**Annotation Driven Approach:**

|-Advantages to using annotations over config based Security

|-Disadvantages of annotations over config based Security

Java EE defines a common set of annotations that can define authorization metadata. The JAX-RS specification suggests, but does not require, vendor implementations to support these annotations in a non–Java EE 6 environment. These annotations live in the javax.annotation.security package and are @RolesAllowed, @DenyAll, @PermitAll, and @RunAs.

The @RolesAllowed annotation defines the roles permitted to execute a specific operation. When placed on a JAX-RS annotated class, it defines the default access control list for all HTTP operations defined in the JAX-RS class. If placed on a JAX-RS method, the constraint applies only to the method that is annotated.

The @PermitAll annotation specifies that any authenticated user is permitted to invoke our operation.

As with @RolesAllowed, we can use this annotation on the class to define the default for the entire class or we can use it on a per-method basis. Let’s look at an example:

@Path("/customers")

@RolesAllowed({"ADMIN", "CUSTOMER"})

public class CustomerResource {

@GET

@Path("{id}")

@Produces("application/xml")

public Customer getCustomer(@PathParam("id") int id) {

...

}

@RolesAllowed("ADMIN")

@POST

@Consumes("application/xml")

public void createCustomer(Customer cust) {

...

}

@PermitAll

@GET

@Produces("application/xml")

public Customer[] getCustomers() {

....

}

}

Our CustomerResource class is annotated with @RolesAllowed to specify that, by default, only ADMIN and CUSTOMER users can execute HTTP operations and paths defined in that class. The getCustomer() method is not annotated with any security annotations, so it inherits this default behavior.

The createCustomer() method is annotated with @RolesAllowed to override the default behavior. For this method, we only want to allow ADMIN access. The getCustomers() method is annotated with @PermitAll. This overrides the default behavior so that any authenticated user can access that URI and operation.

**Advantages to using annotations over config based Security:**

There are some advantages to using annotations, though. For one, it is a workaround for doing finegrained constraints that are just not possible in web.xml because of the limited expression capabilities of <url-pattern>. Also, because we can apply constraints per method using these annotations, we can fine-tune authorization per media type.

For example:

@Path("/customers")

public class CustomerService {

@GET

@Produces("application/xml")

@RolesAllowed("XML-USERS")

public Customer getXmlCustomers() {

.....

}

@GET

@Produces("application/json")

@RolesAllowed("JSON-USERS")

public Customer getJsonCustomers() {

....

}

}

Here we only allow XML-USERS to obtain application/xml content and JSON-USERS to obtain application/json content. This might be useful for limiting users in getting data formats that are expensive to create.

By using annotations we easily apply the security compared to config approach bcz if we wanted allow multiple roles of users then we need to configure 2-times <security-role> but if we go for annotations we can configure easily with less no.of lines of code.

**Disadvantages of annotations over config based Security:**

In practice, we don’t like to specify security metadata using annotations. Security generally does not affect the behavior of the business logic being executed and falls more under the domain of configuration.

Administrators may want to add or remove role constraints periodically. We don’t want to have to recompile our whole application when they want to make a simple change. So, if we can avoid it, we usually use web.xml to define my authorization metadata.

**Note:**

To work annotation based security we need to configure context-param which vendor specific.

<context-param>

<param-name>resteasy.role.based.security</param-name>

<param-value>true</param-value>

</context-param>

Similarly try for JERSEY

Example:

Access the application:

Case: 1 Success Case

http://localhost:8082/2.3AnnotationSecurityRESTEasy/rest/trading/buy

Select POST

Content-Type: application/xml

Req Body:

<buyOrder>

<customerID>2</customerID>

<stockName>cypla</stockName>

<quatity>24</quatity>

<exchange>bse</exchange>

</buyOrder>

Login as

Username= robin

password= welcome

Response:

Status Code: 200 (OK)

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<invoice><invoiceNo>ce8fab39-fb0b-4b63-9952-cfb00bccab93</invoiceNo>

<status>partial</status>

</invoice>

Login as

Username= john

password= welcome

Case: 2 Success Case

http://localhost:8082/2.3AnnotationSecurityRESTEasy/rest/trading/status/1

Select GET

Response:

Status Code: 200 (OK)

Status: active

Case: 3 Failure Case:

http://localhost:8082/2.3AnnotationSecurityRESTEasy/rest/trading/buy

Select POST

Content-Type: application/xml

Req Body:

<buyOrder>

<customerID>2</customerID>

<stockName>cypla</stockName>

<quatity>24</quatity>

<exchange>bse</exchange>

</buyOrder>

Login as

Username= david

password= welcome

Response:

Status Code: 401: Unauthorized