The **javax.ws.rs.core.SecurityContext** interface provides access to security-related information for a request. The **SecurityContext** provides functionality similar to **javax.servlet.http.HttpServletRequest**, enabling you to access the following security-related information:

* **java.security.Principal** object containing the name of the user making the request.
* Authentication type used to secure the resource, such as BASIC\_AUTH, FORM\_AUTH, and CLIENT\_CERT\_AUTH.
* Whether the authenticated user is included in a particular role.
* Whether the request was made using a secure channel, such as HTTPS.

You access the **SecurityContext** by injecting an instance into a class field, setter method, or method parameter using the**javax.ws.rs.core.Context** annotation.

For more information, see the Javadoc at:

* **SecurityContext** interface: [**http://docs.oracle.com/javaee/6/api/index.html?javax/ws/rs/core/SecurityContext.html**](http://docs.oracle.com/javaee/6/api/index.html?javax/ws/rs/core/SecurityContext.html)
* **@Context** annotation: [**http://docs.oracle.com/javaee/6/api/index.html?javax/ws/rs/core/Context.html**](http://docs.oracle.com/javaee/6/api/index.html?javax/ws/rs/core/Context.html)

[Example 5-3](http://docs.oracle.com/middleware/1212/wls/RESTF/secure-restful-service.htm#BABJDBHI) shows how to inject an instance of **SecurityContext** into the **sc** method parameter using the **@Context** annotation, and check whether the authorized user is included in the **admin** role before returning the response.

***Example 5-3 Securing RESTful Web Service Using SecurityContext***

package samples.helloworld;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

**import javax.ws.rs.core.SecurityContext;**

**import javax.ws.rs.core.Context;**

...

@Path("/stateless")

@Stateless(name = "JaxRSStatelessEJB")

public class StlsEJBApp {

...

@GET

@Produces("text/plain;charset=UTF-8")

@Path("/hello")

public String sayHello(**@Context SecurityContext sc**) {

if (**sc.isUserInRole("admin")**) return "Hello World!";

throw new SecurityException("User is unauthorized.");

}

Securing RESTful Web Services Using Java Security Annotations

The **javax.annotation.security** package provides annotations, defined in [Table 5-1](http://docs.oracle.com/middleware/1212/wls/RESTF/secure-restful-service.htm#BABBGCDE), that you can use to secure your RESTful Web services. For more information, see:

* ["Specifying Authorized Users by Declaring Security Roles"](http://docs.oracle.com/javaee/6/tutorial/doc/bnbyl.html#gjgcq) in *The Java EE Tutorial*.
* [**javax.annotation.security** Javadoc](http://docs.oracle.com/javaee/6/api/index.html?javax/annotation/security/package-summary.html)

***Table 5-1 Annotations for Securing RESTful Web Services***

| **Annotation** | **Description** |
| --- | --- |
| **DenyAll** | Specifies that no security roles are allowed to invoke the specified methods. |
| **PermitAll** | Specifies that all security roles are allowed to invoke the specified methods. |
| **RolesAllowed** | Specifies the list of security roles that are allowed to invoke the methods in the application. |

[Example 5-4](http://docs.oracle.com/middleware/1212/wls/RESTF/secure-restful-service.htm#BABJFDAJ) shows how to define the security roles that are allowed, by default, to access the methods defined in the **helloWorld** class. The sayHello method is annotated with the **@RolesAllows** annotation to override the default and only allow users that belong to the **ADMIN** security role.

***Example 5-4 Securing RESTful Web Service Using Java Security Annotations***

package samples.helloworld;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

**import javax.annotation.Security.RolesAllowed;**

@Path("/helloworld")

**@RolesAllowed({"ADMIN", "ORG1"})**

public class helloWorld {

@GET

@Path("sayHello")

@Produces("text/plain")

@**RolesAllows("ADMIN")**

public String sayHello() {

return "Hello World!";

}

}

## Securing RESTful Web Services Using web.xml

You secure RESTful Web services using the **web.xml** deployment descriptor as you would for other Java EE Web applications. For complete details, see:

* ["Developing Secure Web Applications"](http://www.oracle.com/pls/topic/lookup?ctx=fmw121200&id=SCPRG141) in *Developing Applications with the WebLogic Security Service*.
* "Securing Web Applications" in *The Java EE 6 Tutorial* at: [**http://docs.oracle.com/javaee/6/tutorial/doc/gkbaa.html**](http://docs.oracle.com/javaee/6/tutorial/doc/gkbaa.html)

For example, to secure your RESTful Web service using basic authentication, perform the following steps:

1. Define a **<security-constraint>** for each set of RESTful resources (URIs) that you plan to protect.
2. Use the **<login-config>** element to define the type of authentication you want to use and the security realm to which the security constraints will be applied.
3. Define one or more security roles using the **<security-role>** tag and map them to the security constraints defined in step 1. For more information, see ["security-role"](http://www.oracle.com/pls/topic/lookup?ctx=fmw121200&id=SCPRG180) in *Developing Applications with the WebLogic Security Service*.
4. To enable encryption, add the **<user-data-constraint>** element and set the **<transport-guarantee>** subelement to **CONFIDENTIAL**. For more information, see ["user-data-constraint"](http://www.oracle.com/pls/topic/lookup?ctx=fmw121200&id=SCPRG186) in *Developing Applications with the WebLogic Security Service*.

The following shows an example of how to secure a RESTful Web service using basic authentication.

***Example 5-2 Securing RESTful Web Services Using Basic Authentication***

<web-app>

<servlet>

<servlet-name>RestServlet</servlet-name>

<servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>RestServlet</servlet-name>

<url-pattern>/\*</url-pattern>

</servlet-mapping>

<security-constraint>

<web-resource-collection>

<web-resource-name>Orders</web-resource-name>

<url-pattern>/orders</url-pattern>

<http-method>GET</http-method>

<http-method>POST</http-method>

</web-resource-collection>

<auth-constraint>

<role-name>admin</role-name>

</auth-constraint>

</security-constraint>

<login-config>

<auth-method>BASIC</auth-method>

<realm-name>default</realm-name>

</login-config>

<security-role>

<role-name>admin</role-name>

</security-role>

</web-app>