Spring boot and Oauth2

Rokin Maharjan Follow
Dec 2, 2018 · 3 min read

In this guide we will learn how to secure a spring boot application using Oauth2. This tutorial is for developers who have experience in developing web application using Spring framework and basic understanding of Oauth2.

Technologies Used

The technologies used in this guide are:

1. Spring Boot
2. Oauth2
3. MongoDB
4. Gradle

Dependencies

Here is the list of dependencies required:

```
dependencies {
        // Spring boot
        compile group: 'org.springframework.boot', name: 'spring-boot-starter-web', version: '2.

        //Spring Security Oauth2
        compile group: 'org.springframework.security.oauth', name: 'spring-security-oauth2', ver

        //Spring Data Mongo
        compile group: 'org.springframework.boot', name: 'spring-boot-starter-data-mongodb', ver
}
```

Spring Oauth2 - Dependencies hosted with ♥ by GitHub view raw

build.gradle

Authorization Server

Let us configure the authorization server. Let us register *rokin-client* as a client. We will be using in-memory token store.

```
package com.rokin.oauth2.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceCon
import org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServ
import org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizati
import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServer
import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServer
import org.springframework.security.oauth2.provider.token.TokenStore;
import org.springframework.security.oauth2.provider.token.store.InMemoryTokenStore;

@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfiguration extends AuthorizationServerConfigurerAdapter {

        @Autowired
        private AuthenticationManager authenticationManager;

        @Autowired
        private PasswordEncoder passwordEncoder;

```

```
26          @Autowired
27          private CustomUserDetailsService userDetailsService;
28
29          @Override
30          public void configure(AuthorizationServerSecurityConfigurer security) throws Exception {
31                  security.allowFormAuthenticationForClients();
32          }
33
34          @Override
35          public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
36                  clients.inMemory()
37                                  .withClient("rokin-client")
38                                  .secret(passwordEncoder.encode("secret"))
39                                  .authorizedGrantTypes("password", "client_credentials", "refresh_
40                                  .scopes("all")
41                                  .accessTokenValiditySeconds(3600)
42                                  .refreshTokenValiditySeconds(86400);
43          }
44
45          @Override
46          public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception
47                  endpoints.tokenStore(tokenStore())
48                          .authenticationManager(authenticationManager)
49                          .userDetailsService(userDetailsService);
50          }
51
52          @Bean
53          public TokenStore tokenStore() {
54                  return new InMemoryTokenStore();
55          }
56
57  }
```

Spring Oauth2 - Authorization Server Configuration hosted with ♥ by GitHub view raw

AuthorizationServerConfiguration.java

Here, allowFormAuthenticationForClients() method is used for authenticating a client using form parameters instead of basic auth.

Resource Server

```
1   package com.rokin.oauth2.security;
2
3   import org.springframework.context.annotation.Configuration;
4   import org.springframework.security.config.annotation.web.builders.HttpSecurity;
5   import org.springframework.security.oauth2.config.annotation.web.configuration.EnableResourceSer
6   import org.springframework.security.oauth2.config.annotation.web.configuration.ResourceServerCon
7   import org.springframework.security.oauth2.config.annotation.web.configurers.ResourceServerSecur
```

```
@Configuration
@EnableResourceServer
public class ResourceServerConfiguration extends ResourceServerConfigurerAdapter {

	private static final String RESOURCE_ID = "rokin-application";

	@Override
	public void configure(ResourceServerSecurityConfigurer resources) throws Exception {
		resources.resourceId(RESOURCE_ID);
	}

	@Override
	public void configure(HttpSecurity http) throws Exception {
		http.csrf().disable().authorizeRequests().anyRequest().authenticated();
	}
}
```

Spring Oauth2 - Resource Server Configuration hosted with ♥ by GitHub view raw

ResourceServerConfiguration.java

Security Configuration

```
package com.rokin.oauth2.security;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSec
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAda
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {


	@Override
	protected void configure(HttpSecurity http) throws Exception {
		http.csrf().disable().authorizeRequests().antMatchers("/oauth/token").permitAll(
	}

	@Bean
	@Override
	public AuthenticationManager authenticationManagerBean() throws Exception {
```

```
27                  return super.authenticationManagerBean();
28          }
29
30          @Bean
31          public PasswordEncoder passwordEncoder() {
32                  return new BCryptPasswordEncoder();
33          }
34
35  }
```

Spring Oauth2 - Security Configuration hosted with ♥ by GitHub view raw

SecurityConfiguration.java

Here, we will be using BCryptPasswordEncoder to encrypt user's password. The AuthenticationManagerBean is required for *password grant type*.

User Details Configuration

Now that we have configured the security settings, authorization server and resource server, let us dive into user configurations. We will be using MongoDB to store user details. Let us start by creating a User entity.

```
1   package com.rokin.oauth2.user.model;
2
3   import java.util.List;
4
5   import org.springframework.data.annotation.Id;
6   import org.springframework.data.mongodb.core.mapping.Document;
7
8   import com.fasterxml.jackson.annotation.JsonProperty;
9   import com.fasterxml.jackson.annotation.JsonProperty.Access;
10
11  @Document(collection = "users")
12  public class User {
13          @Id
14          private String id;
15          private String name;
16          private String username;
17          @JsonProperty(access = Access.WRITE_ONLY)
18          private String password;
19          private List<String> roles;
20
21          public User() {
22
23          }
24
25          public User(String name, String username, String password, List<String> roles) {
26                  this.name = name;
```

```
27                 this.username = username;
28                 this.password = password;
29                 this.roles = roles;
30         }
31
32         public String getId() {
33                 return id;
34         }
35
36         public void setId(String id) {
37                 this.id = id;
38         }
39
40         public String getName() {
41                 return name;
42         }
43
44         public void setName(String name) {
45                 this.name = name;
46         }
47
48         public String getPassword() {
49                 return password;
50         }
51
52         public void setPassword(String password) {
53                 this.password = password;
54         }
55
56         public String getUsername() {
57                 return username;
58         }
59
60         public void setUsername(String username) {
61                 this.username = username;
62         }
63
64         public List<String> getRoles() {
65                 return roles;
66         }
67
68         public void setRoles(List<String> roles) {
69                 this.roles = roles;
70         }
71
72         @Override
73         public String toString() {
74                 return "User [id=" + id + ", name=" + name + ", username=" + username + ", passw
75                                 + roles + "]";
76         }
77
```

```
78  }
```

Spring Oauth2 - User hosted with ♥ by GitHub view raw

User.java

Now, we will create a user repository to perform database queries on *users* collection. We will be using Spring Data MongoDB for this purpose.

```
1  package com.rokin.oauth2.user.repository;
2
3  import org.springframework.data.mongodb.repository.MongoRepository;
4
5  import com.rokin.oauth2.user.model.User;
6
7  public interface UserRepository extends MongoRepository<User, String> {
8
9          User findByUsername(String username);
10
11 }
```

Spring Oauth2 - UserRepository hosted with ♥ by GitHub view raw

UserRepository.java

For Spring Data MongoDB to work, we need to specify the database name, host, port, username and password in the application.properties file.

```
1  #MongoDB
2  spring.data.mongodb.database=security
3  spring.data.mongodb.host=localhost
4  spring.data.mongodb.port=27017
5  spring.data.mongodb.username=rokin
6  spring.data.mongodb.password=rokin
```

Spring Oauth2 - application.properties hosted with ♥ by GitHub view raw

application.properties

In order to verify that the credentials used to login are correct, we need to create a CustomUserDetailsService class which implements org.springframework.security.core.userdetails.UserDetailsService. In this class, we will fetch user from database and map it to org.springframework.security.core.userdetails.User.

```
1  package com.rokin.oauth2.security;
2
3  import java.util.ArrayList;
```

```
import java.util.Collection;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import com.rokin.oauth2.user.model.User;
import com.rokin.oauth2.user.repository.UserRepository;

@Service
public class CustomUserDetailsService implements UserDetailsService {

        @Autowired
        UserRepository userRepository;

        @Override
        public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException
                User dbUser = this.userRepository.findByUsername(username);

                if (dbUser != null) {
                        Collection<GrantedAuthority> grantedAuthorities = new ArrayList<>();

                        for (String role : dbUser.getRoles()) {
                                GrantedAuthority authority = new SimpleGrantedAuthority(role);
                                grantedAuthorities.add(authority);
                        }

                        org.springframework.security.core.userdetails.User user = new org.spring
                                        dbUser.getUsername(), dbUser.getPassword(), grantedAutho
                        return user;
                } else {
                        throw new UsernameNotFoundException(String.format("User '%s' not found",
                }
        }

}
```

Spring Oauth2 - CustomUserDetailsService hosted with ♥ by GitHub view raw

CustomUserDetailsService.java

Now that all of our configurations are done, let us insert a user in database during application startup. This is important because we cannot generate an access token using password grant without a user in our database.

```java
package com.rokin.oauth2;

import java.util.Arrays;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Lazy;
import org.springframework.security.crypto.password.PasswordEncoder;

import com.rokin.oauth2.user.model.User;
import com.rokin.oauth2.user.repository.UserRepository;

@SpringBootApplication
public class SpringOauth2Application implements CommandLineRunner {

	@Autowired
	private UserRepository userRepository;

	@Autowired
	private PasswordEncoder passwordEncoder;

	public static void main(String[] args) {
		SpringApplication.run(SpringOauth2Application.class, args);
	}

	@Override
	public void run(String... args) throws Exception {
		if (this.userRepository.findByUsername("rokin") == null) {
			User user = new User("Rokin Maharjan", "rokin", passwordEncoder.encode("

			this.userRepository.save(user);
		}
	}

}
```

Spring Oauth2 - SpringOauth2Application hosted with ♥ by GitHub view raw

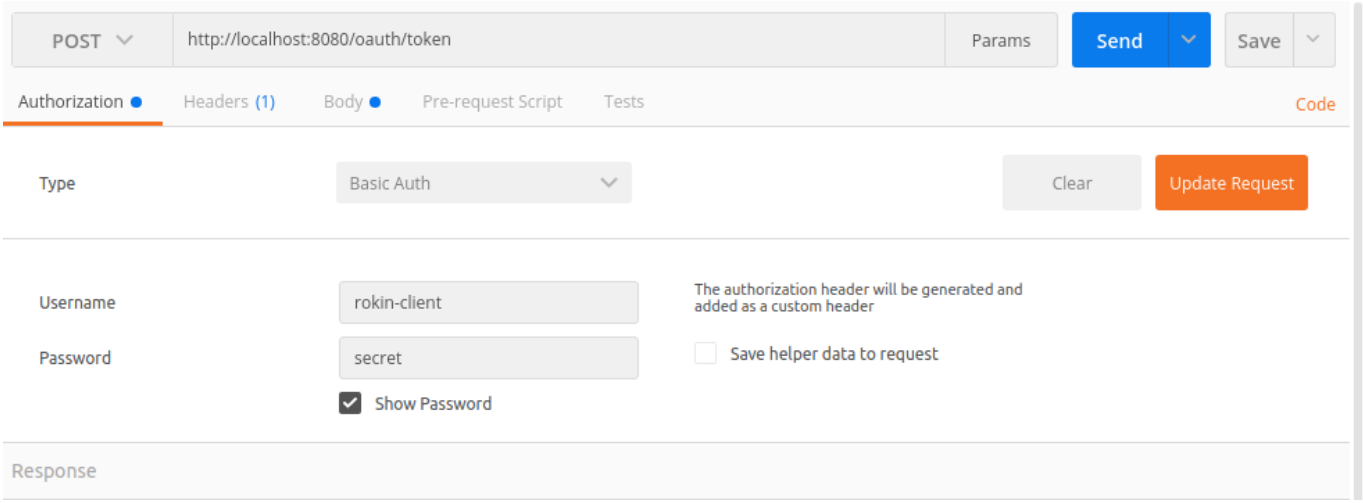
SpringOauth2Application.java

. . .

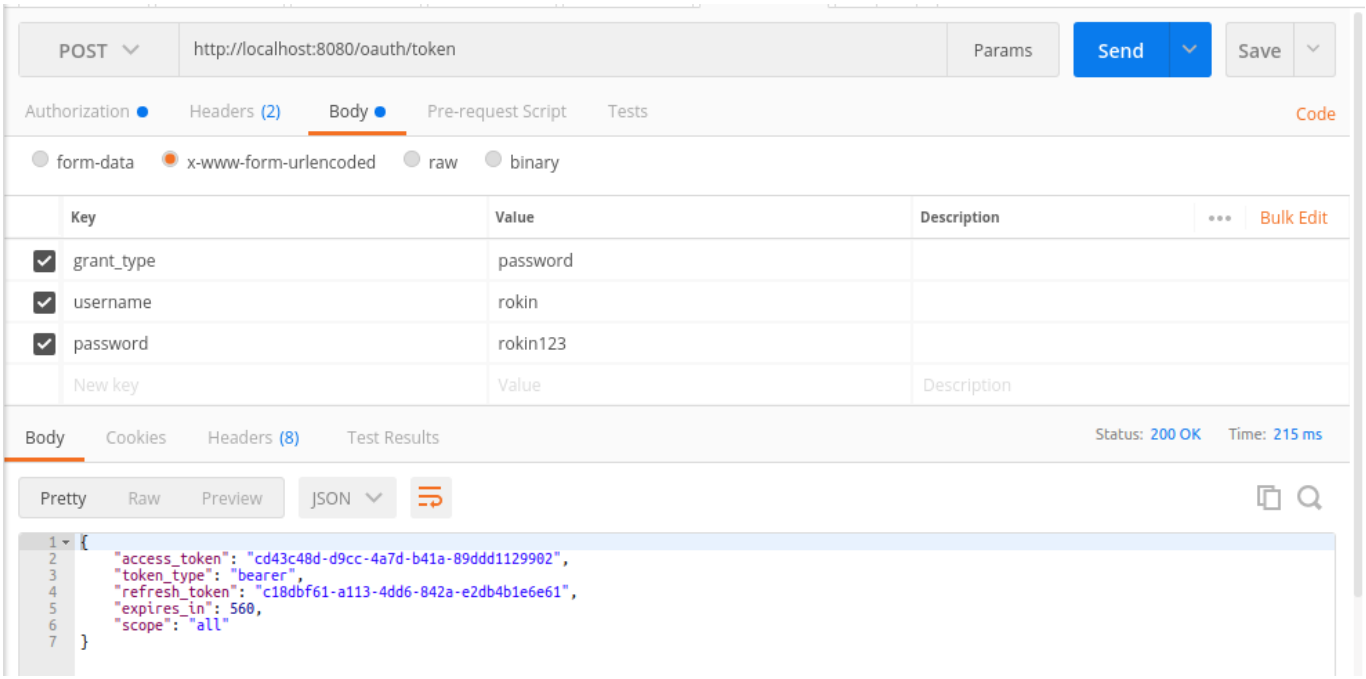
Token endpoints

Access and Refresh token

To get access and refresh token, first, update your request with your *client id* and *client secret* in the Authorization header.



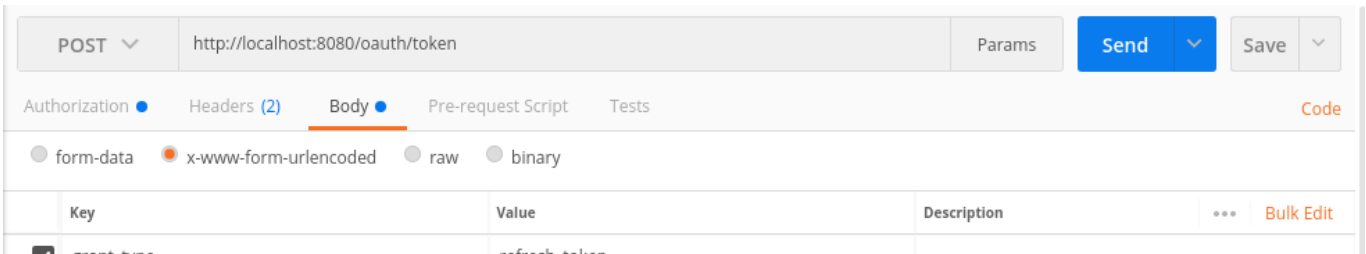
Then, in your body set ***grant_type*** as ***password*** and provide your ***username*** and ***password***.

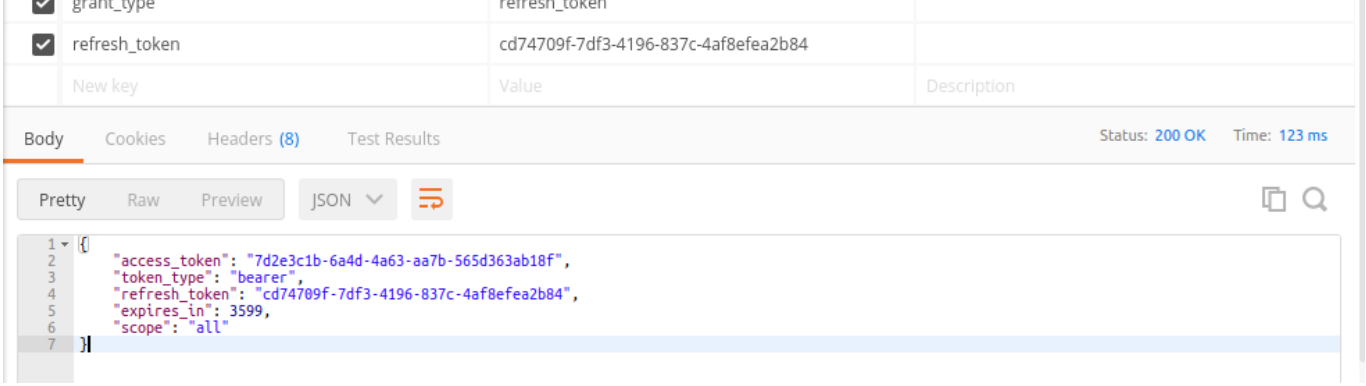


Access and Refresh token using password grant

Access Token from Refresh Token

Just as the when getting access token, first, update your request with your client id and client secret in the Authorization header. Then, in your body set ***grant_type*** as ***refresh_token*** and provide the ***refresh_token***.





Access and Refresh token using refresh_token grant

. . .

If you want to check out the complete project on github, please visit https://github.com/rokinmaharjan/spring-security-oauth2.

This is the end of this guide. Don't forget to clap if you liked it. :)

Oauth2 Spring Boot Mongodb