CONTENTS INCLUDE:

Getting Started with Java EE Security

By Masoud Kalali

SECURITY IN JAVA EE APPLICATIONS

Java EE security supports a fine set of security functionalities in the specification level. These capabilities include authentication, authorization, data integrity and transport security. Before going deep into the Java EE security, everyone should know the following terms:

A **User** is an individual identity which is defined in the identity storage. The storage which can be an RDBMS, flat file or LDAP server contains user attributes like username and password.

A **Group** is a set of users classified with a set of common characteristics which usually lead to a set of common permissions and access levels.

A **Security Realm** is the access channel for the application server to storage containing user's authentication and grouping information.

A **Role** is a Java EE concept to define access levels. A Java EE application developer specifies which roles can access which set of the application functionalities. These roles are then mapped to users and groups using the vendor specific configuration files.

A **Principal** is an identity with known credentials which can be authenticated using an authentication protocol.

A **Credential** contains or references information used to authenticate a principal for Java EE product services. Password is a simple credential used for authentication.

Different application servers use different methods to map users, groups and roles to each other.

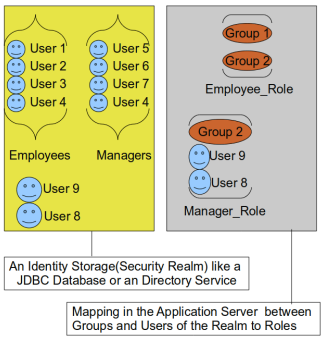

Figure 1: A further illustration of the roles, users and groups concept

Authentication and authorization in Java EE

A Java EE application server consists of multiple containers including the Web/ Servlet container, an EJB container and finally an **Application Client Container (ACC)**.

The Web container as the door to EJB container performs authentication and propagates the authenticated subject to EJB container. EJB container then performs the authorization prior to letting the EJB invocation go through.

When EJB container is accessed by an application client, EJB container itself performs authentication and authorization on the credentials collected and provided by the ACC.

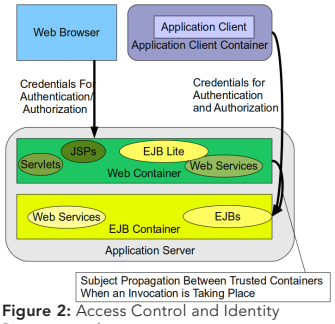

Figure 2: Access Control and Identity Propagation between containers

Web and EJB containers host different sets of resources and therefore each one of them has its separate authorization mechanism suitable for its deployed components.

Each Java EE application can consist of multiple modules as shown in Figure 2. Each one of these modules can have zero or more deployment descriptors which can contain different types of configuration for the application components (JSPs, Servlets, EJBs, Etc.) including but not limited to their security descriptions. Figure 3 shows these files and their locations.

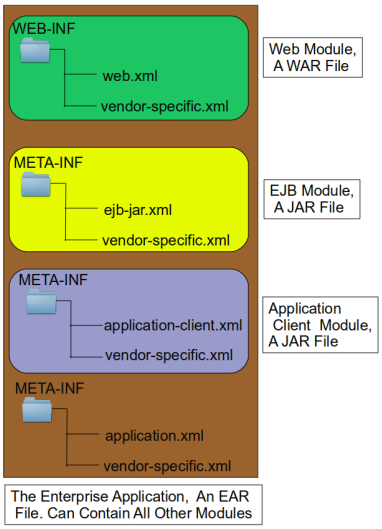

Figure 3: Java EE application modules and their deployment descriptors

Although all of the vendor specific deployment descriptors are in XML format that is not a requirement.

Hot Tip In Java EE 6, more annotations are introduced which we can use to plan the application deployment and we can override any standard Java EE annotation used in the source code using the corresponding Java EE deployment descriptors elements.

WEB MODULE SECURITY

In the Web module we can apply authentication, authorization and transport level encryption using the provided annotations and deployment descriptors.

Authentication and Authorization in Web Module

The following snippet instructs the application server to only let

manager role to access a resource with a URL matching /mgr/* in our Web application.

Listing 1: Declaring security constraint in web.xml

```
<security-constraint>
        <display-name>mgr resources</display-name>
        <web-resource-collection>
            <web-resource-name>managers</web-resource-name>
            <description/>
            <url-pattern>/mgr/*</url-pattern>
            <http-method>GET</http-method>
            <http-method>PUT</http-method>
            <http-method>POST</http-method>
        </web-resource-collection>
        <auth-constraint>
            <description/>
            <role-name>manager</role-name>
        </auth-constraint>
</security-constraint>
<security-role>
   <description>All Manager </description>
    <role-name>manager</role-name>
</security-role>
```

We defined a security constraint, defined a collection of resources matching the `/mgr/*` URL and defined a constraint over the `GET`, `PUT`, and `POST` methods. Then we permitted the manager role to access the constrained resources. The URL pattern can specify anything from a Servlet to a set of JSP files. We can include as many roles as we need in the `auth-constraint` element.

Any security role referenced in the `auth-constraint` elements should be defined using a security-role element as we did for the manager role.

So far we told which role has access to the secured resource but we still need to let the application server know how to authenticate the users and later on how to determine which roles the user has.

Java EE containers provide some standard authentication mechanisms for using in the Web modules. These methods with their specification names are as follow:

(1) HTTP Basic Authentication: `BASIC`
(2) Digest Authentication: `DIGEST`
(3) HTTPS Client Authentication: `CLIENT-CERT`
(4) Form-Based Authentication: `FORM`

In the first two methods container initiates an HTTP basic authentication and usually the Web client (Browser) shows the standard dialog to collect the user name and the password. The only difference is that when using `DIGEST`, client sends a digest of the password instead of sending it in clear text. To use any of these methods we only need to include the following snippet in the web.xml.

```
<login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>file-realm</realm-name>
</login-config>
```

In the `CLIENT-CERT` method, clients authenticate the server by asking the server for its digital certificate and the server also asks the client to provide its digital certificate to authenticate its identity. In this mode nothing is required to be done except that the client and the server must have a certificate issued by a certificate authority trusted by the other side.

Finally the `FORM` method lets the developer have more control over authentication by letting them provide their own credentials collecting pages. So we basically create a login and login-err page and let the application server know about our pages. Application server will use these pages to collect the user credentials and verifying them. To use this method we should include the following snippet into the `web.xml`.

```
<login-config>
        <auth-method>FORM</auth-method>
        <realm-name>file-realm</realm-name>
        <form-login-config>
            <form-login-page>auth/login.jsp</form-login-page>
            <form-error-page>auth/login-error.jsp</form-error-page>
        </form-login-config>
 </login-config>
```

The simplest content for the login.jsp is as follow:

```
<form action="j_security_check" method="POST">
<input type="hidden" name="A" value="1">
<input type="hidden" name="B" value="2">
</form>
```

The `login-error.jsp` page can contain any sort of information you feel necessary for the users to understand they provided wrong credentials and they can probably recover the password and so on.

Now it is time to let the application server know where the users credentials are stored so it can authenticate the received credentials with them and decide whether the user is authentic or not. This is where vendor specific deployment descriptor comes into play. Basically we need to map the roles we used in the standard deployment descriptor to users and groups (in some cases users and roles) defined in security realm. Different vendors use different configuration elements to map roles to individual users and groups of users in the security realm. The following table shows how a role can be mapped to users and groups in different application servers.

Application Server	Mapping Sample

GlassFish: sun-web.xml	`<security-role-mapping>`
	`<role-name>manager</role-name>`
	`<principal-name>JoneDoe</principal-name>`
	`<group-name>managers</group-name>`
	`</security-role-mapping>`

Geronimo: geronimo-web.xml	`<security-realm-name>file-realm</security-realm-name>`
	
	`<security>`
	`<role-mapping>`
	`<role role-name="manager">`
	`<principal class="org.apache.geronimo.security.realm.providers.GeronimoUserPrincipal" name="JohnDoe" />`
	`<principal class="org.apache.geronimo.security.realm.providers.GeronimoGroupPrincipal" name="managers" />`
	`</role>`
	`</role-mapping>`
	`</security>`
	
	The realm need to be created as a top level realm in the application server management console or it can be added to the web application as a deployment module. Using the second way, the security realm will be deployed along with the application and will be underployed when we undeploy the application.

JBoss: jboss-web.xml	The JBoss `case` is different because JBoss uses the concept of Security Domain which is `defined` in a separate descriptor file named jboss-web.xml and is located in the conf directory of server instance. Following element is used to specify the security domain. `<security-domain>java:/jaas/jboss-sec-domain</security-domain>` For JBoss application server some of the declaration we specified in `*-web.xml` is moved to `login-config.xml` which includes both role mappings and security realm definition. A security domain can be deployed with the enterprise application itself or it can be defined in the global `login-conf.xml` file. A sample security domain definition is shown in the listing 2.

Table 1: Application server specific role mappings

Listing 2: Sample security domain for JBoss

```
<application-policy name="jboss-sec-domain">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.DatabaseServerLoginModule"
flag="required">
      <module-option name="dsJndiName">java:/user-data-source</module-option>
      <module-option name="principalsQuery">select passwd from users where
userid=?</module-option>
      <module-option name="rolesQuery">select roleid, 'Roles' from roles where
userid=?</module-option>
    </login-module>
  </authentication>
</application-policy>
```

This sample domain specifies that the user information is stored in a database which is accessible through a data source named user-data-source. Two other elements specify how a username can be searched in the users table and how the associated roles can be extracted from the roles table.

So far we specified how we can perform authentication using the container provided features. Now we need to conduct access control or authorization.

Hot Tip: Passwords and user names are not protected from eavesdropping when we use FORM or BASIC authentication methods. To protect them from being viewed and intercepted by third parties we should enforce transport security.

Enforcing Transport Security

Transport security ensures that no one can tamper with the data being sent to a client or data we receive from a client. Java EE specification lets us enforce the transport security in two levels.

CONFIDENTIAL: By using SSL, this level guarantees that our data is encrypted so that it cannot be deciphered by third parties and the data remains confidential.

INTEGRAL: By using SSL, this level guarantees that the data will not be modified in transit by third parties.

NONE: This level does not apply SSL, and lets the data transport happen as usual.

We can enforce transport security in web.xml using the user-data-constraint element which we should place inside the security-constraint tag containing the resource which need transport protection. For example we can add the following snippet inside the security-constraint of Listing 1 to enforce use of SSL when user is accessing manager resources.

```
<user-data-constraint>
 <transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
```

Hot Tip: We can define as many security-constraints as required and each one of them can use a different user-data-constraint level.

When we specify `CONFIDENTIAL` or `INTEGRAL` as transport guarantee level, the application server will use the HTTPS listener (HTTP listener with SSL enabled) to communicate with client. Different application servers use a variety of methods to define and configure the HTTPS listeners. Each listener will have a dedicated port like 8080 for HTTP and 8181 for HTTPs.

Hot Tip: In production environment we usually front the application server with a Web server or a dedicated hardware appliance to accelerate the SSL access among other tasks like hosting static content, load distribution, decorating HTP headers and so on.

For the security purpose the front end Web server or appliance (like a Cisco PIX 535, F5 Big IP, etc) can be used to accelerate SSL certificate processing, unify the access port to both HTTP and HTTPS, act as a firewall and so on.

Other Security Elements of Web application deployment descriptors

Other elements which we can use in web.xml for security purposes which are listed in the Table 2.

Element	Description

`security-role`	Each role must be referenced in a security-role tag before it can be used in the auth-constraint element of a security-constraint. For example:
	`<security-role>`
	`<description>All Manager </description>`
	`<role-name>manager</role-name>`
	`</security-role>`

`session-config`	To specify for how long a session should remain valid. For example:
	`<session-config>`
	`<session-timeout>120</session-timeout>`
	`</session-config>`

`run-as`	To use an specific internal role for any out going call from the Servlet.
	`<run-as>`
	`<role-name>internal_role</role-name>`
	`</run-as>`
	This element resides inside the Servlet tag.

`security-role-ref`	We can alias a role with a more meaningful title and then link the alias to real realm using this element. For example:
	`<security-role-ref>`
	`<role-name>mid_level_managers</role-name>`
	`<role-link>manager</role-link>`
	`</security-role-ref>`

Table 2: Complete list of Security related elements of web.xml

Hot Tip: We use the run-as element or its counterpart annotation to assign a specific role to all outgoing calls from a Servlet or an EJB. We use this element to ensure that an internal role which is required to access some secured internal EJBs is never assigned to a client and stays fully in control of the developers.

Using Annotations to enforce security in Web modules

We can use annotations to enforce security in a Web module. For example, we can specify which roles can access a Servlet by adding some annotations in the Servlet or we can mark a method in a Servlet stating that no one can access it.

List of all Java EE 6 annotations and their descriptions are available in the Table 3.

Annotation	Description
`@DeclareRoles`	Prior to referencing any role, it should be defined. The `@DeclareRoles` acts like security-role element in defining the roles used in application.
`@RunAs`	Specifies the run-as role for the given Components.
`@ServletSecurity`	The `@ServletSecurity` can optionally get a `@HttpMethodConstraint` and `@HttpConstraint` as its parameters. The `@HttpMethodConstraint` is an array specifying the HTTP methods specific constraint while `@HttpConstraint` specifies the protection for all HTTP methods which are not specified in the `@HttpMethodConstraint`.
`@PermitAll`	Permitting users with any role to access the given method, EJB or Servlet
`@DenyAll`	If placed on a method, no one can access that method. In case of class level annotation, all methods of annotated EJB are inaccessible to all roles unless a method is annotated with a `@RolesAllowed` annotation.
`@RolesAllowed`	In case of method level annotation, it permits the included roles to invoke the method. In case of class level annotation, all methods of annotated EJB are accessible to included roles unless the method is annotated with a different set of roles using `@RolesAllowed` annotation.

Table 3: Security Annotations in Java EE 6

Each of the annotations included in table 3 can be placed on different targets like methods, classes or both and on different Java EE components like Servlets and EJBs. Table 4 shows what kind of targets are supported for each one of these annotations.

Annotation	Targets Level	Target Kind
`@DeclareRoles`	Class	EJB, Servlet
`@RunAs`	Class	EJB, Servlet
`@ServletSecurity`	Class	Servlet
`@PermitAll`	Class, Method	EJB
`@DenyAll`	Method	EJB
`@RolesAllowed`	Class, Method	EJB

Table 4: Security Annotation targets in Java EE 6

Some of the security annotations can not target a method like `@DeclareRoles` while some other can target both methods and classes like `@PermitAll`. Annotation applied on a method will override the class level annotations. For example if we apply `@RolesAllowed("employee")` on an EJB class, and we apply `@RolesAllowed("manager")` on one specific method of that EJB, only admin role will be able to invoke the marked method while all other methods will be available to the employee role.

Hot Tip: A role can be mapped to one or more specific principals, groups, or to both of them. The principal or group names must be valid in the specified security realm. The role name we use in the mapping element must match the role-name in the security-role element of the deployment descriptor [web.xml, ejb-jar.xml] or the role name defined in the @DeclareRoles annotation.

Programmatic Security in Web Module

We can access some of the container security context programmatically from our Java source code. Table 5 shows the seven methods of `HTTPServletRequest` class which we can use to extract security related attributes of the request and decide manually about how to process the request.

Method	Descriptions
`String getRemoteUser()`	If the user is authenticated returns the username otherwise returns null.
`boolean isUserInRole(String role)`	Returns whether the current user has the specified roles or not.
`Principal getUserPrincipal()`	Returns a `java.security.Principal` object containing the name of the current authenticated user.
`String getAuthType()`	Returns a String containing authentication method used to protect this application.
`void login(String username, String password)`	This method authenticates the provided username and password against the security realm which the application is configured to use. We can say this method does anything that the `BASIC` or `FORM` authentication does but gives the developer total control over how it is going to happen.
`Void logout()`	Establish null as the value returned when `getUserPrincipal`, `getRemoteUser`, and `getAuthType` is called on the request.
`String getScheme()`	Returns the schema portion of the URL, for example HTTP or HTTPS.

Table 5: Programmatic Security functionalities in Web modules

The following snippet shows how we check the user role and decide where to redirect him.

```
protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {

        if (request.isUserInRole(“manager”)) response.sendRedirect(“/mgr/index.
jsp”);
        else  response.sendRedirect(“/guests/index.jsp”);
    }
```

This snippet demonstrates the use of login method to programmatically login a user using the container security.

```
String userName = request.getParameter(“user”);
String password = request.getParameter(“password”);

try {
request.login(userName, password);
}catch(ServletException ex) {
//Handling Exception
    return;
}
```

In the sample code, which can happen inside the doGet or doPost of a Servlet we are extracting the username and password from the request and then we use the login method to ask the container to authenticate the given username and password against the configured realm.

EJB MODULE SECURITY

Like Web Container and Web module we can enforce security on EJB modules as well.

In an EJB module we can enforce security (Authentication & Authorization) on Entity Beans and Session Beans. No Security enforcement for the MDBs.

In figure 1 you can see that we either access the EJBs through Web container or the ACC. In the first method, the Web container conducts the authentication and propagate the subject to EJB container when using EJBs. In the second method, the ACC performs authentication and pass on the subject during context initialization to the EJB container for authorization.

EJB module deployment descriptors

Each EJB module has one or more deployment description which contains standard EJB module deployment elements and vendor specific information.

During this section we assume we have an Entity Bean named Employee as follows:

Listing 3: Sample Employee EJB

```
@Entity
public class Employee implements Serializable {
    public String getName() {
        return “name”;
    }
    public void promote(Position toPosition) {
        //promote the emplyee
    }

    public List<EvaluationRecords> getEvaluationRecords() {
        List<EvaluationRecords> evalRecord;
        //return a list containing all
        // EvaluationRecords of
        //this employee
        return evalRecord;
    }
    public List<EvaluationRecords> getEvaluationRecords(Date from, Date to)
{
        List<EvaluationRecords> evalRecord;
        //return a list containin all
        //productivity evaluation of
        //this employee in the given time range
        return evalRecord;
    }
    @Id
    private Integer id;
    public Employee() {
    }
}
```

Now in the standard deployment descriptor we have can have something like the following snippet to restrict execution of the Employee Bean methods to certain roles:

Listing 4: Enforcing access restriction on EJB methods invocation

```
<method-permission>
    <role-name>manager</role-name>
    <method>
        <ejb-name>Employee</ejb-name>
        <method-name>getName</method-name>
    </method>
</method-permission>
<method-permission>
    <role-name>manager</role-name>
    <method>
        <ejb-name>employee</ejb-name>
        <method-name>getEvaluationRecords</method-name>
        <method-params>
            <method-param>from</method-param>
            <method-param>to</method-param>
        </method-params>
    </method>
</method-permission>

<method-permission>
    <role-name>hr_manager</role-name>
    <method>
        <ejb-name>Employee</ejb-name>
        <method-name>*</method-name>
    </method>
</method-permission>
```

This snippet should be placed inside the EJB declaration to invoke any method of the EJB under the given role.

The snippet is instructing EJB container to allow any subject with `manager` role to invoke `getName` method, and only the `getEvaluationRecords` overloads which takes a date range. Then it allows any subject with `hr_manager` role to invoke all methods of the `Employee` EJB.

Like `web.xml` we will need to include role definitions in the deployment descriptor. So we will need to add three `security-role` elements in the `ejb-jar.xml` file to define the roles we are using. The syntax is the same as `web.xml` element which is included in listing 1.

We said that the EJB module performs authentication only when it is accessed from ACC and all configurations for the authentication is provided by the vendor specific deployment descriptors.

Vendor specific deployment descriptors for EJB module are below.

Application Server	Description
GlassFish: sun-ejb-jar.xml	(1) Define the role mapping using the same syntax used in the sun-web.xml (2) Adding transport security on EJBs using the ior-security-config subelement of the ejb element. (3) Adding Web Services security declarations using webservice-endpoint subelement of the ejb element. (4) Specifying the authentication realm for EJBs when they are accessed by the ACC using: `<ior-security-context> <as-context> <realm>myrealm</realm> </as-context> </ior-security-context>`
Geronimo: openejb-jar.xml	(1) Adding role mapping using the same syntax as geronimo-web.xml (2) Adding web services security using web-service-security subelement of the enterprise-beans element. (3) Adding the security realm using the same syntax as geronimo-web.xml
JBoss: jboss.xml	(1) Specifying the security domain similar to jboss-web.xml (2) Adding EJB transport security using the ior-security-config subelement of the message-driven, ejb, service, and session elements.

Table 6: Different application server's EJB deployment descriptors

Security Annotation of EJB modules in Java EE 6

Java EE provides a rich set of security related annotations for the EJB modules. Each of these annotations can be applied on one or more types, as explained previously in Table 3 and Table 4.

Following snippet shows how we can use these annotations to apply the same security restrictions we declared in the deployment descriptor showed in listing 4 on the entity source code shown in Listing 3.

Listing 5: Using annotations to enforce acces restriction on EJBs

```
@Entity
@DeclareRoles({"manager","hr_manager"})
public class Employee implements Serializable {
@RolesAllowed({"manager","hr_manager"})
    public String getName() {
        return "name";
    }

@RolesAllowed("hr_manager")
    public void promote(Position toPosition) {
        //promote the emplyee
    }
@RolesAllowed({"manager","hr_manager"})
    public List<EvaluationRecords> getEvaluationRecords() {
        List<EvaluationRecords> evalRecord;
        //return a list containing all
        // EvaluationRecords of
        //this employee
        return evalRecord;
    }
@RolesAllowed("hr_manager")
    public List<EvaluationRecords> getEvaluationRecords(Date from, Date to) {
        List<EvaluationRecords> evalRecord;
        //return a list containin all
        //productivity evaluation of
        //this employee in the given time range
        return evalRecord;
    }
    @Id
    private Integer id;
    public Employee() {
    }
}
```

Using only two annotations, `@RolesAllowed` and `@DeclareRoles`, frees us from adding all deployment descriptor elements.

Similar to Web module which had a run-as element in the standard deployment descriptor, here in EJB module we have the same element. This element will allow outgoing calls from the EJB to use a specific role included in the `role-name` element.

```
<security-identity>
    <run-as>
      <description/>
       <role-name>internal_role</role-name>
    </run-as>
 </security-identity>
```

This snippet should be placed inside the EJB declaration element of the deployment descriptor

Hot Tip

Different vendors may have specific non-Java EE compliant annotations for different Java EE components. Like JBoss @SecurityDomain annotation. Using non-standard compliant annotations will make it harder to port an application between different application server.

Securing EJB Modules programmatically

We can use EJB context, `javax.ejb.EJBContext`, to check whether the current user has a specific role using `isCallerInRole` method or we can extract the principal name of the subject using `getCallerPrincipal` method. For example:

```
@Stateless
public class EmployeeServiceBean
  {
    @Resource
    SessionContext ctx;
public void raiseEmployeePaygrade(int amount, long empID){
Employee employee = null;
//find the employee
String raisedBy =ctx.getCallerPrincipal().getName();
employee.raisePayGrade(850000, raisedBy);
//persist the employee
}
}
```

In the above sample code we injected the context and then we used it to get the principal name. Then we used it to keep record of who changed the salary of employee.

We can use Around Invoke Interceptors to intercept an EJB business methods call. Intercepting the call lets us access the method name, its parameters, EJB context (and therefore `isCallerInRole` and `getCallerPrincipal` methods). We can perform tasks like security check, logging and auditing or ever changing the values of method parameters using interceptors.

```
public class SampleInterceptor {
 @Resource
    private EJBContext context;
 @AroundInvoke
 protected Object audit(InvocationContext ctx) throws Exception {

  Principal p = context.getCallerPrincipal();
   if (userIsValid(p)) {
            //do some logging...
    }else{
            //logging and raising exception..
                    }
        return ctx.proceed();
    }
}
```

To use this interceptor we need only to place an Annotation on the designated EJB, for example to intercept any method call on `EmployeeServiceBean` we can do the following:

```
@Interceptors(SampleInterceptor.class)
@Stateless
public class EmployeeServiceBean {
// Source code omitted.
}
```

The @Interceptors can target classes, methods or both. To exclude a method from a class level interceptor we can use @ExcludeClassInterceptors annotation for that method.

We can use interceptor element of ejb-jar.xml deployment descriptor to specify interceptors if preferred.

APPLICATION CLIENT SECURITY

Application Client Container, which can host first tier client for enterprise applications, conducts the authentication by itself and when communicating with the EJBs, sends the authenticated subject along with the call. In the standard deployment descriptor we can configure the callback handler which collect the user credentials for authentication and all other measures are configured in the vendor specific deployment descriptor.

For example, the following snippet specifies the callback handler to collect user identity information.

```
<callback-handler>
security.refcard.SwingCallBackHandler
</callback-handler>
```

Hot Tip: The callback-handler element specifies the name of a callback class provided by the application for JAAS authentication. This class must implement the javax.security.auth.callback.CallbackHandler interface.

```
<resource-ref>
        <res-ref-name>TaskQueueFactory</res-ref-name>
            <jndi-name>jms/TaskQueueFactory</jndi-name>
        <default-resource-principal>
            <name>user</name>
            <password>password</password>
        </default-resource-principal>
    </resource-ref>
```

Application Client Container will use the authentication realm specified in the application.xml file to authenticate the users when a request for a constrained EJB is placed.

Security enforcement in Geronimo ACC

Similar to GlassFish, Geronimo provides some configuration elements in the vendor specific file named `geronimo-application-client.xml`.

Notable configuration elements are `default-subject`, `realm-name` and `callback-handler`. For example to configure the callback handler we can use the following snippet.

```
<callback-handler>
security.refcard.SwingCallBackHandler
</callback-handler>
```

Security enforcement in JBoss ACC

Configuration for the JBoss ACC container is stored in the `jboss-client.xml` file. This configuration file provides no further security customization for the application client.

DEFINING SECURITY IN ENTERPRISE APPLICATION LEVEL

As we saw in the Figure 3, the enterprise application archive (EAR) file can contain multiple modules intended to be deployed into different containers like Web, EJB or ACC. This EAR module has the deployment descriptor of its ow which we can use to include the share deployment plan details in addition to EAR specific declarations.

We can use the application level deployment descriptors to define roles, include the required role mappings and to specify the default security realm of all included modules.

We can use the standard deployment descriptor to define security roles. The syntax is the same as what we used in the `web.xml` and the `ejb-jar.xml`.

Similar to other vendor specific deployment descriptors, we can use the application level descriptor to define the application-wide role mapping and also to define the default security realm for all included modules.

The following table shows how we can use different vendor specific deployment descriptors for role mapping and specifying the default security realm and shows what security measures are accessible through the vendor specific enterprise application deployment descriptor.

Application Server	Description

GlassFish: sun-application.xml	Role mapping is similar to other Sun specific deployment descriptors.
	Defining the default security realm is as follow:
	`<realm>`
	`jdbc_realm`
	`</realm>`
	
	The element is an immediate child of the sun-application

Geronimo: geronimo-application.xml	Roles mapping syntax is similar to other Geronimo specific deployment descriptor.
	
	Specifying the default security realm is as follow:
	`<dependency>`
	`<groupId>console.realm</groupId>`
	`<artifactId>file_realm</artifactId>`
	`<version>1.0</version>`
	`<type>car</type>`
	`</dependency>`
	
	This element is immediate child element of the dependencies element which is a subelement of the environment element

JBoss: jboss-app.xml	Role mapping and specifying the security realm is the same as with jboss-web.xml and jboss.xml using the security domain element.

SECURING JAVA EE WEB SERVICES

In the Java EE specification, Web services can be deployed as a part of a Web module or an Enterprise module.

Web Services Security in Web Modules

In the Web module we can protect the Web service endpoint the same way we protect any other resource. We can define a resource collection and enforce access management and authentication on it. The most common form of protecting a Web service is using the HTTP Basic or HTTP Digest authentication.

For example if we use the HTTP basic authentication and our Web service client uses the Dispatch client API to access the Web service we we can use a snippet like the following one to include the username and password with the right access role to invoke a Web service.

```
sourceDispatch.getRequestContext().put(Dispatch.USERNAME_PROPERTY,"user");
sourceDispatch.getRequestContext().put(Dispatch.PASSWORD_PROPERTY,"password");
```

The user and the password should be valid in the configured realm of Web application and should have access right to the endpoint URL.

Hot Tip: Another way of authenticating the client to the server in HTTP level is using the Authenticator class which provides more functionalities and flexibilities. For more information about the authenticator class check http://java.sun.com/javase/6/docs/technotes/guides/net/http-auth.html

Web Services Security in EJB Modules

We can expose a Stateless Session Bean as a Web Service and therefore we can use all security annotations like `@RolesAllowed`, `@PermitAll` and their corresponding deployment descriptor elements to define its security plan.

But the authentication enforcement of the Web Services is vendor specific and each vendor uses its own method to define the authentication, security realms and so on.

Web Services Authentication in GlassFish

For GlassFish we should specify the authentication method and the security realm in the `sun-ejb-jar.xml`. For example, to specify HTTP Basic authentication method and a realm named file_realm as the security realm for a Web service called Echo we will have something similar to the following snippet.

```
<ejb>
  <ejb-name>Echo</ejb-name>
  <webservice-endpoint>
    <port-component-name>Echo</port-component-name>
    <login-config>
      <auth-method>BASIC</auth-method>
      <realm>file_realm</realm>
    </login-config>
  </webservice-endpoint>
</ejb>
```

Web Services Authentication in Geronimo

In Geronimo we can use the annotations to define the security plan and then the EJB deployment descriptor to specify the authentication mechanism and the security realm. For if the following snippet is placed inside the `openejb-jar.xml` we can expect an HTTP Basic authentication to protect the Echo Web service.

```
<enterprise-beans>
   <session>
    <ejb-name>Echo</ejb-name>
    <web-service-security>
      <security-realm-name>file_realm</security-realm-name>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
      <auth-method>BASIC</auth-method>
    </web-service-security>
    </session>
</enterprise-beans>
```

We simply specified HTTP Basic authentication and the file_realm to be used for this Web service.

Web Services Authentication in JBoss

To specify the authentication realm for a Web service deployed as a Stateless Session Bean we can use both annotation and deployment descriptor elements in JBoss. For example using annotation to secure a Web service can be as follows:

```
@WebService()
@WebContext(contextRoot="/EchoService",
            urlPattern="/Echo",
            authMethod="BASIC",
            secureWSDLAccess = false)
@SecurityDomain(value = "jboss-sec-domain")
@Stateless
...
```

The `@WebContext` annotation simply specifies the Web service endpoint, the authentication method which can be `CLIENT-CERT`, `BASIC` or `DIGEST` and finally it specifies whether clients need to provide credentials to view the WSDL or not.

The `@SecurityDomain` specify which security domain should be used for this Web service authentication.

We can access EJB Web services in the same way we access the Servlet powered Web services using Dynamic Proxy or the Dispatch API.

Hot Tip Each one of the studied application servers provides plethora of configuration and tweaking for Web services security to comply with WS-Security profiles. You can check their Websites to see what are the available options.

For a basic comparison and a quick start for these application servers take a look at: http://weblogs.java.net/blog/kalali/archive/2009/11/17/state-open-source-java-ee-application-servers.

ABOUT THE AUTHOR

Masoud Kalali has a software engineering degree and has been working on software development projects since 1998. He has experience with a variety of technologies (.NET, J2EE, CORBA, and COM+) on diverse platforms (Solaris, Linux, and Windows). His experience is in software architecture, design, and server-side development. Masoud has published several articles at Java.net and Dzone. He has authored multiple refcards, published by Dzone, including Using XML in Java, Berkeley DB Java Edition and GlassFish v3. Masoud is author of GlassFish security book published by Packt and he is one of founder members of NetBeans Dream Team and a GlassFish community spotlighted developer.

Masoud blog on Java EE, Software Architecture and Security at http://weblogs.java.net/blog/kalali/ and you can follow him at http://twitter.com/MasoudKalali/

Masoud can be reached via Kalali@gmail.com in case you had some queries about the book or if you just felt like talking to him about software engineering.

RECOMMENDED BOOK

Security was, is, and will be one of the most important aspects of Enterprise Applications and one of the most challenging areas for architects, developers, and administrators. It is mandatory for Java EE application developers to secure their enterprise applications using Glassfish security features.

Learn to secure Java EE artifacts (like Servlets and EJB methods), configure and use GlassFish JAAS modules, and establish environment and network security using this practical guide filled with examples. One of the things you will love about this book is that it covers the advantages of protecting application servers and web service providers using OpenSSO.

BUY NOW
books.dzone.com/books/glassfish-security

DZone, Inc.
140 Preston Executive Dr.
Suite 100
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-71-4
ISBN-10: 1-934238-71-6
50795
9 781934 238714
$7.95