12

Web Service Security

Objectives

After completing this lesson, you should be able to do the following:

- Explain Authentication, Authorization, and Confidentiality
- Apply Basic Java EE Security by using deployment descriptors (web.xml)
- Create users and groups and map them to application roles
- Apply JSR-250 Security annotations
- Enable an assortment of filters including the `RolesAllowedResourceFilterFactory`
- Obtain a SecurityContext and perform programmatic security
- Authenticate using the Jersey Client API

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

- Lesson 1: Introduction to Web Services
- Lesson 2: Creating XML Documents
- Lesson 3: Processing XML with JAXB
- Lesson 4: SOAP Web Services Overview
- Lesson 5: Creating JAX-WS Clients

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

Lesson 11: Web Service Error Handling

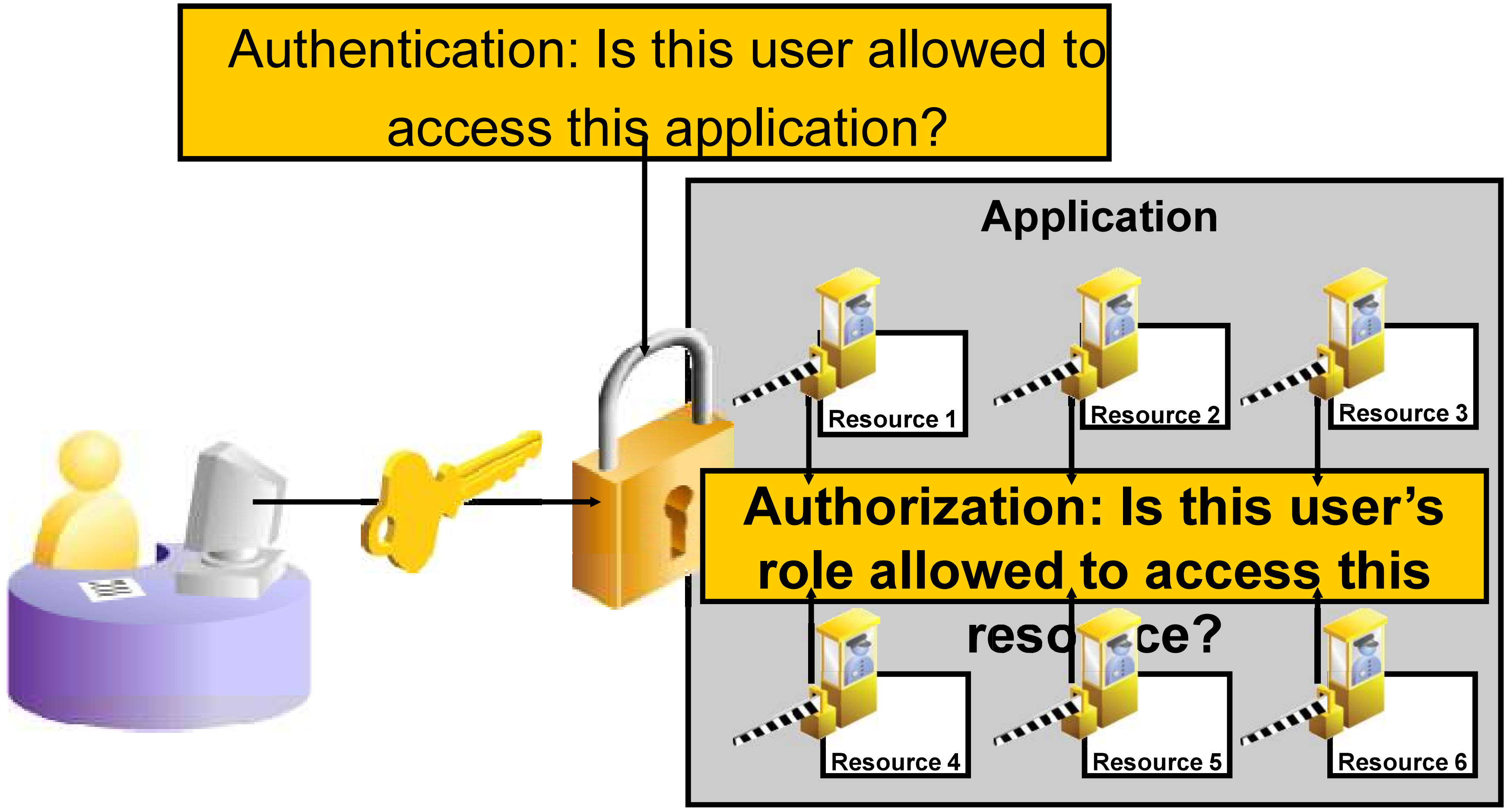
Lesson 12: Java EE Security and Securing JAX WS

You are here!

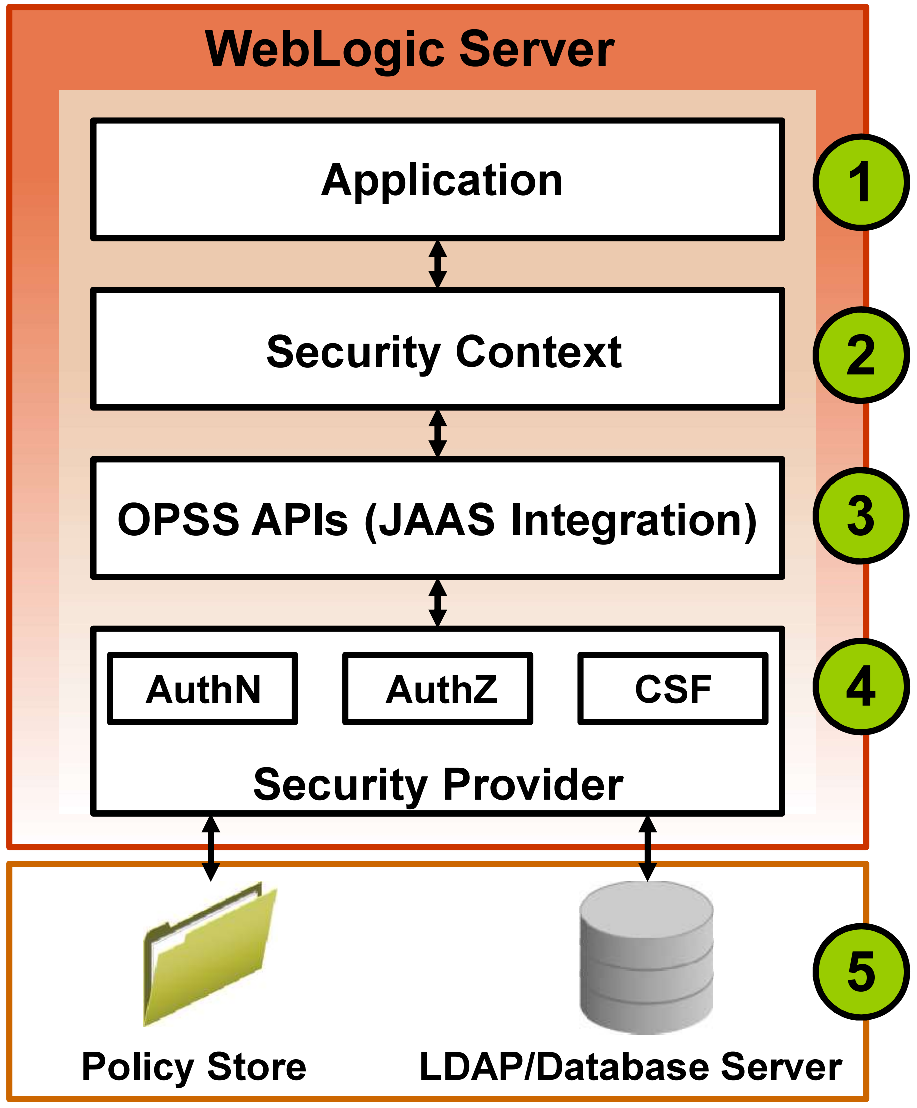
Benefits of Securing Web Applications

- WebService applications often connect with a single database user account. Therefore, separate application users accounts must be utilized.
- Identity can be used to:
  - Ensure that only authenticated users can access the application
  - Restrict access to parts of the application
  - Customize the UI (such as pick lists)
  - Provide the user name for auditing
  - Set up a virtual private database (VPD)

Examining Security Aspects

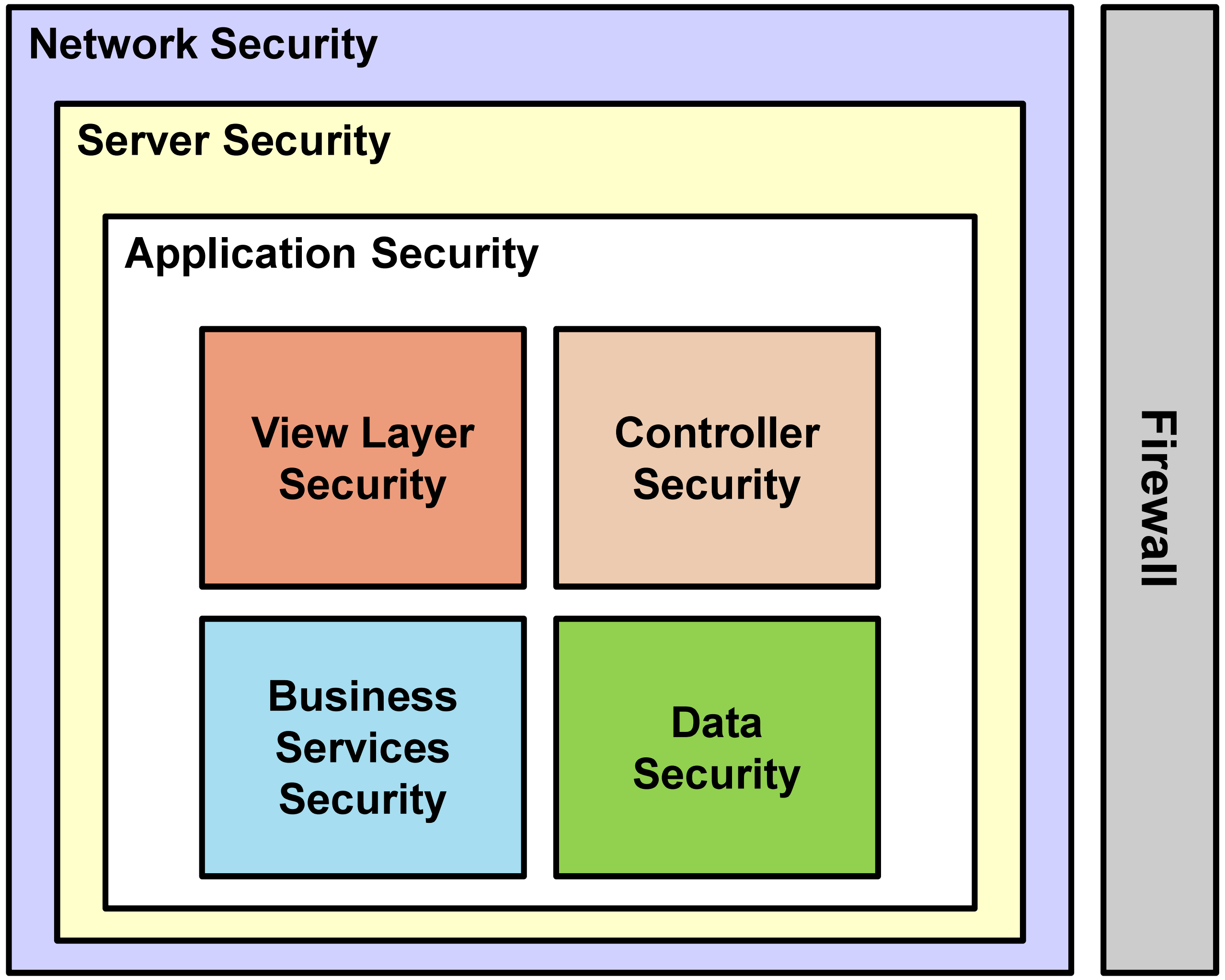

Authentication: Is this user allowed to access this application?
Application
Resource 1
Resource 2
Resource 3
Authorization: Is this user's role allowed to access this resource?
Resource 4
Resource 5
Resource 6

Security Framework and OPSS



1. Sends request to authenticate or authorize user for a specific page or resource
2. Passes the request to OPSS
3. Delegates to an authentication and authorization provider
4. Redirects request to an identity management system.
5. Provides authentication and authorization services

Securing the Layers of an Application



Authorization

- Authorization is the process of determining if a caller is allowed to perform an action.
- Relies on authentication to verify identity.
- Resources are restricted by:
  - Security annotations such as `@RolesAllowed` and `@ServletSecurity`
  - Elements in deployment descriptors (web.xml, ejb-jar.xml)
  - Application logic (programmatic security)

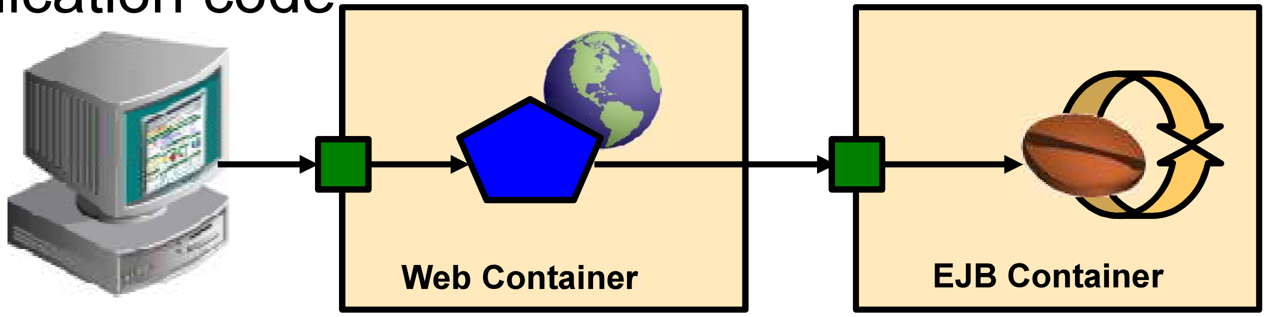
Data Integrity

- Received data has not been modified, destroyed, or lost
- Data integrity problems can result from unauthorized data access or accidental mishap
- Data in a web service exchange is defined as all or part of a SOAP message, including the SOAP header element and attachment parts
- Two subcategories include:
  - Transport Data Integrity
  - SOAP Message Integrity

Container-Managed Security

The security model in the Java EE platform is primarily an authorization model.

- If required, the container authenticates the client.
- The container checks a client's rights to carry out the requested action on a component.
- After the authorization is complete, the container invokes application code.



Web-Tier Authentication Challenge

- HTTP Basic: The web browser prompts the user for a username and password, and supplies this information in the request header.
- Client Certificate: The client presents the user’s digital certificate in response to a challenge from the server.
- Form-based: The developer controls the look and feel of the authentication process by supplying HTML forms.
- Programmatic: Technically, this is not a challenge method. Java EE 6 includes programmatic servlet security, which adds the ability to force a challenge in-code or to collect authentication information in a custom way.

Interaction With the Security Infrastructure

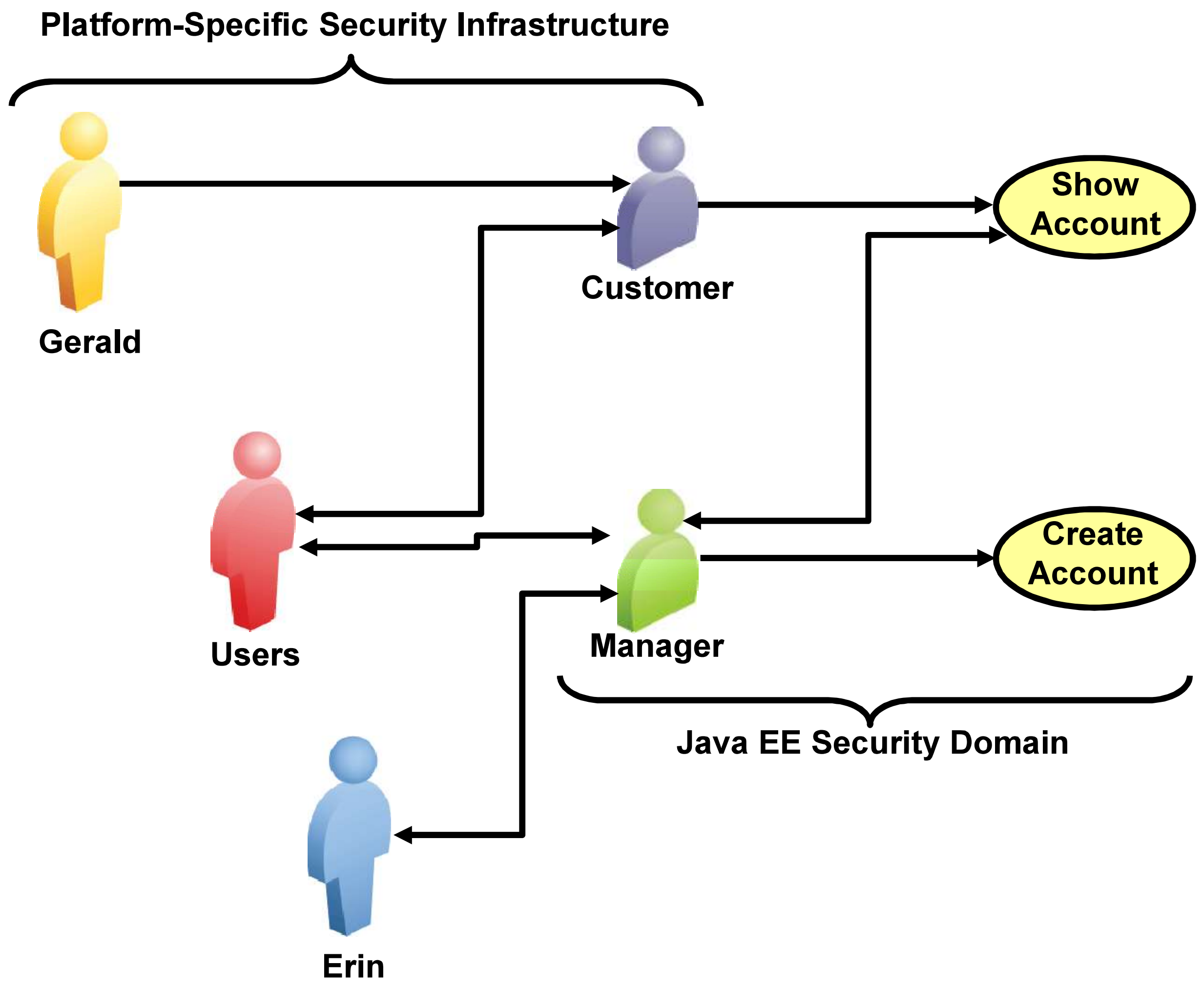
- Remember, the security model in the Java EE platform is vendor-neutral and platform-independent.
- User credentials and permissions are stored in various ways, such as directory servers and relational database tables.
- The application server interacts with the security infrastructure. Applications cannot do this without loss of portability.
- The range of security infrastructures supported by the application server can be extended by the use of JAAS/JACC modules.

Roles and Responsibilities

A role is an abstraction of a set of user authorization privileges.

- Users in the same role have broadly similar rights and responsibilities.
- The role structure of the security model in the Java EE platform is flat, not hierarchical.
- Individual users can, and often will, occupy more than one role.
- There is some correspondence between a role and a group in many security infrastructures, but the mapping of real users or groups to roles is platform specific.

Role-Based Java EE Security Model



javax.annotation.security Common Annotations

JSR-250, Common Annotations for the Java Platform, defines annotations that are used by varied types of Java EE components including web services.

- `@RunAs("admin")` – Regardless of who calls the annotated resource, runs as the listed role
- `@RolesAllowed({"user", "admin"})` – Limits the allowed callers to users in the listed roles
- `@PermitAll` – Permits all callers. Typically used on a method when `@RolesAllowed` is at the class level.
- `@DenyAll` – Denies all callers

JAX-WS EJB endpoints, JAX-RS EJB endpoints, and JAX-RS POJO endpoints can use JSR-250 annotations. JAX-WS POJO endpoint cannot use JSR-250 annotations.

Role Mapping

A web application will globally declare the roles that will be used in an application. These roles should be mapped using a vendor specific descriptor file.

```
<security-role>
    <description>meaningful text</description>
    <role-name>users</role-name>
</security-role>
<security-role>
    <role-name>admin</role-name>
</security-role>
```

Role Mapping

Mapping application roles to the principal accounts or groups that exist within the application server is done with a vendor deployment descriptor file(s). The `WEB-INF/weblogic.xml` file for web components in WebLogic Server is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-web-app
  <security-role-assignment>
    <role-name>user</role-name>
    <principal-name>joe</principal-name>
  </security-role-assignment>
</weblogic-web-app>
```

`web.xml` Security Constraints

URL patterns can be restricted via `web.xml`.

```
<security-constraint>
    <display-name>MembersOnly</display-name>
    <web-resource-collection>
      <web-resource-name>secret-page</web-resource-name>
      <url-pattern>/faces/membersonly.xhtml</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>member</role-name>
    </auth-constraint>
</security-constraint>
```

`web.xml` Security Constraints

Specific HTTP methods can be restricted.

```
<security-constraint>
    <display-name>MembersOnly</display-name>
    <web-resource-collection>
      <web-resource-name>secret-page</web-resource-name>
      <url-pattern>/faces/membersonly.xhtml</url-pattern>
      <http-method>GET</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>member</role-name>
    </auth-constraint>
</security-constraint>
```

RolesAllowedResourceFilterFacto

`RolesAllowedResourceFilterFactory` enables the use of `@RolesAllowed`, `@PermitAll`, `@DenyAll`, and `@RunAs`. Without it the security annotations will not function in JAX-RS resource classes.

```
<init-param>
    <param-name>com.sun.jersey.spi.container.ResourceFilters</param-name>
     <param-value>
    com.sun.jersey.api.container.filter.RolesAllowedResourceFilterFactory
    </param-value>
</init-param>
```

Applied to the Jersey Servlet in `web.xml`.

Filtering by Annotation (Step 1)

A ResourceFilter is just a factory for container filters. The benefit of a ResourceFilter is that it can be applied at the method and class levels using the `@ResourceFilters` annotation.

```
@ResourceFilters({LoggerLoggingResourceFilter.class})
@GET
@Produces("text/html")
public String getXml() {
    return "<html><body><h1>Hello " + name +
        "!</h1></body></html>";
}
```

Custom class

Filtering by Annotation (Step 2)

```
public      class      LoggerLoggingResourceFilter      implements
   ResourceFilter {
    private @Context ProviderServices providerServices;
    private @Context ResourceConfig rc;
    private static final Logger logger =
        Logger.getLogger("jersey");

    @PostConstruct
    private void init() {
        rc.getFeatures()
       .put(LoggingFilter.FEATURE_LOGGING_DISABLE_ENTITY,
           false);
    }
```

Context Providers

JAX-RS uses its own form of dependency injection, the @Context annotation. Only the listed types are required to be supported by JAX-RS.

- `@Context Application`
- `@Context UriInfo`
- `@Context HttpHeaders`
- `@Context Request`
- `@Context SecurityContext`
- `@Context Providers`
- `@Context ServletConfig`
- `@Context ServletContext`
- `@Context HttpServletRequest`
- `@Context HttpServletResponse`

Retrieving Security Information

```
@GET
public String get(@Context SecurityContext secContext) {
   if(secContext.getUserPrincipal() != null) {
       return "AuthenticationScheme: " +
               secContext.getAuthenticationScheme() +
               ", Principal: " +
               secContext.getUserPrincipal().getName() +
               ", isSecure: " +
               secContext.isSecure() +
               ", isUserInRole(\"person\"): " +
               secContext.isUserInRole("person");
    } else {
       return "not logged in";
   }
}
```

Servlet 3.0 Programmatic Login

By obtaining HttpServletRequest, a JAX-RS resource class can perform programmatic login and logout.

```
@POST
public String post(@Context HttpServletRequest request,
  @QueryParam("user") String user,
  @QueryParam("password") String pass) throws ServletException {
    request.login(user, pass);
    return "ok";
}
```

```
@DELETE
public String delete(@Context HttpServletRequest request)
                                        throws ServletException {
    request.logout();
    return "ok";
}
```

Authenticating Jersey Client

```java
public class AuthenticatingJerseyClient {
 static public void main( String[] args ) {
   String contextURL = "http://localhost:8080/jaxrs";
   String resourcePath = "/airports";
   String requestPath = "/numAirports";
   String urlString =
    contextURL + resourcePath + requestPath;
   Client client = Client.create();
   ClientFilter authFilter =
    new HTTPBasicAuthFilter("login", "password");
   client.addFilter(authFilter);
   WebResource resource =
    client.resource( urlString );
   String result = resource.get( String.class );
```

Quiz

Which methods are available in a JAX-RS SecurityContext?

a. getUserPrincipal()
b. isUserInRole("role")
c. getCallerPrincipal()
d. isCallerInRole("role")

Resources

Topic	Website
Jersey User Guide	http://jersey.java.net/nonav/documentation/latest/user-guide.html

Summary

In this lesson, you should have learned how to:

- Explain Authentication, Authorization, and Confidentiality
- Apply Basic Java EE Security by using deployment descriptors (web.xml)
- Create users and groups and map them to application roles
- Apply JSR-250 Security annotations
- Enable an assortment of filters including the `RolesAllowedResourceFilterFactory`
- Obtain a SecurityContext and perform programmatic security
- Authenticate using the Jersey Client API

Practice 12 : Overview

This practice covers the following topics:

- Securing a JAX-WS Endpoint with WS-Security
- Using Java EE Roles and Principles