Securing OpenRules Decision Services

OpenRules, Inc.
www.openrules.com
August-2021

Table of Contents

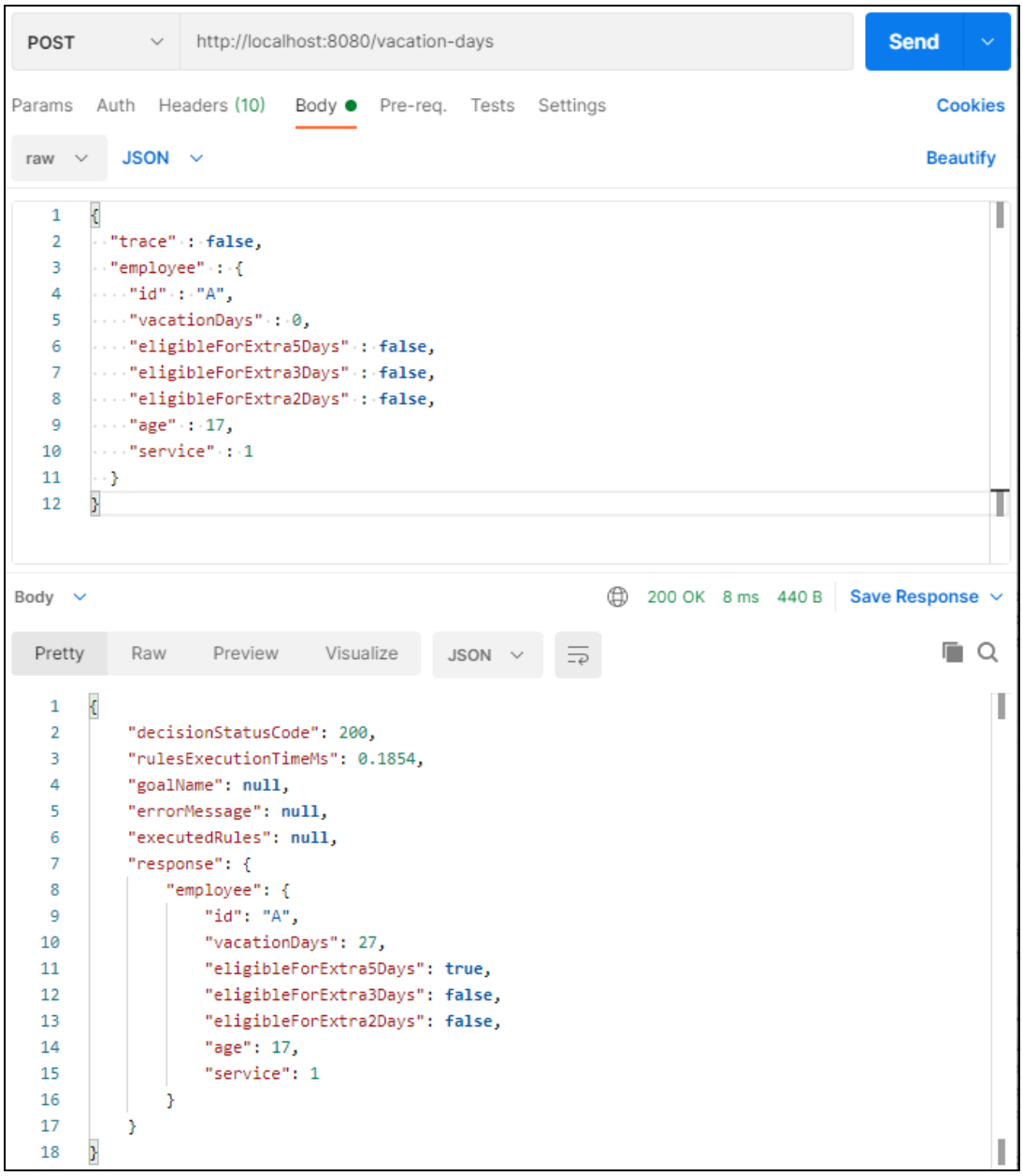
INTRODUCTION

OpenRules Decision Manager helps enterprises develop and maintain operational decision services that can be invoked from their decision-making business applications. The most popular Modern enterprises have very serious security requirements to any used RESTful services, and decision services obviously should follow established security protocols. In this manual, we describe how to secure SprintBoot-based OpenRules Decision Service using JWT Authentication and SSL communication.

CREATING SPRINGBOOT-BASED DECISION SERVICE

The standard OpenRules Decision Manager installation workspace "openrules.install" comes with a sample decision model "VacationDays" and several more projects that demonstrate how to deploy this model. You may look at the standard project "VacationDaysSpringBoot" that deploys the VacationDays decision model as a RESTful web service using the popular framework SpringBoot. You can find a detailed description of how to create and test this decision service in the User Manual for Developers.

You don't have to be a software expert to do it. You just add the property "***deployment=spring-boot***" to the file "project.properties" and double-click on the provided file "**runLocalServer.bat**". It will install the necessary software, build the decision model, and deploy it as a RESTful web service on the local server. You may test this RESTful decision service with Postman as shown below:



However, this RESTful decision service can be accessed from Postman or any client by knowing only its endpoint URL http://localhost:8080/vacation-days. It can be not acceptable for customers who have strong security requirements.

Securing Access to Decision Service with JWT Authentication

The most common way to secure access to the generated RESTful web service is to use JWT-based authentication. JWT (JSON Web Token) is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. In this section, we will demonstrate how to secure access to a SpringBoot-based Decision Service using JWT Authentication.

We will use the same decision model "VacationDays" incorporated into the new project "**VacationDaysSpringBootSecure**" included in the latest release (see "openrules.install"). This decision project was created based on the project "VacationDaysSpringBoot". First, we added all required dependencies to the file "**pom.xml**":

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
    <version>${spring.boot.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-oauth2-resource-server</artifactId>
    <version>${spring.security.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-oauth2-jose</artifactId>
    <version>${spring.security.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.security.oauth.boot</groupId>
    <artifactId>spring-security-oauth2-autoconfigure</artifactId>
    <version>${spring.boot.version}</version>
</dependency>
```

Then we changed the property "*model.package=vacation.days.springbootsecure*" in the file "project.properties". This Java package will include all generated and manually created Java classes. To configure a SpringBoot security service, we need to create a configuration class

SecurityConfig.java in the package *vacation.days.springbootsecure.service*. It is important to place this class into the folder "***service***" inside the same package. Here is its code:

```
package vacation.days.springbootsecure.service;

import org.springframework.beans.factory.annotation.Value;

@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Value("${spring.security.oauth2.resourceserver.jwt.issuer-uri}")
    private String issuer;

    @Override
    protected void configure(HttpSecurity http) throws Exception {

        http.csrf().disable();
        http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

        // secure access to all POST requests
        http.authorizeRequests()
                .antMatchers(HttpMethod.POST)
                .authenticated()
                .and()
                .oauth2ResourceServer()
                .jwt();
    }
}
```

Our project should also include the folder **src/main/resources/** in which we will place the following file "**application.yml**":

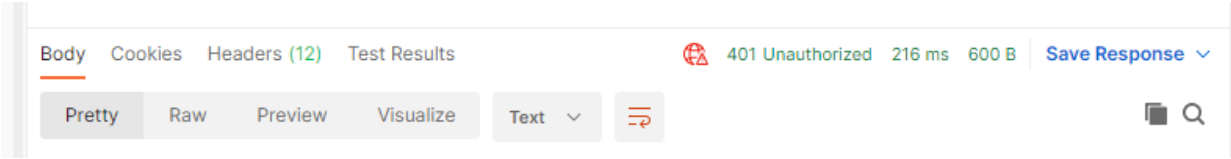
```
spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: https://cognito-idp.us-east-1.amazonaws.com/us-east-1_ed77oOAhi
```

Is structure corresponds to the parameter "*spring.security.oauth2.resourceserver.jwt.issuer-uri*" defined in the class SecurityConfig.

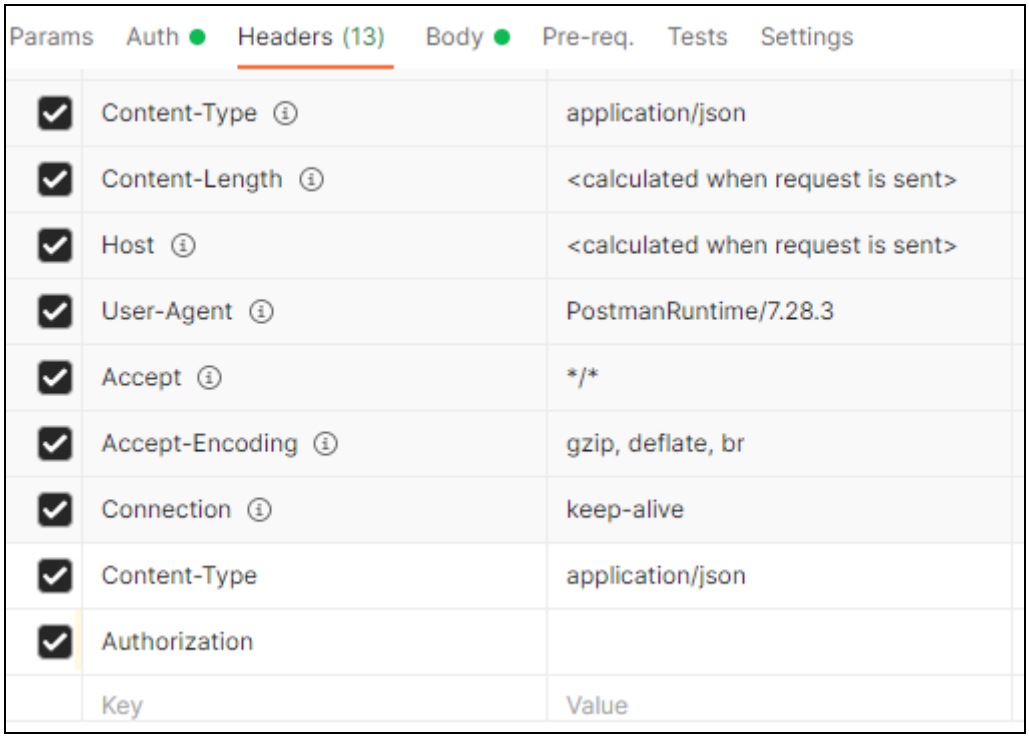
In this example, we use AWS Cognito as an authentication server. To use your authentication server, you need to consult with your IT department to find out a server URI that should be used to provide authentication for your decision service.

Now we are ready to run the standard file "**runLocalServer.bat**" as we did in the first section to create our RESTful decision service on our local server and to test it with POSTMAN. However,

when we try to use Postman to test this service, our “Send” request will be denied with the HTTP status code “401- Unauthorized”:

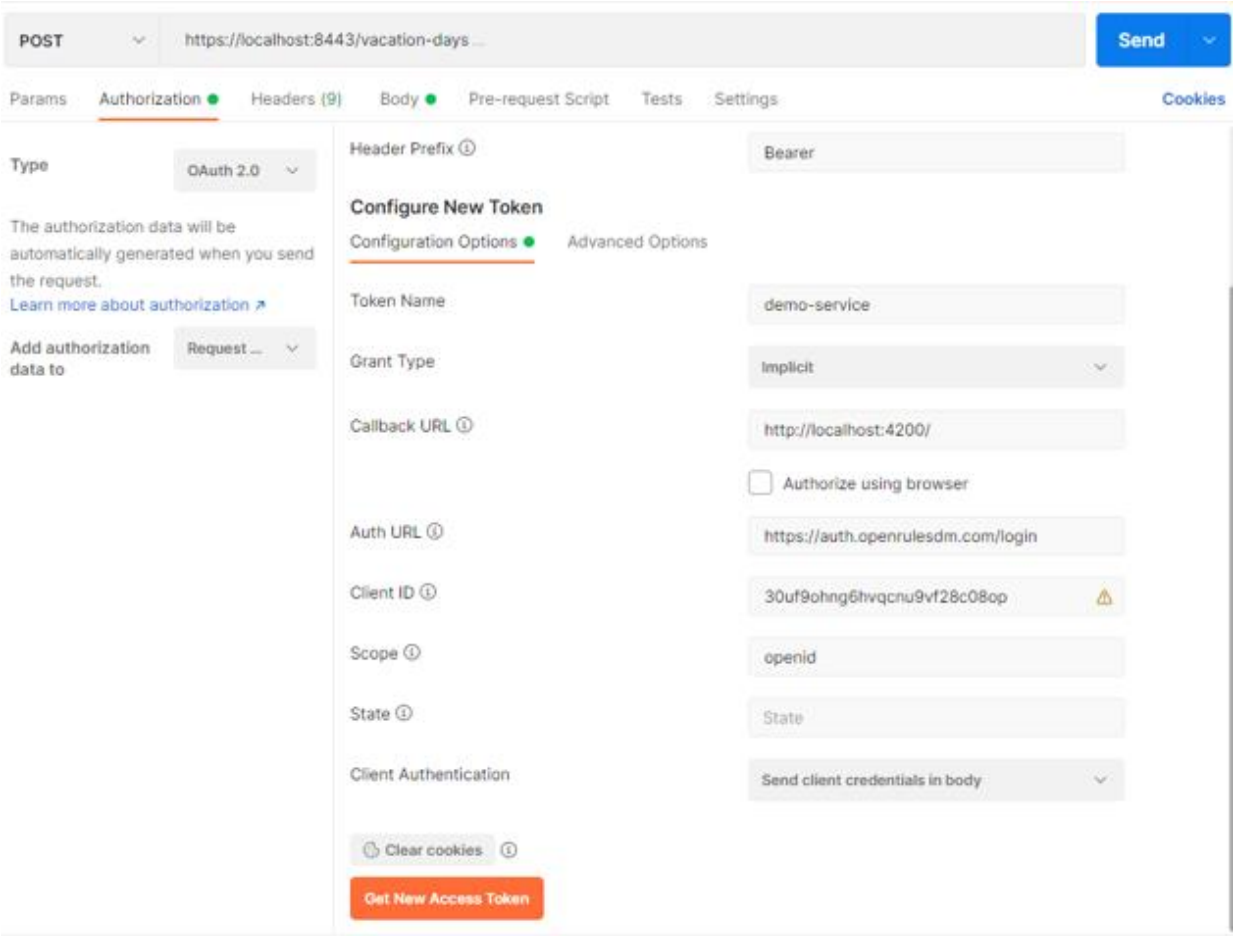


The reason is that we also need to configure Postman to make it be able to work with JWT Authentication. The detailed manual for how to do it can be found here. Here we will describe the simplest Postman JWT configuration option. Select the POSTMAN’s tab “Headers”, add a new key called “Authorization”:



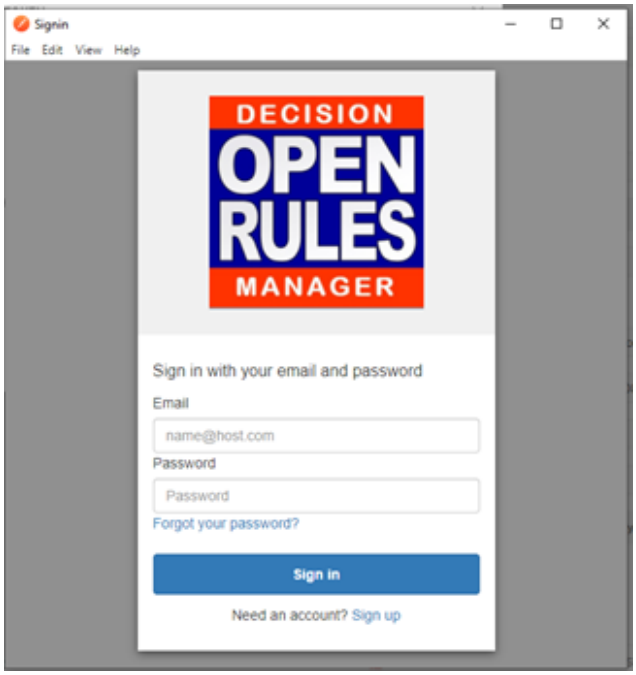
We need to enter an Authorization value in the format “Bearer <jwt-token>”, where <jwt-token> should be obtained from an authentication server. Alternatively,we can configure Postman to automatically request a JWT token from the authentication server.

Open Postman's tab "**Auth".** From the dropdown box '**Type**' select '**OAuth 2.0**'. From the dropdown '**Add authorization data to**' select '**Request Headers**'. Then fill in the section as shown below and click on the button "**Configure New Token**":



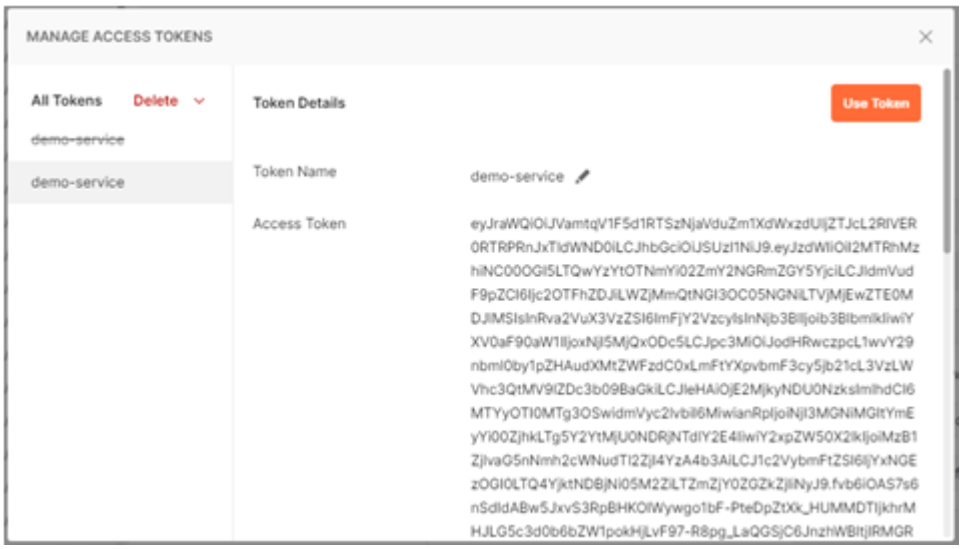
Here we use settings that are valid for this example only. We took Client ID from our AWS Cognito + User Pools + App Client settings.

When you click on the button "**Get New Access Token**", the following windows popups:



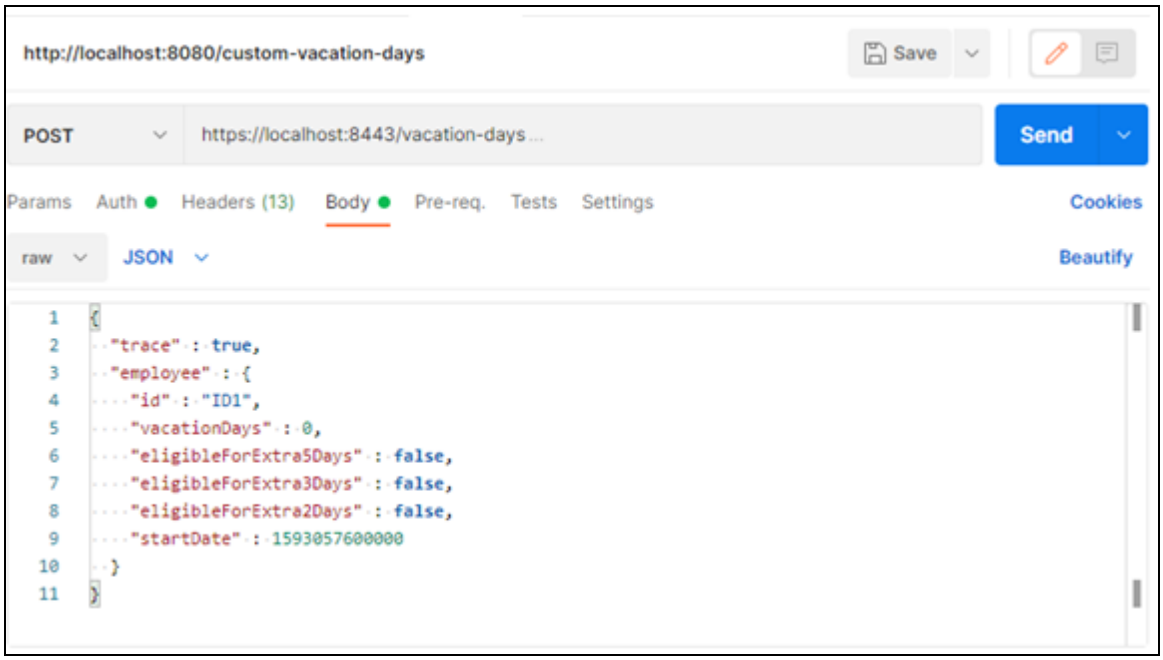
You need to enter your email and password which you have used to download OpenRules Decision Manager at https://download.openrulesdm.com.

After a successful login, the following token management window will appear:



Click on the button "**Use Token**". The access token will be retrieved from the authentication server's response. It will be automatically added to each Postman's request. The token is valid for an hour. After it expires you need to log in again and click on "Use Token".

After you provide a valid token, the request can be successfully executed by Postman. It will return a JSON result with HTTP status code 200 OK:



Now our decision service has been secured with JWT Authentication.

Enabling HTTPS for SpringBoot Decision Service REST Endpoint

In this section, we will explain how to configure our decision service to use encrypted communication between client and service with HTTPS protocol. To enable HTTPS, we need an

SSL certificate. For production, you should get a certificate issued by a certificate authority, but for the local testing and development, you can create a self-signed certificate using either the standard keytool shipped with Java JRE/JDK or OpenSSL. In this tutorial, we will use keytool. From the command line execute the following command:

```
>keytool -genkeypair -alias vacation-days-service -keyalg RSA -keysize 2048 -storetype PKCS12
-keystore vacation-days-service.p12 -validity 360
```

When asked, enter a password (in this example, we use **vacation-days-service**). Then answer all questions about your organization. The file **vacation-days-service.p12** will be generated by the keytool. Copy this file to the folder "**src/main/resources"**, in which we created the file "**application.yml**".

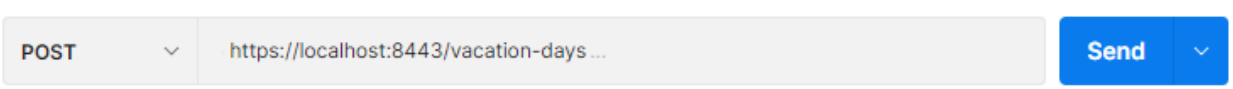
Now let's edit "**application.yml"** and configure the server's properties as below:

```
spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: https://cognito-idp.us-east-1.amazonaws.com/us-east-1_ed77oOAhi

server:
  port: 8443
  ssl:
    enabled: true
    key-store-type: PKCS12 # The keystore format.
    key-store: classpath:vacation-days-service.p12 # The path to the keystore
    key-store-password: vacation-days-service # The password used to generate the certificate
    key-alias: vacation-days-service # The alias mapped to the certificate

logging:
  level:
    root: INFO
    org.springframework: INFO
    com.openrules: ERROR
```

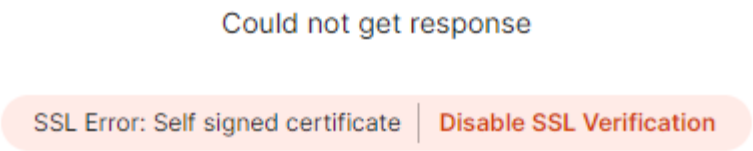
Now we can invoke **runLocalServer.bat.** It will show:

```
OpenRules Rest Decision Service for model VacationDays

    [POST] https://localhost:8443/vacation-days
```

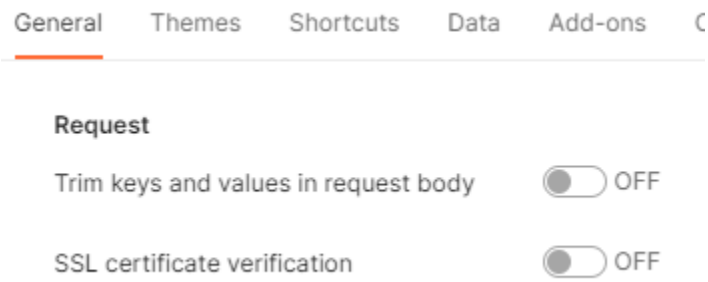
Let's test this secured decision service. In Postman change URL to https://localhost:8443/vacation-days and click on teh button "Send":



First, you may get the error "Could not get response" caused by our self-signed certificate:



Click on "Disable SSL Verification" or go to Postman's Settings/General screen and turn off SSL Verification:



Click on the button "Send" again and now you should see an expected JSON response from the service. Now our decision service uses a secure SSL communication.

Conclusion

This tutorial provides step-by-step instructions for how to secure SpringBoot-based OpenRules decision services. We provided specific examples:

1) Securing Access to Decision Service with JWT Authentication
2) Enabling HTTPS for Decision Service REST Endpoint.

While these examples utilized commonly used Spring Security, OpenRules decision services can be similarly secured inside any development and deployment environment used at your organization. If you have any issues securing OpenRules decision services, direct all your technical questions to support@openrules.com.