore 입니다.

기본으로 명시적으로 무시하지 않은 모든 리소스를 막아줍니다.

This project show what you can do with the minimum configuration to set up a Resource Server

All that is needed is the @EnableResourceServer annotation and a TokenStore.

By default it protects all resources that are not explicitly ignored.

security-jwt 를 추가 한뒤에,

|  |
| --- |
| <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-jwt</artifactId>  </dependency> |

Resource Server 에 @EnableResourceServer 설정을 한다.

|  |
| --- |
| <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-jwt</artifactId>  </dependency> |

**c. jdbc - uses JDBC stores for everything**

<https://github.com/spring-projects/spring-security-oauth/tree/master/tests/annotation/jpa>

이 프로젝트는 인증서버와 JDBC 백앤드를 갖는 리소스 서버를 최소 설정으로 셋업하는 것을 보여드립니다.

인증 서버는 클라이언트(ClientDetailStore) , 토큰(TokenStore) , 인증 코드(AuthorizationCodeStore) 그리고 사용자 계정(UserDetailsManager) 에 대한 JDBC 백앤드을 가지고 있습니다.

이런 서비스들 조차도, 만약 stateful 승인 타입들이 사용된다면(인증 코드 또는 내재되어있는,함축성 있는), 수평적으로 확장된 인증서버는 까다로운 세션들을 공유하는 로드 밸런서를 앞에 둘 필요가 있습니다.

(또는 스프링 SessionAttributeStore 가 여러분이 보고 있는 이것에 추가적으로 제공해야 합니다.)

* stateful : The program has a memory(state)
* stateless : There's no memory (State) that's maintained by the program

AuthenticationManager가 생성됩니다.(application.yml 마다  "user" 라는 이름으로, "password"라는 패스워드와 함께 싱글유저를 가집니다. )

인증서버에서 Resource Owner Password 승인 타입을 위한 인증을 제공하기 위해 필요로 합니다.

리소스서버는 TokenStore를  인증 서버와 공유 하지만 리소스 서버가 구지 다른 서비스들에 대해서 알 필요가 없습니다.

(그래서  인증서버가 싱글 인스턴스 라면, 그들은 인메모리 로 해도 된답니다.)

뭐 이정도로 해석을 하고, 실제 코드로 들어갑니다.

This project shows what you do with the minimun configuration to set up an Authorization Server and Resource Server with JDBC backends.

The Authorization Server has JDBC backends for client (ClientDetailStore), tokens (TokenStore), authorization codes(AuthorizationCodeStore) and user accounts(UserDetailsManager).

Even with these services, a horizontally scaled Authorization Server needs to be fronted by a load balancer with sticky sessions (or else a Spring SessionAttributeStore should be provides in addition to what you see here), if the stateful grant types are used (authorization code or implicit).

An AuthenticationManager is created (it has a single user, named "user" , with password "password" , per application.yml).

It is needed in the Authorization Server to provide authentication for the Resource Owner Password grant type.

The Resource Server shares the TokenStore with the Authorization Server, but it doesn't need to know about the other services

(so they could be in-memory if there is a single instance of the Authorization Server)