**REST with Java (JAX-RS) using Jersey – Tutorial**

# Jersey REST API Security Example

August 19, 2015 by Lokesh Gupta

In this example, we will learn to **secure Jersey REST APIs behind basic authentication** which will make mandatory to provide username/password by user. Also, user must have certain level of role as well. I have extended this example from my other example created for [RESTEasy API security](https://howtodoinjava.com/resteasy/jax-rs-2-0-resteasy-3-0-2-final-security-tutorial/) and used [ContainerRequestFilter](https://jax-rs-spec.java.net/nonav/2.0-SNAPSHOT/apidocs/javax/ws/rs/container/ContainerRequestFilter.html" \t "_blank)implementation to verify access of user before he land on actual REST API.

**Table of Contents**

[Building Request AuthenticationFilter](https://howtodoinjava.com/jersey/jersey-rest-security/" \l "build-auth-filter)

[Register AuthenticationFilter with ResourceConfig](https://howtodoinjava.com/jersey/jersey-rest-security/" \l "register-auth-filter)

[Secure REST APIs](https://howtodoinjava.com/jersey/jersey-rest-security/" \l "secure-rest-api)

[Test AuthenticationFilter](https://howtodoinjava.com/jersey/jersey-rest-security/" \l "demo)

[Sourcecode](https://howtodoinjava.com/jersey/jersey-rest-security/" \l "soucecode)

## Building Request AuthenticationFilter

you know that JAX-RS 2.0 has filters for pre and post request handling, so we will be using ContainerRequestFilterinterface. In this filter, we will get details of the method which request is trying to access. We will find-out all security related configuration on that method, and verify everything here in this filter e.g. annotation like @PermitAll, @DenyAll or @RolesAllowed.

According to annotation applied on methods, we will make the decision to pass or block the request.

|  |
| --- |
| package com.howtodoinjava.jersey.provider;    import java.lang.reflect.Method;  import java.util.Arrays;  import java.util.HashSet;  import java.util.List;  import java.util.Set;  import java.util.StringTokenizer;    import javax.annotation.security.DenyAll;  import javax.annotation.security.PermitAll;  import javax.annotation.security.RolesAllowed;  import javax.ws.rs.container.ContainerRequestContext;  import javax.ws.rs.container.ResourceInfo;  import javax.ws.rs.core.Context;  import javax.ws.rs.core.MultivaluedMap;  import javax.ws.rs.core.Response;  import javax.ws.rs.ext.Provider;    import org.glassfish.jersey.internal.util.Base64;    /\*\*   \* This filter verify the access permissions for a user   \* based on username and passowrd provided in request   \* \*/  @Provider  public class AuthenticationFilter implements javax.ws.rs.container.ContainerRequestFilter  {        @Context      private ResourceInfo resourceInfo;        private static final String AUTHORIZATION\_PROPERTY = "Authorization";      private static final String AUTHENTICATION\_SCHEME = "Basic";      private static final Response ACCESS\_DENIED = Response.status(Response.Status.UNAUTHORIZED)                                                          .entity("You cannot access this resource").build();      private static final Response ACCESS\_FORBIDDEN = Response.status(Response.Status.FORBIDDEN)                                                          .entity("Access blocked for all users !!").build();        @Override      public void filter(ContainerRequestContext requestContext)      {          Method method = resourceInfo.getResourceMethod();          //Access allowed for all          if( ! method.isAnnotationPresent(PermitAll.class))          {              //Access denied for all              if(method.isAnnotationPresent(DenyAll.class))              {                  requestContext.abortWith(ACCESS\_FORBIDDEN);                  return;              }                //Get request headers              final MultivaluedMap<String, String> headers = requestContext.getHeaders();                //Fetch authorization header              final List<String> authorization = headers.get(AUTHORIZATION\_PROPERTY);                //If no authorization information present; block access              if(authorization == null || authorization.isEmpty())              {                  requestContext.abortWith(ACCESS\_DENIED);                  return;              }                //Get encoded username and password              final String encodedUserPassword = authorization.get(0).replaceFirst(AUTHENTICATION\_SCHEME + " ", "");                //Decode username and password              String usernameAndPassword = new String(Base64.decode(encodedUserPassword.getBytes()));;                //Split username and password tokens              final StringTokenizer tokenizer = new StringTokenizer(usernameAndPassword, ":");              final String username = tokenizer.nextToken();              final String password = tokenizer.nextToken();                //Verifying Username and password              System.out.println(username);              System.out.println(password);                //Verify user access              if(method.isAnnotationPresent(RolesAllowed.class))              {                  RolesAllowed rolesAnnotation = method.getAnnotation(RolesAllowed.class);                  Set<String> rolesSet = new HashSet<String>(Arrays.asList(rolesAnnotation.value()));                    //Is user valid?                  if( ! isUserAllowed(username, password, rolesSet))                  {                      requestContext.abortWith(ACCESS\_DENIED);                      return;                  }              }          }      }      private boolean isUserAllowed(final String username, final String password, final Set<String> rolesSet)      {          boolean isAllowed = false;            //Step 1. Fetch password from database and match with password in argument          //If both match then get the defined role for user from database and continue; else return isAllowed [false]          //Access the database and do this part yourself          //String userRole = userMgr.getUserRole(username);            if(username.equals("howtodoinjava") && password.equals("password"))          {              String userRole = "ADMIN";                //Step 2. Verify user role              if(rolesSet.contains(userRole))              {                  isAllowed = true;              }          }          return isAllowed;      }  } |

## Register AuthenticationFilter with ResourceConfig

Now you will need to register above filter with ResourceConfig instance. So create an instance like below:

|  |
| --- |
| package com.howtodoinjava.jersey;    import org.glassfish.jersey.filter.LoggingFilter;  import org.glassfish.jersey.server.ResourceConfig;    import com.howtodoinjava.jersey.provider.AuthenticationFilter;  import com.howtodoinjava.jersey.provider.GsonMessageBodyHandler;    public class CustomApplication extends ResourceConfig  {      public CustomApplication()      {          packages("com.howtodoinjava.jersey");          register(LoggingFilter.class);          register(GsonMessageBodyHandler.class);            //Register Auth Filter here          register(AuthenticationFilter.class);      }  } |

And add this resource config in web.xml file.

|  |
| --- |
| <!DOCTYPE web-app PUBLIC   "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"   "<http://java.sun.com/dtd/web-app_2_3.dtd>" >    <web-app>        <display-name>Archetype Created Web Application</display-name>        <servlet>          <servlet-name>jersey-serlvet</servlet-name>          <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>          <init-param>              <param-name>javax.ws.rs.Application</param-name>              <param-value>com.howtodoinjava.jersey.CustomApplication</param-value>          </init-param>          <load-on-startup>1</load-on-startup>      </servlet>        <servlet-mapping>          <servlet-name>jersey-serlvet</servlet-name>          <url-pattern>/rest/\*</url-pattern>      </servlet-mapping>    </web-app> |

## Secure REST APIs

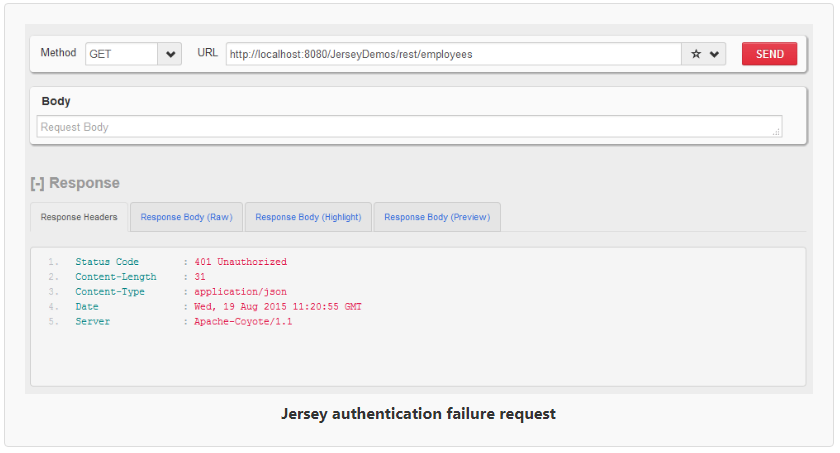
Now it’s time to secure the REST APIs. Use standard JAX-RS annotations for that like below.

|  |
| --- |
| @Path("/employees")  public class JerseyService  {      @RolesAllowed("ADMIN")      @GET      @Produces(MediaType.APPLICATION\_JSON)      @Consumes(MediaType.APPLICATION\_JSON)      public Employees getAllEmployees()      {          Employees list = new Employees();          list.setEmployeeList(new ArrayList<Employee>());            list.getEmployeeList().add(new Employee(1, "Lokesh Gupta"));          list.getEmployeeList().add(new Employee(2, "Alex Kolenchiskey"));          list.getEmployeeList().add(new Employee(3, "David Kameron"));            return list;      }  } |

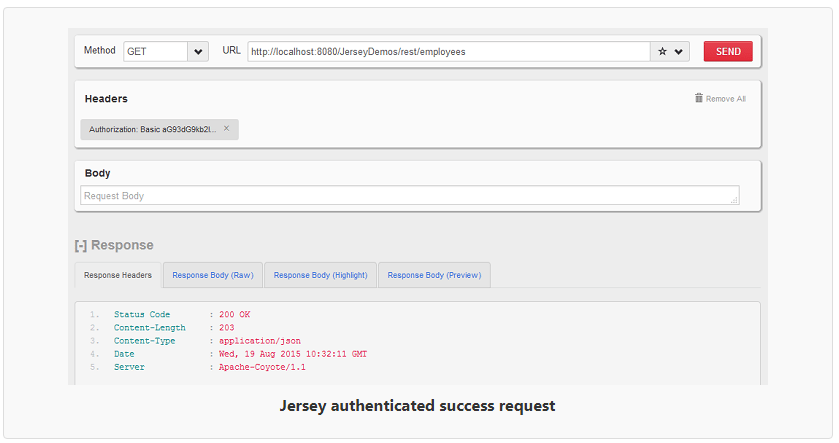
## Test AuthenticationFilter

Let’s test if authentication settings are working or not.

#### HIT URL : http://localhost:8080/JerseyDemos/rest/employees

Jersey authentication failure request

#### Pass username and password in basic auth parameters: howtodoinjava/password



Jersey authenticated success request