J2EE Security in Oracle ADF Web Applications

An Oracle White Paper
September 2004
# J2EE Security in Oracle ADF Web Applications

Introduction........................................................................................................ 4
Apache Struts in Oracle ADF web applications ........................................... 5
Introduction to web-application security in J2EE ....................................... 5
web.xml........................................... 6
Authentication elements in web.xml.......................................................... 6
Authorization elements in web.xml............................................................. 7
  <security-role> ......................................................................................... 8
  <security-role-ref> ............................................................................... 8
  <web-resource-collection> ................................................................... 9
  <security-constraint> ......................................................................... 9
Struts application deployment ................................................................. 10
Deployment profiles ................................................................................... 11
  Business Services ...................................................................................... 11
  View and Controller ................................................................................ 12
Deployment to standalone OC4J .............................................................. 13
  Starting OC4J ......................................................................................... 13
  Stopping OC4J ....................................................................................... 14
Installing the ADF runtime to OC4J......................................................... 14
Creating an application server connection for OC4J............................... 14
Deploying the view project ....................................................................... 15
Deploying to Oracle Application Server 10g........................................... 15
Adding J2EE Security to Struts web applications ..................................... 17
  Creating J2EE security roles in Oracle JDeveloper ................................ 18
  Configuring form-based authentication .................................................. 18
  Accessing login information .................................................................... 20
Protecting the web application root URL .................................................. 21
Storing JSP source files in the WEB-INF directory .................................. 23
Handling J2EE security roles in the view layer ........................................ 24
  Struts Request Tag Library setup for Oracle JDeveloper 10g................. 26
Adding J2EE security roles to Struts Actions ........................................... 27
Redirect users to where they are coming from ........................................... 28
Showing security related messages to the client ...................................... 30
Programmatically evaluating J2EE roles in Struts Actions ....................... 32
Row level security....................................................................................... 34
Enhance the isUserInRole tag library class to evaluate JSTL expressions.................................................................34
Enhance the isUserInRole tag library class to support multiple J2EE roles........................................................................35
Edit the taglibs-request.tld file to enable JTLS processing..............36
Usage example..............................................................................37
Testing and deploying secured Struts applications..........................39
Required OC4J configuration files ............................................39
Orion-application.xml....................................................................39
Jazn-data.xml..............................................................................39
Orion-web.xml............................................................................40
Adding orion-application.xml to the web application project............40
Adding jazn-data.xml to the web application project .......................41
Adding orion-web.xml to the web application project.....................42
Setting up the embedded OC4J server for testing .........................43
Deployment to stand-alone OC4J....................................................44
Deployment to Oracle Application Server 10g..............................45
Single Sign-On and centralized user account management...............46
Configure a deployed web application for SSO .............................46
Create user accounts and user groups in OID .............................47
Map J2EE roles to OID groups.....................................................48
Summary......................................................................................50
Appendix a: Modified class File IsUserInRoleTag.java ....................51
INTRODUCTION

Just as you wouldn’t leave your house with the front door wide open, you shouldn’t run J2EE applications on the web without a reasonable level of precaution. Contrary to what many people think, security is not a single product that, once installed, protects applications against any kind of attack or vulnerability. Ideally, you should implement security as part of your application design, and not as an afterthought to the development process.¹

J2EE container-managed security allows you to configure web applications to be accessible by authenticated and authorized users only. Container-managed security is easy to use and well integrated in the J2EE platform. It is the J2EE container’s responsibility to ensure that all application requests from a specific user are executed within the user’s security context.

To keep this paper short, I will focus on applying J2EE security to web applications built with the Oracle Application Development Framework (Oracle ADF) and Apache Struts only. This paper does not cover model security, nor does it examine the use of JAAS² in this context. Reading this paper, you’ll learn about:

- General J2EE security for web applications
- Deploying Struts applications to Oracle Application Server and OC4J
- Protecting Struts applications developed with Oracle ADF
- Testing “J2EE web applications” in Oracle JDeveloper 10g
- Deploying J2EE secured web applications to Oracle Application Server 10g and OC4J

A word on the code examples in this white paper: As a best practice advise, consider subclassing the Apache Struts classes and JSP tag libraries. Though it seems easier to change the actual sources directly, you are much better off subclassing them if you plan to upgrade to future versions of Apache Struts.

¹ Don’t rely on security only as applied to the application itself. Good defense is structured like an onion, with many outer layers protecting the core.
You can download an Oracle JDeveloper 10g workspace with an accompanying Oracle ADF sample application from OTN.

APACHE STRUTS IN ORACLE ADF WEB APPLICATIONS

Oracle JDeveloper 10g includes an integrated J2EE framework, the Oracle Application Development Framework (Oracle ADF). Using Oracle ADF, application developers follow a consistent approach to developing J2EE applications, independent from the user interface technology and business model they use—thus, the user’s experience in JDeveloper is one of “productivity with choice.”

In modern Model-View-Controller (MVC) architectures, the application user interface (the view) is decoupled from the business logic. Because Oracle ADF follows the MVC paradigm, web application views for the same data model can be built using JavaServer Pages (JSP), Oracle ADF UIX, and Java Server Faces (JSF).

![Figure 1: Oracle ADF Model-View-Controller architecture](image)

The Oracle ADF model layer interacts with events from the controller, represented by Apache Struts, and feeds data from the business service to the application view.

INTRODUCTION TO WEB-APPLICATION SECURITY IN J2EE

For web resources like JSP pages and Servlets, declarative security is applied through security elements set in the web.xml deployment descriptor, restricting access based on a protected URL pattern. Programmatic security uses APIs exposed on the request object to define complex security constraints, such as those that require evaluating the time of the request in addition to an application security role.

J2EE security provides authentication and authorization support:

---

3 http://www.oracle.com/technology/products/jdev/collateral/papers/10g/strutssecuritysample.zip
Authentication – the name of the process that identifies a caller as a trustworthy identity

Authorization – determines which trusted user is allowed to access a particular resource

web.xml

The web.xml deployment descriptor is an XML-based configuration file that is deployed with the application in the web archive (WAR) deployment file.

web-app

<table>
<thead>
<tr>
<th>security-constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>web-resource-collection</td>
</tr>
<tr>
<td>() web-resource-name hr</td>
</tr>
<tr>
<td>() url-pattern secure_hr</td>
</tr>
<tr>
<td>auth-constraint</td>
</tr>
<tr>
<td>() role-name HR_EMPLOYEES</td>
</tr>
<tr>
<td>user-data-constraint</td>
</tr>
<tr>
<td>() transport-guarantee NONE</td>
</tr>
</tbody>
</table>

login-config

| () auth-method BASIC |
| () realm-name oracle.ccm.hr |

security-role

| () description Role required by all employees that work for HR and that need to have access to the secure HR application |
| () role-name HR_EMPLOYEES |

Figure 2: J2EE security-related elements in the web.xml file

The web.xml file contains configuration settings for web resources such as Java Server Pages and servlets used in an application. Optionally, J2EE security can be configured for either individual application URLs or the complete web application. Figure 2 shows a list of web.xml elements that are used to define security in web applications.

These elements are explained briefly in the following sections.

Authentication elements in web.xml

The <login-config> element defines the type authentication that is enforced when the J2EE server receives a client request for an access-restricted web page.

The <auth-method> element in the <login-config> element sets the type of authorization to one of the following:

- NONE – no authentication, which is the default setting
- BASIC – authentication is performed against the J2EE container’s user repository, for example, against jazn-data.xml in Oracle OC4J. To authenticate, a browser logon dialog is launched for the user to provide username and password.
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>oracle.com.hr</realm-name>
</login-config>

BASIC authentication adds the user credentials to each page request.

- **FORM** – authentication is performed against the J2EE deployment’s user file, but the logon dialog used is a custom web application, like a JSP form. For this to work, the logon form must look like the following:

  ```xml
  <form method="POST" action="j_security_check">
    <input type="text" name="j_username">
    <input type="password" name="j_password">
  </form>
  ```

  FORM-based authentication stores the username/password pair in the session, making it safer than BASIC authentication.

- **CLIENT-CERT** – uses HTTP over Secure Socket Layer (SSL). SSL provides data encryption on the communication protocol and optionally requires the client to authenticate using a public key certificate.

- **DIGEST** – the user id is authenticated by a one-way hash key that is sent to the browser as part of the HTTP request. No passwords are sent over the network. J2EE servers are not required to support DIGEST authentication, though OC4J and Oracle Application Server both do.

The `<realm-name>` element, which is available for BASIC and DIGEST authentication, specifies a `realm`, which defines the set of users and roles the client request is authenticated against.

A realm is like a database of users and roles that is used by individual applications for authentication and authorization. For example, the user Scott could be a member of two different realms. Having been granted the privileged role “admin” in one of the realms allows him to access web applications as an administrator only if they authenticate and authorize against that realm.

---

Using basic and form-based authentication may make your applications vulnerable. The HTTP protocol sends usernames and passwords to the server using either a clear text format for form-based authentication, or the base64-encoded format for basic authentication. For better protection, consider using HTTPS with basic and form-based authentication.

---

**Authorization elements in web.xml**

Once a user has been authenticated, the J2EE server has access to the security roles granted to that user. The user security roles are compared with the roles that
are required by a web resource. If the user has been granted the required security privileges, the client request succeeds; otherwise, the J2EE server returns HTTP error 401, SC_UNAUTHORIZED, which indicates that the user does not have enough privileges to access the component.

<security-role>
The <security-role> element defines a logical application role that is used to protect web resources. The role name defined through the security-role element must be mapped to an existing role on the target system at deployment time:

<security-role>
  <description></description>
  <role-name>HR_EMPLOYEES</role-name>
</security-role>

<security-role-ref>
The <security-role-ref> element is nested within the <servlet> element that configures the servlet class.

While developing a web component, the application developer may need to verify that the authenticated user is a member of the security role required to process the requested task. Without knowing about any existing roles defined in the web.xml file or roles available on the target system, the developer defines a role name reference that gets hard-coded in his application source code.

It’s the responsibility of the application assembler to make sure that the role name reference used in the application gets mapped to a J2EE security role defined in the web.xml file. For this mapping, the application assembler uses the <security-role-ref> element. The following example maps the security role name “BOSS” used in a servlet or JSP page to “HR_MANAGER” specified in the web.xml file:

<servlet>
  <servlet-name>…</servlet-name>
  <servlet-class>…</servlet-class>
  …
  <security-role-ref>
    <role-name>BOSS</role-name>
    <role-link>HR_MANAGER</role-link>
  </security-role-ref>
</servlet>

4 You don’t need to specify the <security-role-ref> element if the security role name reference used in the component is the same as the role name specified in the web.xml file.
<web-resource-collection>
    The <web-resource-collection> element is used in the <security-constraint> element to identify web resources that should be protected:

    <web-resource-collection>
        <web-resource-name>secure_hr</web-resource-name>
        <url-pattern>*.do</url-pattern>
    </web-resource-collection>

A web resource collection defines protection through a URL pattern that may contain wildcards at the beginning or end. The above URL pattern protects all web sources that end with a .do extension. A web resource collection can define one or more URL patterns, all protected by the same set of security roles.

<security-constraint>
Web resources are protected by security constraints. A <security-constraint> element in the web.xml file defines the access privileges required for a collection of resources, a list of URL patterns and HTTP methods, defined by the <web-resource-collection> element in web.xml. All URL patterns in a web resource collection are protected by the same set of security roles and HTTP methods. To protect URL patterns with different authorization levels, use additional security constraint elements.

A web.xml file can contain zero or more <security-constraint> elements. When configuring multiple security constraint elements, it is important to configure constraints in order from least to most privileged.

If a web application needs to define public access to areas protected by a security constraint added to the web application root, then define a security constraint for these pages and directories first, before protecting the root.

The following example is a login screen that is deployed with the application. This login screen may contain style sheet references and image references that also need to be accessible to unauthenticated users.

<!-- define the image and css directory for public access so that content stored in this paths can be used in a login screen -->

<security-constraint>
    <web-resource-collection>
        <web-resource-name>images</web-resource-name>
        <url-pattern>/images</url-pattern>
    </web-resource-collection>
    <web-resource-collection>
        <web-resource-name>css</web-resource-name>
        <url-pattern>/css</url-pattern>
    </web-resource-collection>
</security-constraint>
<!-- protect the web application context root so that only users in the HR_EMPLOYEES role can access it -->

<security-constraint>
  <web-resource-collection>
    <web-resource-name>HR</web-resource-name>
    <url-pattern>/</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>HR_EMPLOYEES</role-name>
    <http-method>POST</http-method>
    <http-method>GET</http-method>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint>

Using the optional <http-method> element allows you to specify the HTTP method (POST or GET) that should be protected.

Omitting the <http-method> element will automatically apply the defined security constraints to all HTTP methods, which is recommended and also the default setting when working with Oracle JDeveloper.

The transport guarantee is set to NONE, INTEGRAL and CONFIDENTIAL. INTEGRAL means that the communication between the client and the server is setup so others cannot tamper it. Using CONFIDENTIAL also requires that the communication happen in a way that cannot be read by others. To make either CONFIDENTIAL or INTEGRAL work, the HTTP server needs to support the Secure Socket Layer (SSL) protocol. All subsequent client requests to web resources protected by the same web resource collection will run with the same security constraint defined for the authentication.

The configuration above does protect POST method calls issued by the client that are targeting a web resource that ends on “*.do”.

STRUTS APPLICATION DEPLOYMENT

When you build new web applications using the default web application template in Oracle JDeveloper, you create two projects: a “Model” project and a “ViewController” project. In this paper, we assume that the model — the persistence layer — is built with ADF Business Components and that the application is deployed as a J2EE web application to a standalone installation of OC4J for testing and to Oracle Application Server 10g for production.
Both deployments are done directly from JDeveloper by defining a connection to the J2EE servers. Alternatively, the Web application can be deployed to a web archive (WAR) file, which then is deployed to the target J2EE server. The following explains both options:

- For the deployment to the standalone OC4J server, an application server connection gets built in Oracle JDeveloper.
- For the deployment to Oracle Application Server 10g, a WAR file gets created and deployed using Oracle Application Server Control 10g, previously known as Enterprise Manager.

Deployment profiles
To deploy the web application, a deployment profile needs to be created for the Model project and the ViewController project.

Because the application is going to be deployed as a web application, the deployment profile of the ViewController project will be configured to have a dependency on the deployment profile created for the model. This way only one WAR file needs to be deployed for the whole application.

Business Services
In the New Gallery select Deployment Profiles in the Categories tree, and Business Components Archive from the list of available items. If the model wasn’t Business Components, but EJB or another business service, a different archive is needed to create the deployment profile.

In the following, accept all default settings, which creates a Model.bcdeploy profile entry under the Resources node of the Model project.
In the JDeveloper Application Navigator, select the ViewController project node, right-click, and choose New from the context menu. In the New Gallery, select Deployment Profiles in the Categories tree and then select WAR File from the list of available deployment profiles.

Define a name for the deployment profile and accept the default location.

In the WAR Deployment Profile Properties dialog, you define the context root name for the web application, which will become a part of the application request URL, and set the profile dependency to the Model project.

To define the application root context name, in the WAR Deployment Profile Properties dialog, select the General node in the tree to the left, and then select the Specify J2EE Web Context Root radio button on the right. Enter a name for the context root of the Web application. The example in this white paper uses secure_HR.

To set the profile dependency, in the same dialog select the Profile Dependencies node in the tree on the left, and then the deployment profile for the model you created earlier. Doing this you ensures that the deployment archive for the model gets created and included in the deployment of the ViewController project.

Figure 3: Creating a Business Component deployment profile

View and Controller
Close the dialog by clicking **OK**, and save the workspace. The application is now ready to be deployed.

**Deployment to standalone OC4J**

To deploy the Struts web application to a standalone OC4J instance, you’ll follow this overall process: Start OC4J.

1. Install the ADF runtime on OC4J if not already done.
2. Create an application server connection in JDeveloper.
3. Deploy the view project.

**Starting OC4J**

You can download standalone OC4J from the Oracle Technology Network\(^5\) and install it by unzipping the downloaded file to the file system. To install OC4J, open a command-line window and navigate to the `<OC4J Home>/j2ee/home` directory, where `<OC4J Home>` is the directory to which you unzipped the content of the zip file, and type: `java -jar oc4j.jar -install`

You will be prompted to change the password for the admin user. The default password is “welcome”.

After the initial installation, you can start OC4J by typing:

```
Java -jar oc4j.jar
```

---

Stopping OC4J

To shut down OC4J, open a second command-line window, navigate to the 
<JDeveloper Home>\j2ee\home directory and type

```
java -jar admin.jar ormi://server:23791 admin <password>
-shutdown [force]
```

where <password> is the one specified when installing OC4J. The server name is the name of the machine that hosts OC4J, while 23791 is the default RMI port used by OC4J. Optionally, you can use the “force” argument to immediately shut down OC4J.

Installing the ADF runtime to OC4J

Oracle JDeveloper 10g has an ADF Runtime Installer option in its Tools menu that is used to install the ADF runtime libraries to different application servers and J2EE containers. The runtime libraries need to be installed only once. Before installing the runtime libraries, make sure that OC4J is stopped to avoid conflicts with files that are in use.

![Runtime Installer menu option in JDeveloper 10g](image)

The second dialog expects you to point the installer to the OC4J root directory, which is the directory you unzipped the OC4J software into.

Creating an application server connection for OC4J

In Oracle JDeveloper 10g, open the Connection Navigator (Ctrl-Shift-O) and select the Application Server node. Use the right-mouse button to bring up the context menu and choose New Application Server Connection. Provide a name for the new connection, for example, “OC4J_STAND-ALONE”. On the second screen, provide the admin password that you defined during installation of OC4J, which is “welcome” if you kept the default.

On the third dialog screen, specify the URL for ORMI access and point the dialog to the <JDeveloper Home>\j2ee\home directory, where the admin.jar file for OC4J is located. Test the connection in the next dialog and click Finish.
Deploying the view project

The view project contains the JSP files used by the web application, as well as the Struts controller configuration.

To deploy the Struts web application to OC4J, select the deployment profile created earlier for the view-controller project, and choose **Deploy to** | <OC4J>

connection name> from the context menu.

Figure 6: Deploying the web application to standalone OC4J

After the deployment finishes successfully, restart OC4J before calling the web application.

To start the application, type the following URL into a browser:

http://<server>:<port>/<web context root>/<source>

Deploying to Oracle Application Server 10g

You can manage Oracle Application Server 10g using the Oracle Application Server Control 10g, a web-based management console that allows you to deploy web archive files to a managed OC4J instance.

Before deploying any ADF application to the Oracle Application Server 10g middle-tier, make sure that it has the ADF runtime libraries installed.

As with standalone OC4J, you can install the ADF runtime on OracleAS 10g using the ADF Runtime Installer menu option in OracleJDeveloper 10g. To avoid problems when copying the files, make sure that the Oracle Application Server middle tier is shut down before running the installer.
As shown in Figure 6 above, the deployment profile created for the ViewController project can be deployed to a WAR file, which is created in the deploy directory of the ViewController project on the file system. Oracle Application Server Control 10g is used to deploy the WAR archive from its location on the file system.

During deployment, you need to specify a name for the application, as well as the URL for the web application root context.

Figure 7: Deploying a Struts Web application as a WAR file from the Oracle Application Server Control 10g management console

After successfully deploying the web application to Oracle Application Server 10g, you can start the application with the following URL:

http://<server>:<port>/<application-root-context>/
<Struts Action>

For the sample application used in this white paper, the URL is

http://<server>:7779/secure_hr/welcome.do
ADDING J2EE SECURITY TO STRUTS WEB APPLICATIONS

To protect Struts applications, J2EE security roles are added to the web.xml to protect URL patterns and to the struts-config.xml configuration file to protect Struts actions. For example, protecting the URL pattern "*.do" with the HR_EMPLOYEES security role restricts access to any Struts action to those authenticated users within this role. Adding the security role "HR_MANAGER" to the "roles" attribute of a Struts action in the struts-config.xml file requires a user to be also in the HR_MANAGER role to be able to request this particular Struts action. Using URL pattern security in combination with Struts action security allows for fine-grained access control.

Application security should handle missing access privileges gracefully. It should not show itself to the user unless things go terribly wrong.

To achieve this goal, follow this overall process: Create J2EE security roles with Oracle JDeveloper.

1. Configure form-based authentication.6
3. Protect the web application root URL.
4. Store JSP source files in the WEB-INF directory.
5. Handle J2EE security roles in the view layer.
6. Add J2EE security roles to Struts Actions.
7. Redirect users to where they are coming from.
8. Programmatically evaluate J2EE roles in Struts Actions.
9. Log off of the application.

6 This section introduces two tag libraries, “Request” and “Session,” that are in the Struts-supported tag libraries rather than the Struts standard tag library. The binary files for the libraries can be downloaded from the nightly builds link on the Apache Struts Web page.7 The tag libraries are listed on the left, and you need to click on them to get to the link. Instructions on how to import tag libraries are contained later in this section.

6 Form-based authentication is widely used, though not safe, because it allows users to integrate the logon form within the application. Form-based authentication should always be used in conjunction with SSL.
7 http://jakarta.apache.org/taglibs/index.html
You can download an Oracle JDeveloper 10g workspace with an accompanying
Oracle ADF sample application from OTN.

Creating J2EE security roles in Oracle JDeveloper

Security roles are created in the web.xml file and later mapped to existing roles
on the target platform.

In JDeveloper, the web.xml file is automatically added to the web application
project when using the default web template. If the web.xml file doesn’t exist,
you can create one by selecting the New entry on the context menu for the web
application project node. Choose web.xml from the list of deployment
descriptors and click the OK button. In the Application Navigator (ctrl+shift+A),
the web.xml file is shown under the “Web Content”| WEB-INF node.

![Image](image.png)

Figure 8: Create J2EE security roles in web.xml file

To define security roles, select the web.xml file entry in JDeveloper and select
Properties from the right-mouse context menu.

Configuring form-based authentication

Form-based authentication is defined in the web.xml deployment descriptor. To
configure the authentication method to use with this web application, edit the
web.xml file by selecting it and choosing the Properties entry from the context
menu.

---

8 http://www.oracle.com/technology/products/jdev/collateral/papers/10g/
strutssecuritysample.zip
Select the **Login Configuration** node and define the authentication and error form to be used with form-based authentication.

**Figure 9: Configuring form-based authentication using welcome.jsp as the “Login Page” and “Error Page”**.

In the example shown in the image above, the user gets redirected to the login page if authentication fails. The following entry is generated in the `web.xml` file for this settings

```xml
<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>welcome.jsp</form-login-page>
    <form-error-page>welcome.jsp</form-error-page>
  </form-login-config>
</login-config>
```

For form-based login to work, the `welcome.jsp` file, specified as “form-login-page”, must contain specific HTML elements to perform container based authentication

> You can also decide to use HTTP basic authentication. If you do, your application will use the login dialog of the browser instead of a custom HTML form. HTTPS client authentication is the most convenient authentication because it doesn’t require the user to provide username and password.
If you need to implement additional authentication functionality, e.g., lock a user out for a specific time after his authentication failed for the third time, then you can use a Servlet filter in front of the Struts servlet.

The filter can access the information provided through form-based authentication using the Servlet request object. Assuming that the variable “req” represents the handler to the request object:

```java
String username = req.getParameter("j_username");
```

**Accessing login information**

The login information provided by the user can be accessed as a parameter from the request object, which is accessible from the Struts Action. For form-based authentication, the following information can be obtained:

- `j_username` – (String) request.getParameter("j_username");
- `j_password` – (String) request.getParameter("j_password");
- `authType` FORMS BASIC – (String) request.getAuthType(); returns information about the authentication type configured for the application
- `method` POST/GET – (String) request.getMethod(); returns information about the request. Get requests are those where the request URL is completely shown in the browser’s URL field. A request issued from a Browser after the user typing in a URL is always a GET request.
- `Secure Socket Layer` – request.isSecure(); returns information about whether or not SSL is used for authentication.
Roles – request.isUserInRole(<name>); 9, returns a boolean information on whether or not the authenticated user is member in the requested role.

All this information can be used e.g. by a login Action to write log information or to determine the users access condition. The provided username and password can be used to authenticate the user to an external authentication system such as JAAS, and if this authentication fails, send a response using code like

```java
response.sendError(HttpServletResponse.SC_UNAUTHORIZED);
```

**Protecting the web application root URL**

Authentication isn’t required if no authorization constraints are specified. J2EE web applications are access-controlled by securing URL patterns configured for the web application. You can access control web pages based on file extensions, virtual paths and individual file names.

A Struts application that is deployed as a web application runs under a named J2EE application context. All application URLs start with the context root name and then add the virtual directory mapped for the application source files.

Best practice dictates that page navigation in Struts applications should use Struts Actions and not directly access the JSP page that renders the view. To enforce user authentication for Struts Actions, edit the `web.xml` file to protect the application context root or the Struts Action name or extension with a J2EE security role.

In JDeveloper, select the `web.xml` node in your project and choose the Properties entry from the context menu. In the “Web Application Deployment Descriptor” dialog, select the `security constraint` node to define the Web Resource Collection.

---

9 The Struts request tag library element isUserInRole() does not allow to specify more than one role name. To allow multiple user roles to access content protected by the isUserInRole() element, make sure the role used is one that contains the other. Later in this paper you learn how to enhance this library to handle multiple J2EE roles.
Based on the information shown in Figure 12, the following security elements are added to the web.xml file, protecting the application context root with the HR_EMPLOYEES role.

```xml
<security-constraint>
  <web-resource-collection>
    <web-resource-name>hr</web-resource-name>
    <url-pattern>/</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>HR_EMPLOYEES</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Securing the web application context root will also protect all the sub-directories it contains.

If form-based authentication is used and the login form contains images, or uses URL references to a style sheet, that are deployed as part of the web application, then no images and no styles will show until the user has authenticated successfully.

To avoid this side effect, you can either protect the Struts application based on the `.do` file extension or define a public `security-constraint` elements (see the `"<security-`
To protect Struts Actions based on the file extension, add the following to the web.xml file:

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>hr</web-resource-name>
        <url-pattern>*.do</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>HR_EMPLOYEES</role-name>
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

Protecting the .do file extension, as shown above, protects all client access to Struts Actions. Other file types need to be protected separately by their URL pattern as needed.

**Storing JSP source files in the WEB-INF directory**

JSP page files are created in the public_html directory of the web application project, if not specified differently.

An Action is an adapter between the contents of an incoming HTTP request and the corresponding business logic that should be executed to process the request. JSP files are only used for data input and therefore, to protect the JSP sources from direct access, it is considered good practice to store all JSP files in the WEB-INF directory within the “public_html” directory.

J2EE, by design, does not allow any client to directly access sources stored in the WEB-INF directory. JSP files are saved in the WEB-INF directory directly from JDeveloper when creating a form page in the “Struts Flow Diagram”.

---

Note that though J2EE demands the J2EE server to protect the WEB-INF directory, some application servers don’t respect this restriction. Oracle Application Server and OC4J do respect this restriction.

If you need to access a source in the WEB-INF directory from a client, you could write a servlet that acts as a proxy to the protected file.
If you use form-based authentication for J2EE security and the logon and error page files are contained in the web application deployment, then these files can also be moved to the WEB-INF directory. If you do this, make sure that the `<form-login-page>` and `<form-error-page>` elements in the `web.xml` file are updated accordingly:

```
<form-login-config>
  <form-login-page>WEB-INF/login.jsp
  </form-login-page>
  <form-login-page>WEB-INF/login_error.jsp
  </form-login-page>
</form-login-config>
```

Java Server Page files used within the Struts application may contain image, JavaScript and style sheet file references that, if located in the WEB-INF directory, will not be accessible at runtime using relative or absolute URLs.

Locating the files outside of the WEB-INF directory, while keeping the JSP files in the WEB-INF directory, will prevent images and applied style sheets from being shown at design time in Oracle JDeveloper's visual JSP editor.

To show images and style sheets at design time and runtime, it is recommended that you maintain two image and css directories, one in the public_html directory for runtime and one in the WEB-INF directory for design time. The issue of having to maintain duplicate directories will be addressed in a next release of Oracle JDeveloper.

The duplicated directories in WEB-INF should be excluded from web application deployment.

### Handling J2EE security roles in the view layer

You can evaluate J2EE roles in the presentation layer to enable or disable UI components. Some applications are designed to use a specific color to indicate inaccessible components, others decide to completely hide components that the authenticated user is not authorized to see or access. Both techniques are acceptable, though the first raises users' curiosity, challenging them on how to switch the inactive color to something indicating accessibility.

Be careful with security added to the view layer because under some conditions it is weak security. If, for example, view layer security is used to hide navigation controls from the eyes of an unauthorized user, then he still can try tampering the request URL for the page he isn’t allowed to access. Thus make sure that all Struts actions check authorization before granting access to a user.

Use either the roles attribute on the Struts action or, for a more complex evaluation of authorization, the Action’s implementation class to determine access control.

The Struts `struts-logic.tld` library contains a `<present>` tag that evaluates a specified condition before executing the tag body. You can access the
<present> element of the logic tag library from the JDeveloper Component palette, which automatically imports the tag library into the JSP page.

You use the logic element to hide sensitive information from being rendered in a Web application, based on a specific J2EE security role.

![Diagram](image)

**Figure 13: Accessing the Struts logic taglib from the JDeveloper Component Palette**

For example, the salary column on an employee page should only be shown to users that are in the HR_MANAGER role.

```html
<%@ taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt"%>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@ taglib uri="/WEB-INF.struts-html.tld" prefix="html"%>
<%@ taglib uri="/WEB-INF.struts-logic.tld" prefix="logic"%>
<%@ page contentType="text/html;charset=windows-1252"%>

<html>
...
</html>

...  

<td bgcolor="Silver">

  <logic:present role="HR_MANAGER">
  
  <c:out value="${Row['Salary']}"/>

  </logic:present>

</td>
...

</html>

In many situations you just want to change the way the information represents itself to the user instead of removing it completely. For example you might want to implement the following rules:
If a user isn’t allowed to access a page, then all the links to the page should be removed, but the text of the links should still be shown.

A user who isn’t authorized to see the salary data of a column should see "******" instead.

To do this, Struts provides a tag library, the “Request Tag Library,” that allows a page to render elements depending on security related information. The tag library uses JSTL to access J2EE roles.

In the example above, the user must be in the HR_SALARY role defined in the web.xml deployment descriptor to see the salary value. Otherwise, the user will see "******".

The Struts Request Tag Library and the Session Tag Library are not contained in the JDeveloper installation by default. You can download this library from the Apache Struts web page before you can use them.

To make the Struts RequestTagLibrary work with your web application, you need to perform the following setup.

---

1. Copy the `taglibs-request.tld` and the `taglibs-request.jar` file from the downloaded zip file to the file system.

   ▶ The `taglibs-request.tld` file downloaded for writing this whitepaper required a small modification to work properly. Open the tld file in a text editor and remove the entry
   ```
   ```
   This change was also required for the Response and Session taglib tlds.

2. Create a new library entry for the web application project in Oracle JDeveloper. To do this, use the right mouse button and choose **Project Properties** from the context menu. In the Properties dialog, select the Libraries node and press the New button. Provide a name for the new library (e.g. “StrutsAdditionalLibs”) and add the `taglibs-request.jar` file to the classpath.

   ▶ You can add the jar files for the Response Tag Library and the Session Library that are explained later in this document to the same named library.

3. To import the library into Oracle JDeveloper so that it shows in the Component Palette, choose the **Manage Libraries** option in the Tools menu. Select the JSP tag library **tag** and press the New button. In the following dialog, provide the location of the `taglibs-request.tld` file and the `taglibs-request.jar` file. Finally define a prefix used with this tag library when entered to a JSP page. The example above uses “req” as a prefix, which is also the prefix used in the taglib documentation.

4. The library now shows in the Component Palette under the name specified by saving the new library settings.

   ▶ Repeat step 3 for the Session Tag Library and any other additional Struts libraries that you want to make available in the Component Palette.

---

**Adding J2EE security roles to Struts Actions**

Since Struts 1.1, developers have been able to protect Struts Actions with J2EE security roles. The J2EE roles are added as an attribute to the Action. Use a comma separated list to define more than one security role for an Action.

▶ Struts does not invoke J2EE basic authentication even when J2EE roles are added to its Actions. For authenticating the user before accessing protected Struts Actions, it is important to protect at least one URL pattern outside of Struts.

The J2EE roles added to a Struts action are evaluated by `org.apache.struts.action.ProcessRequestor`, which reads the roles from the...
action mapping defined in the Struts configuration file. Once a Struts module is loaded, it gets frozen, which means you cannot add additional security roles to one of its configured Actions. Thus the use of J2EE security roles in Struts is a static implementation that cannot be changed at runtime.\textsuperscript{13}

When processing the Action, the roles added to the Action’s roles property are compared with the roles granted to the authenticated user for a match. If none of the required roles has been granted to the user, a SC_BAD_REQUEST error is raised on the \texttt{HTTPServletResponse} object. The SC_BAD_REQUEST has an associated error number of 400, which is checked in the \texttt{web.xml} deployment file.

\begin{verbatim}
<error-page>\textsuperscript{14}
  <error-code>
    400
  </error-code>
  <location>/handleBadPageRequest.do
</location>
</error-page>
\end{verbatim}

The above declaration declares that all errors with the error code 400 should be directed to the \texttt{/handleBadPageRequest.do} Struts action. The location to handle the error can be any source accessible on the Web, including static files.

**Redirect users to where they are coming from**

If a user requests a Struts action that he isn’t privileged to perform because of applied J2EE roles, the Struts RequestProcessor issues an SC_BAD_REQUEST response that must be handled in the \texttt{web.xml} configuration file, using an <error-page> element.

So as not to interrupt the users work within an application, he should be redirected to the page he visited before accessing the page he isn’t authorized for. For this, the SC_BAD_REQUEST error should be handled by a Struts action and this action must know about the path to the last visited page.

\begin{footnotesize}
\begin{itemize}
  \item \textsuperscript{13} If you wanted to make Struts security dynamic by changing the roles used with a Struts Action, you could change the way roles are looked up by subclassing the RequestProcessor and implementing your own version of the \texttt{processRole()} method.
  \item \textsuperscript{14} The current release of JDeveloper 10g (9.0.5.2) has a missing DTD entry for the \texttt{<error-page>} element, which means that it needs to be taken out of the \texttt{web.xml} file for compiling the project and added back after.
\end{itemize}
\end{footnotesize}
A better way of handling redirection to a page the user visited before is to subclass `org.apache.struts.action.ProcessRequestor` and to override the `processRoles()` method.

Instead of the `processRoles()` method adding a `SC_BAD_REQUEST` error to the `HttpServletResponse` object, the `RequestProcessor` could call the `forward()` method on the `RequestDispatcher` object, which is obtained from the session, and pass it a reference to the last visited page.

```java
Object lastVisitedPage = request.getSession().getAttribute("LastVisitedPage");
//handle null and zero length and cast to String
...
String prevPage = (String) lastVisitedPage;
request.getRequestDispatcher(prevPage).forward(request, response);
```

To ensure compatibility with existing Struts applications, the process method should call

```java
response.sendError(HttpServletResponse.SC_BAD_REQUEST,
getInternal().getMessage("notAuthorized", mapping.getPath()));
```
when no value for a last visited page can be found in the session.

Using this approach does eliminate the need for creating a separate Struts Action for handling bad page requests as well as using the <error-page> element, which then is used for other application flow unrelated errors.

To make this approach even more generic and less error prone, you could add the “LastVisitedPage” attribute reference as a constant to org.apache.struts.Globals.java.

Showing security related messages to the client
As explained in the section above, a user who requested a Struts resource that he isn’t privileged to access will be redirected to the page he last visited. To handle this even more gracefully, a message should be added to the JSP page informing him why this happened. To handle security related messages in Struts, do the following:

- Set a message in the HandleBadPageRequestAction, or whatever the name is in your application, before redirecting the user’s request
- Edit the project’s ApplicationsResources.properties file to include the security message
- Add a <html:message> element to the JSP page

Because a missing security role in the user’s profile isn’t the same as an application error, the message should be shown using org.apache.struts.action.ActionMessage and not ActionError.

In the example used in the section above, the message needs to be set in the HandleBadPageRequestAction.

... ActionMessages am = new ActionMessages(); am.add(ActionMessages.GLOBAL_MESSAGE, new ActionMessage("security.message.missing.role")); saveMessages(request, am);

Struts supports internationalization using resource bundles to store application messages and their translations. You can look up the location of the message bundle as the value of the <message-resources> element in the Struts configuration file, struts-config.xml. The default resource property file - ApplicationResources.properties - is located in the same directory with the application source files.

---

\(^{15}\) Double check that the message reference is contained in the “ApplicationResources.properties” file. Otherwise a hard to trace “page not found error” is shown.
To display the message in a JSP page, the following tags are added to the page source.

```
<logic:messagesPresent message="true">
  <html:messages id="message" message="true">
    <bean:write name="message"/>
  </html:messages>
</logic:messagesPresent>
```

Figure 16: Printing Struts messages in a JSP page

A more common approach is to use the `<bean:message>` element instead of the `<html:messages>` element, which allows to access a specific message only instead all of them. This allows context related messages to be written to the page.

```
<logic:messagesPresent message="true">
  <bean:message key="security.message.missing.role">
    <bean:write name="message"/>
  </bean:message>
</logic:messagesPresent>
```

The code snippet shown above was created declaratively using the JDeveloper component palette, which will also create the required taglib references in the header of the JSP page.

HTML tag elements can be added between the Struts elements to beautify the message presentation, e.g., to show an icon in front of the message:

```
<logic:messagesPresent message="true">
  <img src="images/info.gif"/>
  <bean:message key="security.message.missing.role"/>
</logic:messagesPresent>
```
Programmatically evaluating J2EE roles in Struts Actions

A DataPage is the logical pairing of a page that renders information from the model, and a DataForwardAction that forwards to that page.

If a static security role doesn’t handle an application’s security requirements properly, authorization can be performed within the implementation class of the DataForwardAction. For example, a customer who wants to enter a Web page to initiate an online bank-transfer may be required to be within the “TRANSFER_OK” role and also be checked to have enough money in his account. While “TRANSFER_OK” can be a role that is granted based on a written agreement that the bank customer signed, the balance of the account can changes\textsuperscript{16}.

To handle this grant, you need to

- Get the authenticated user roles from the request and check for the “TRANSFER_OK” role
- Check the user’s account balance using
- Forward or reject the request

To create a Java implementation class for a DataForwardAction in a DataPage, select the DataPage icon in the Struts Page Flow Diagram in Oracle JDeveloper 10g and access the context menu. From the context menu select the Go to Code entry.

The user’s roles is checked in the findForward(DataActionContext actionContext) method of the data page.

To override the findForward() method in the DataPage, choose Tools | Override Methods ... from the top-level menu. In the “Override Methods” dialog select findForward and press ok. To check whether the authenticated user is in the “TRANSFER_OK” role, call

```java
boolean inRole = actionContext.getHttpServletRequest().isUserInRole("TRANSFER_OK")
```

The oracle.adf.controller.struts.actions.DataActionContext extends oracle.adf.controller.lifecycle.LifecycleContext, which handles all the state information of the underlying model.

\textsuperscript{16} Don’t rely on security alone to protect your business data. Though the given example can be handled completely with J2EE security, the account balance must also be checked within the application’s itself and the request rejected if there isn’t enough money in the account. Applying J2EE security to prevent the user from hacking the transfer form though not authorized is good practice, but it must be guaranteed that the web application works as designed even with no security applied.
If a dynamic value needs to be accessed to determine whether the authenticated user is allowed to proceed, this is done using the DataAction Context. Assuming that the account information used in our example is exposed through a client method on the Business Components application module, the following code accesses this information from Struts:

```java
protected void findForward(DataActionContext actionContext) throws Exception {
    HttpServletRequest req = actionContext.getHttpServletRequest();
    boolean userIsInRole = req.isUserInRole("TRANSFER_OK");
    String remoteUsr = req.getRemoteUser();
    double balance = 0;
    DCDataControl dc = actionContext.getBindingContext().
        findDataControl("AppModuleDataControl");

    if (dc instanceof DCJboDataControl) {
        balance = ((AppModule)
            dc.getDataProvider()).getBalanceForUser(remoteUsr);
    }
}
```

This sample code assumed that there was a method, `setBalanceForUser(String username)` that is exposed on the application module and that returns the account balance as a `double`. The argument for this method is the username retrieved for the J2EE authenticated application user. In the following both, the static J2EE role and the dynamic account data are evaluated and used to either forward the request to the online transfer form or reject it. For example:

```java
if (userIsInRole && balance > 2000) {
    // forward to transfer form
    super.findForward(actionContext);
}
```

```java
// reject request an route to other application page
else {
    String anotherPage = "/goPoorFellowPone.do";
    // make this a page forward and not a request redirect
    // by setting the third argument to false
    actionContext.setActionForward(new ActionForward("security", anotherPage, false));
}
```
Row level security

There are two kinds of row level security:

- **The user isn’t allowed to see particular data** – This requirement is best handled in the database, e.g. using Oracle Label security or Virtual Private Databases (VPD). The best practice for hiding data from a user is to exclude it from the query, either by a database security context or a predicate added to the query request.

- **The user is allowed to see data, but not to edit** – This requirement is best handled in the view layer. As an example, a user may only be allowed to edit data that he is the owner of or that he was granted privileges for. The view layer, or presentation layer could render data that the user is allowed to edit as text fields and all other as read only strings. This kind of row level security is explored in the following.

> When adding security constraints to the view layer, make sure that it cannot be worked around and that it is a generic solution that can be reused in other pages. Most important, don’t violate the MVC paradigm by directly accessing the model.

The Struts Request Tag Library was introduced earlier in this paper. You can use this library to render UI components based on whether the authenticated user is a member of the static J2EE role used to protect the information [see Page 26]. The Request Tag Library is a generic solution that renders data not only for users that are in a required J2EE role, but also for users that aren’t. To use this library for row level security, you must make the following changes to the request tag library sources:

- Enhance the isUserInRole tag library class to evaluate JSTL expressions
- Enhance the isUserInRole tag library class to support more than one J2EE role added to a data row
- Edit the taglibs-request.tld file to enable JTLS processing

**Enhance the isUserInRole tag library class to evaluate JSTL expressions**

The Struts Standard Taglib sources can be downloaded from the Apache Struts website and are free to use and modify as long as the license agreement isn’t violated. Part of the standard tag libraries is the Request Tag Library in the \jakarta-taglibs\ request directory of the unzipped source files.

---

17 Write code in one line
19 Java Standard Tag Library
To enable JSTL expression processing in the isUserInRoleTag class\(^2\), a call to the method ExpressionUtil.evalNotNull(...) needs to be added. The ExpressionUtil class is contained in the org.apache.taglibs.standard.tag.el.core package, which can be added as a project library referencing the <JDeveloper 9.0.5 Home>\jakarta-taglibs\jstl-1.0.0 lib\standard.jar archive.

To create a new request-tag library in Oracle JDeveloper:

1. Unzip the Apache Struts tag library to the file system.
2. Create a new empty project in Oracle JDeveloper.
3. Copy the content of the unzipped jakarta-taglibs\request\src directory to the src subdirectory of the project directory.
4. Open the “org” directory using File|Open in JDeveloper. This imports all java source files into the project.
5. Create a new deployment profile for JSP tag libraries by selecting the project and pressing the right mouse button. Choose New from the context menu and select Taglib Jar file – Tag Library for JSP from the list of Deployment Profiles. Name the profile “taglibs-request”, which results in taglibs-reques.jar for the generated deployment archive.
6. Copy the new taglib archive to the <JDeveloper 9.0.5 Home >\jakarta-struts\lib directory, replacing the existing taglibs-reques.jar file\(^2\).
7. Edit the taglibs-request.tld file in the same directory as mentioned below

Enhance the isUserInRole tag library class to support multiple J2EE roles

Without any modifications added, the Request Tag Library only supports one J2EE security role with the isUserInRole element. To change this, the following code needs to be added, using a string tokenizer to separate role names.

```java
public final int doStartTag() throws JspException {
    evaluateExpressions();
    boolean result = true;
    boolean isInRole = false;
    if (role_ != null){
        StringTokenizer stk = new StringTokenizer(role_,",");  
        while(stk.hasMoreElements())
```

\(^2\) The source code for the modified class is added in the appendix of this paper
\(^2\) The taglibs-reques.jar only exists if you followed this paper and configured the Request Tag Library before
{  
String s = stk.nextToken();  
result =  
((HttpServletRequest)pageContext.getRequest())  
.isUserInRole(s);  
if (!isInRole & result==true)  
{
    isInRole = true;
}

if(value == isInRole )  
   return EVAL_BODY_INCLUDE;
else
   return SKIP_BODY;
}
else  
// no J2EE role assigned  
{
    if( value == true )  
        return EVAL_BODY_INCLUDE;
    else
        return SKIP_BODY;

}

**Edit the taglibs-request.tld file to enable JTLS processing**

The taglibs-request.tld file needs to be modified to indicate that the isUserInRole element does perform JSTL expression processing. Open \jakarta-struts\lib\taglibs-request.tld\23 with a text editor and add the following change to the file:

```xml
<tag>
    <name>isUserInRole</name>
    <tagclass>org.apache.taglibs.request.IsUserInRoleTag</tagclass>
    <bodycontent> JSP </bodycontent>
    <attribute>
        <name>role</name>
        <required>yes</required>
        <rtexprvalue>yes</rtexprvalue>
</tag>
```

---

23 The taglibs-request.tld file only exists if you followed this paper and configured the Request Tag Library before
This example uses an additional database table column “roles”, which holds J2EE role names as configured in the web.xml file. Each table row can have one or more J2EE roles assigned. Each J2EE role grants privileged access to a group of users. A row that does not have any J2EE roles assigned is treated as publicly accessible.

The following JSP page uses JSTL expression language to access the J2EE role information stored in the database table. If the user is a member of the row’s indicated J2EE role, then the row’s DEPARTMENT_NAME field is rendered as a hyperlink; otherwise, the data is rendered as a read-only string, using a red font.
Figure 17: J2EE security roles defined in the database table

```xml
<c:forEach varStatus="status" var="Row" items="${bindings.Departments.rangeSet}">
  <req:isUserRole role="${Row['Roles']}">
    <a href="showDepartments.do?event=setCurrentRowWithKeyArg0=<c:out value="${Row.rowKeyStr}"/>">
      <c:out value="${Row['DepartmentName']}"/>
    </a>
  </req:isUserRole>
  <req:isUserRole role="${Row['Roles']}" value="false">
    <font color="Red">
      <c:out value="${Row['DepartmentName']}"/>
    </font>
  </req:isUserRole>
</c:forEach>
```
Make sure that the page you navigate to also checks the constraints stored in the database. Though users can’t navigate to the next page using hyperlinks on the previous page, some users may start tampering the request URL, trying to guess the parameters that are needed to access information they are not allowed to see.

TESTING AND DEPLOYING SECURED STRUTS APPLICATIONS

Oracle Containers for J2EE and Oracle Application Server 10g use the Java Authentication and Authorization Service (JAAS) to provide container-managed security. Two security providers can be configured in Oracle Application Server

- XML – a file based provider that uses jazn-data.xml as a repository for user, role and security policies
- LDAP – a provider that uses Oracle Internet Directory to authenticate and authorize users.

The XML based JAAS provider is the default implementation in OC4J and Oracle Application Server.

Oracle JDeveloper 10g can be used to deploy, or prepare to deploy, and test web applications that have J2EE security constraints added. This section explains the deployments as well as the configuration files and dialogs involved.

**Required OC4J configuration files**

The following OC4J configuration files need to be provided when deploying J2EE secured Struts application built with Oracle ADF to OC4J stand-alone or Oracle Application Server.

**Orion-application.xml**

The orion-application.xml file is one of the deployment descriptors used to configure applications run on OC4J. The orion-application.xml contains information about the JAAS provider, XML based or LDAP based using the Oracle Internet Directory, the name of the default realm and the location of jazn-data.xml used to authenticate and authorize applications if the XML based provider is used.

**Jazn-data.xml**

The jazn-data.xml file is a data store that contains user, role, realm and policy information and is used with the XML based JAAS provider in OC4J and the Oracle Application Server.

jazn-data.xml can be different for each deployed J2EE application but also shared between applications. jazn-data.xml allows software developers to test and run web applications with applied J2EE security.
Having configured BASIC or FORM-based authentication in the web.xml file, jazn-data.xml is used to authenticate users. Passwords in jazn-data.xml are protected through encryption.

Security roles in jazn-data.xml are granted to usernames and roles also defined in jazn-data.xml.

**Orion-web.xml**

The orion-web.xml file maps J2EE security roles defined in the web.xml configuration file to user groups defined in Oracle Application Server or OC4J.

Using Oracle Application Server, user groups are defined either in the jazn-data.xml file or the Oracle Internet Directory (OID).

Using a stand-alone instance of OC4J, users and user groups are configured in jazn-data.xml.

Settings in the orion-web.xml file override settings in web.xml when deployed to OC4J.

**Adding orion-application.xml to the web application project**

To use jazn-data.xml to authenticate and authorize users running a specific web application, the orion-application.xml file must be deployed with the application.

To do this, select the view project in Oracle JDeveloper and select New from the right mouse context menu to open the “New Gallery” dialog. Select the orion-application.xml deployment descriptor entry in the list of Deployment Descriptors.

Open orion-application.xml, which is located in the META-INF package shown in the JDeveloper Application Navigator, by double clicking it. Add the following element to the file.

```
<orion-application>
  <data-sources path="./data-sources.xml"/>
  <jazn provider="XML" location="jazn-data.xml"
    default-realm="jazn.com"/>
</orion-application>
```
This defines \texttt{jazn-data.xml} as the JAAS provider. The realm name defined in this file will be the default realm name set when deploying the Enterprise Archive (EAR) file to Oracle Application Server 10g.

\textbf{Adding \texttt{jazn-data.xml} to the web application project}

To deploy users and roles with the project, the \texttt{jazn-data.xml} file needs to be added to the view project using the “Deployment Descriptors” entry of the “New Gallery” dialog (ctrl+n). Add the file to the project similarly to the way you added the \texttt{orion-application.xml} file.

The \texttt{jazn-data.xml} file is located in the META-INF package shown in the JDeveloper Application Navigator and can be edited with an integrated graphical editor that can be accessed by clicking the right mouse button on the \texttt{jazn-data.xml} entry and choosing “Properties” from the context menu.

To create user entries, select the \texttt{Realms} entry and press the New button. The real name can be any string identifier and needs to match the realm name defined in \texttt{web.xml} file when using BASIC authentication. FORM based authentication always uses the default realm, or the first in cases where many are found.

Create security roles and add users as members to these roles. The security roles must not match the security roles specified in the \texttt{web.xml} file, but can.
This creates `jazn-data.xml` entries like the following

```
<jazn-data>
  <jazn-realm>
    <realm>
      <name>com.oracle.secure.hr</name>
      <users>
        <user>
          <name>scott</name>
          <credentials>tiger</credentials>
        </user>
        <user>
          <name>fnimphiu</name>
          <credentials>welcome</credentials>
        </user>
      </users>
      <roles>
        <role>
          <name>Employee</name>
          <display-name>Employee role</display-name>
          <members>
            <member>
              <type>user</type>
              <name>scott</name>
            </member>
            <member>
              <type>user</type>
              <name>fnimphiu</name>
            </member>
          </members>
        </role>
      </roles>
    </realm>
  </jazn-realm>
</jazn-data>
```

This configuration defines two users, "scott" and "fnimphiu", that are both members of the "Employee" role.

**Adding `orion-web.xml` to the web application project**

To map the J2EE security roles to user groups in Oracle JAAS, the `orion-web.xml` file needs to be added to the view project using the “Deployment Descriptors” entry of the “New Gallery” dialog (ctrl+n). The `orion-web.xml` entry is located in the WEB-INF package of the project.
Like with the jzn-data.xml file, select Properties from the right mouse context menu of the orion-web.xml file to launch the graphical editor. Use the “Security Role Mappings” entry to create mappings for the J2EE security roles defined in the web application to the role names in the jzn-data.xml file.

```xml
<orion-web-app servlet-webdir="/servllet/">
  <security-role-mapping impliesAll="false"
    name="HR_EMPLOYEES">
    <group name="EMPLOYEE"/>
  </security-role-mapping>
</orion-web-app>
```

The configuration example above maps the role name “HR_EMPLOYEES” that exist in the web.xml deployment descriptor to the role “EMPLOYEE” defined in Oracle Application Server JAAS.

**Setting up the embedded OC4J server for testing**

The embedded OC4J server in Oracle JDeveloper can be configured with realm names, user accounts and user groups to simulate a production environment. To configure the integrated server, select Tools | Embedded OC4J Server Preferences from the top menu in JDeveloper.

---

24 You don’t need to create these mappings if the role names in both files are the same.
25 Remember that compiling the project may produce an error because of the <error-page> element in web.xml. To work around this, remove the <error-page> element for compiling the project and then add it back.
Deployment to stand-alone OC4J

Deployment to stand-alone OC4J is the same with security applied as it is without. The jazn-data.xml file with all the user account and role definitions gets deployed to the OC4J “application-deployments/ <web application context root>” directory.

To deploy the application, select the deployment descriptor in the view project and choose the OC4J instance from the right mouse context menu.

After the deployment finished successful, make sure you restarted OC4J before calling the web application. On a command line window, type the following command from the <OC4J home>\j2ee\home prompt:

```
java -jar admin.jar ormi://server:23791 admin <password> -shutdown [force]
```

If you haven’t changed the admin password, the default is “welcome”. Optionally, use the “force” argument can be used when shutting down OC4J. To start up OC4J again, type:

```
java -jar oc4j.jar
```
Web applications that use J2EE security can be deployed to Oracle Application Server as an EAR file, using either of the following techniques:

- Using the Oracle 10g Application Server Control administration console
- Using an application server connection in Oracle JDeveloper 10g

The deployment of web applications that require J2EE security is similar to the deployment of non-secured web applications. The difference is that the jazn-data.xml file needs to be deployed with the application or configured later using the Oracle 10g Application Server Control administration console.

![Image of Oracle Application Server Control interface]

Figure 20: Accessing jazn-data.xml file in Oracle 10g Application Server Control

To access the jazn-data.xml file from the Oracle 10g Application Server Control, select the web application from the “Applications” tab page and click onto the Security link in the following page. The “Security” page allows you to create users and user groups in jazn-data.xml and to assign group memberships to users. All changes in jazn-data.xml become effective after re-starting the OC4J instance.

---

26 To deploy the jazn-data.xml file created in Oracle JDeveloper 10g, use EAR files. Also make sure the orion-application.xml file is contained in the web application project.
Single Sign-On and centralized user account management

Oracle Application Server 10g allows you to configure web applications to use single sign-on, leveraging the Oracle Single Sign-On server and Oracle Internet Directory (OID). Single Sign-On can be enabled during or after application deployment, using the Oracle Application Server Control web console. Single sign-on is not available for OC4J instances that run stand-alone.

Using Oracle Single Sign-On and OID for user authentication and authorization allows a centralized user management that can be leveraged by different J2EE applications in contrast to a jazn-data.xml based deployment that hardly can be shared across applications.

All J2EE web applications that are configured to use the LDAP provider of JAAS in Oracle Application Server 10g are automatically single sign-on enabled. You can use the Oracle Application Server Control to configure a deployed web application to authenticate and authorize users against Oracle Internet Directory instead of jazn-data.xml.

The following steps are required to configure a J2EE web application for single sign-on, using OID as a central user repository:

- Configure the deployed web application to run in single sign-on mode. You can configure single sign-on during or after application deployment using the Oracle 10g Application Server Control.
- Create user accounts and user groups in OID. User entries in OID are created using the Delegated Administration Service (DAS), as HTML based administration console.
- Map J2EE roles to user groups in OID. Use the Oracle 10g Application Server Control console to configure J2EE role names used in the web application to user groups defined in OID.

Configure a deployed web application for SSO

Using Oracle Application Server Control is the recommended way to deploy J2EE web applications to Oracle Application Server 10g.

During deployment the JAAS provider for authenticating and authorizing application users is configured to be either XML based or LDAP based. Configuring the LDAP based provider automatically configures the application for single sign-on.

To use single sign-on, the web application authentication type must be set to BASIC, which can be done in Oracle JDeveloper using the web.xml file.

```xml
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>jazn.com</realm-name>
</login-config>
```
The jazn.com realm is the default security domain in Oracle Application Server JAAS.

Setting the user manager to JAZN LDAP when deploying the web application’s EAR file defines Oracle Internet Directory (OID) as the repository for users and user roles. User roles in Oracle Application Server JAAS are mapped to user groups in OID. The user group names must not be the same as the J2EE role names in the web.xml file.

### Deploy Application: User Manager

- Use JAZN XML User Manager
  - Default Realm: jazn.com
  - XML Data File: jazn-data.xml

- Use JAZN LDAP User Manager
  - LDAP Location: ldap://fnimphiu-pc.de.oracle.com:389
  - Default Realm: jazn.com

- Use Custom User Manager
  - Name:

![Figure 2: Configure LDAP for user authentication and authorization](image)

The OID server and port information is obtained from the Oracle Application Server configuration and must not be provided.

The provider configuration is added to the orion-application.xml file of the web application deployment.

```xml
<jazn provider="LDAP" default-realm="jazn.com"><jazn-web-app auth-method="SSO"/></jazn>
```

### Create user accounts and user groups in OID

To administer user accounts and user groups in OID, you can use any of the following tools:


---

27 Though Oracle Application Server JAAS is used, the authentication and authorization performed is J2EE security.
• **Directory Manager** – a Java based client application that provides a rich client interface to administering OID. This tool requires OID knowledge.

• **Command line** – Idapadd and ldifmigrate utilities to mass upload user data kept in the standard LDAP ldif format.

This paper focuses on the Delegated Administration Service (DAS), a self-service application that is easy to use.

To create users, select the **Directory** tab page on the DAS home page and connect to LDAP using a privileged administrator account like orcladmin. In the following page, click the **create** button to create a new user account. Fill in the required user information and save the new user to OID by pressing the submit button. Repeat the above for all users that need to run the web application.

If a user account exists, you can look it up by searching for it on the user page.

Create user groups by selecting the “Group” tab on the Internet Directory Web page and clicking the **create** button.

In the following page, define the new group name and click **Submit** to save it to OID.

![Internet Directory](image)

**Create Group**

**Basic Information**

- **Name**: EMPLOYEES
- **Display Name**: EMPLOYEES
- **Description**: All Employees that need basic access

**Group Visibility**: Public

* indicates a Required Field.

**Figure 22: Create user groups in DAS**

Further down the same page, you can add users and existing roles as members of this new role. Create as many groups as required to secure your business.

**Map J2EE roles to OID groups**

You use the Oracle Application Server Control to map J2EE roles to groups in OID. To perform this mapping, after deploying the application, follow the steps listed below in Oracle Application Server Control.
1. Select the name entry of the deployed web application in the Oracle Application Server Control and press the “Edit” button to access the application’s configuration page.

2. On the application configuration page, click the Security link to access the security page.

3. Use the Security Roles form to map J2EE roles to user groups in OID. Changes to the J2EE role mapping are written to the `orion-application.xml` configuration file of the web application deployment.

```xml
<security-role-mapping name="HR_EMPLOYEES">
  <group name="de/EMPLOYEES"/>
  <user name="de/fnimphiu"/>
</security-role-mapping>
```
If the OID group names are known before deployment, then the application assembler can create the `orion-application.xml` file within Oracle JDeveloper 10g.

After deploying the application, restart the OC4J instance so that the changes will take effect.

**SUMMARY**

J2EE web applications built with Oracle ADF and Jakarta Struts can be configured for fine-grained access control using container-managed security.

This paper presented an overview of how J2EE security can be used with J2EE web applications and Jakarta Struts to protect page content and page flow. Furthermore, it showed how smart error handling preserves usability.

Apache Struts provides additional tag libraries, such as the Request tag library and the Response tag library, that allow you to add security evaluations to the view layer without breaking the MVC paradigm.

You can make J2EE Web applications automatically Single Sign-On enabled by specifying the JAZN LDAP provider for authentication and authorization when deploying the web application EAR file to Oracle Application Server 10g. Single Sign-On is not available for web applications deployed to stand-alone OC4J instances.
APPENDIX A: MODIFIED CLASS FILE IsUserInRoleTag.java

All comments, including the comments mandated by the Jakarta Struts comments licence, have been removed from the modified code to keep this code sample short. Please see the original class for license agreements.

```java
package org.apache.taglibs.request;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
import org.apache.taglibs.standard.tag.common.core.*;
import org.apache.taglibs.standard.tag.el.core.*;
import java.util.*;

public class IsUserInRoleTag extends TagSupport {
    private boolean value = true;
    private String role_ = null;

    public final int doStartTag() throws JspException {
        evaluateExpressions();
        boolean result = true;
        boolean isInRole = false;
        if (role_ != null) {
            StringTokenizer stk = new StringTokenizer(role_,",");
            while (stk.hasMoreElements()) {
                String s = stk.nextToken();
                result = ((HttpServletRequest) pageContext.getRequest()).isUserInRole(s);
                if (!isInRole && result == true) {
                    isInRole = true;
                }
            }
        }
    }
}
```
if(value == isInRole )
    return EVAL_BODY_INCLUDE;
else
    return SKIP_BODY;
}
else
  // no J2EE role assigned
  {
    if( value == true )
      return EVAL_BODY_INCLUDE;
    else
      return SKIP_BODY;
  }
}
public final void setRole(String str)
{
  role_ = str;
}

private void evaluateExpressions() throws JspException
{
  try {
    role_ = (String) ExpressionUtil.evalNotNull(
      "isUserInRole", "role", role_, Object.class, this,
      pageContext);
  } catch (NullAttributeException ex) {
    role_ = null;
  }
}
}
J2EE Security in Oracle ADF web applications

August 2004
Author: Frank Nimphius
http://www.orablogs.com/fnimphius

Contributing Author: Duncan Mills
http://www.groundside.com/blog/content/DuncanMills

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com

Oracle Corporation provides the software
that powers the internet.

Oracle is a registered trademark of Oracle Corporation. Various
product and service names referenced herein may be trademarks
of Oracle Corporation. All other product and service names
mentioned may be trademarks of their respective owners.

Copyright © 2004 Oracle Corporation
All rights reserved.