Web Application Security

Implementing the Superstition in JDeveloper

Peter Koletzke

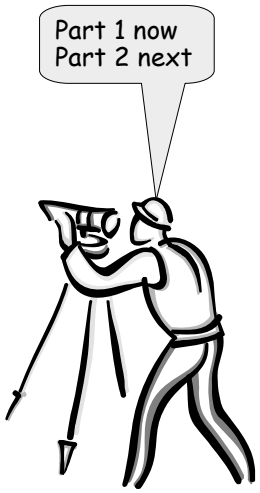
Technical Director &
Principal Instructor

Believe It or Not

> Security is mostly a superstition.
> It does not exist in nature,
> nor do the children of men
> as a whole experience it.
> Avoiding danger is no safer
> in the long run than outright exposure.
> Life is either a daring adventure
> or nothing.
>
> *—Helen Keller (1880–1968)*

Survey

- Jobs
  - Developer?
  - DBA?
  - Sys admin, others?
- Web Application Work
  - J2EE?
  - .NET?
  - PHP, ColdFusion, others?
- Tools
  - JDeveloper
  - Eclipse
  - Others



Agenda – Part 1

- Why security?
- OC4J security
- Set up the user repository
- Set up web descriptor security
- Define View layer security

Some material courtesy co-author Duncan Mills

Slides and white paper with hands-on practices are available on the Quovera and NYOUG websites

Application Areas of Exposure

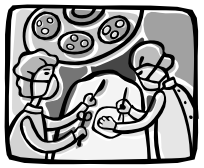
- Unapproved users can run the application
- Approved users can access data or functions they should not access
  - Access through View or Model code
- You cannot track who accesses the data
  - Approved or not
- Users bend normal query functions to gain unauthorized access
  - SQL injection

Security Objectives

- Ultimate security may just be superstition, however, data must be protected
- Why is exposure greater in web apps?
  - More accessible to any WWW hacker than an internal app
  - Given time and CPU power, a motivated hacker can break any security scheme
- Main objective with any security system:
  - Make breaking in as difficult as possible
- Assume file system of app server is secure
  - Reading configuration files with user identity and application security should be really difficult
  - Operating system and network has other security needs and features

Two Primary Operations

- Authentication
  - Validate that the user is who she/he claims to be
    - Normally done with passwords
    - With extra equipment, could be something else
      - Retinal scan, thumbprint, DNA (?)
- Authorization
  - Allow authenticated user access to specific resources
  - Usually done with security roles
    - Like database roles
    - Application components (pages, functions) and data are made available to named roles
    - Users are enrolled in roles
      - User has access to whatever the role is granted

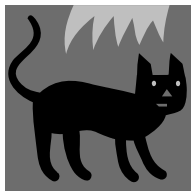


Agenda – Part 1

- Why security?
- OC4J security
- Set up the user repository
- Set up web descriptor security
- Define View layer security

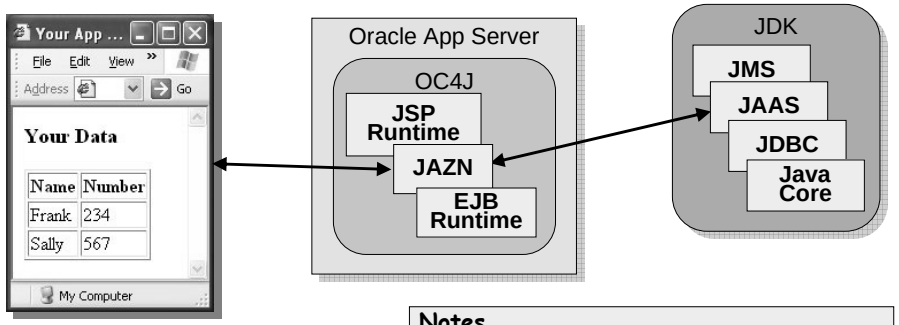
How to Implement the Superstition

- Use recognized, prebuilt, proven, supported security technologies
- Java Authentication and Authorization Services (JAAS)
  - Java API library in the J2SE Development Kit (JDK or J2SDK))
- One solution: JAZN
  - Available in Oracle App Server Containers for J2EE (OC4J)
    - Oracle Application Server's J2EE runtime
  - Java authorization and authentication
  - An API to JAAS
    - Meta-API?
  - You configure your application to use JAZN



Summarizing That

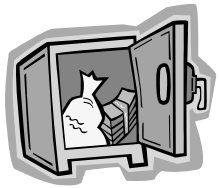
- OC4J in Oracle App Server contains JAZN that calls JAAS in the JDK



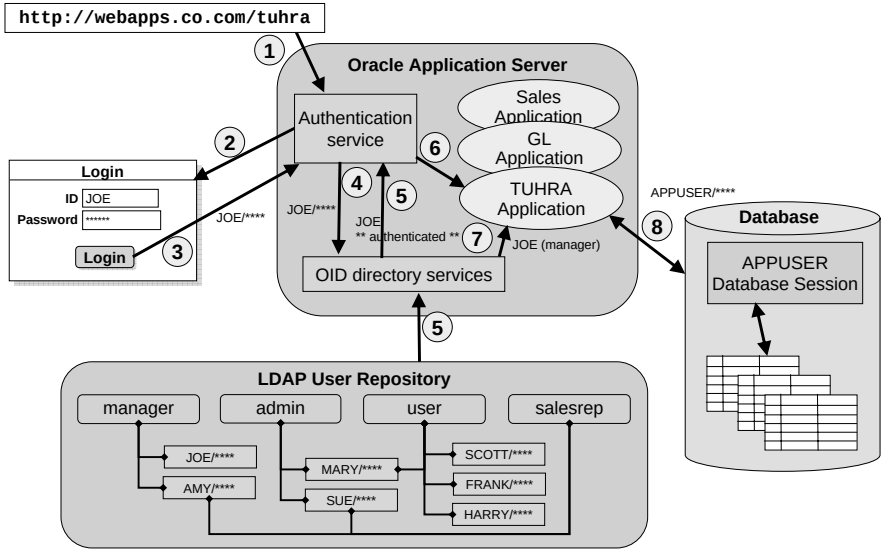
Notes
- This is only one method for security.
- This is not to scale.

The User Repository

- The storehouse of user and role information
  - A.k.a., *credentials store* or *identity store*
- JAZN can tap two types of user repositories
  - XML
    - Extensible Markup Language
    - Properties file containing user and role definitions
    - With 10.1.3 OC4J, can set up lightweight SSO
  - LDAP
    - Lightweight Directory Access Protocol
    - A communications protocol
    - Oracle Internet Directory (OID)
      - Used for Single Sign-On (SSO)
    - OID can read other LDAP providers
      - E.g., Microsoft Active Directory

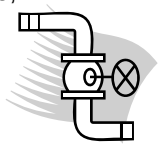


Application Security Flow



Application Security Flow

1. User sends HTTP request including a context root indicating a particular application.
2. The authentication service determines the method (XML or LDAP) and presents a login page.
3. The user enters an ID and password and submits the login page.
4. The authentication service requests OID to verify the user and password.
5. OID verifies the password in from the LDAP source and indicates pass or fail to the authentication service.
6. The authentication service accesses the application and places the user name into the HTTP session state.
7. The application can request the username or group (role, in this example, "manager") to which the user belongs
8. The application connects to the database using the application database user account (APPUSER) written into a configuration file.



Variations

- Single Signon (SSO)
  - The user is authenticated by iAS (OID or LDAP)
  - The user credentials (name and roles) are available in all applications managed by SSO
    - Details in *Oracle Containers for J2EE Security Guide 10g (10.1.3.1.0)* online guide – Ch.8
- Database users
  - You can connect the user repository to users and passwords in the Oracle database
  - Custom Login Module for JAZN or SSO
    - Details in the Nimphius/Mills article mentioned at end
- Other J2EE-compliant containers such as Tomcat work the same way
- HTTPS is preferred and the set up is the same

Agenda – Part 1

- Why security?
- OC4J security
- Set up the user repository
- Set up web descriptor security
- Define View layer security

Review: Security Tasks

Administrator

- ☑ Select a security system
  - JAZN here
- ❑ Set up user repository roles and users
- ❑ Enroll users in roles in the user repository
- ❑ Switch user repositories
  - Before production

Developer

- ❑ Set up logical application roles for the application
- ❑ Configure a login method for the application
- ❑ Set up security constraints to protect pages based on roles
- ❑ Protect items based on roles

To Do – Part 2

- ❑ Secure Model level attributes
- ❑ Create login and logout pages
- ❑ Protect against SQL injection attacks
- ❑ Log data modifications
- ❑ Display the logged-in user
- ❑ Use ADF Security

JDeveloper Support

- Define these files using JDeveloper's XML property editors
  - **<appname>-jazn-data.xml**
  - **<appname>-oc4j-app-data.xml**
  - **web.xml**
  - These files configure the Embedded OC4J Server in JDeveloper
- "<appname>" is the application workspace name in JDeveloper
  - Transfer these settings to the "system" level files in the 10.1.3 server
    - **system-jazn-data.xml**
    - **system-oc4j-app-data.xml**

Set Up Roles and User Accounts

- For XML provider in **<appname>-jazn-data.xml**
- Define within a realm (namespace within the XML file)
  - By default jazn.com

Role

```
<role>
  <name>admin</name>
</role>
```

User

```
<users>
  <user>
    <name>SKING</name>
    <credentials>{903}1JHgZuUDp..
    </credentials>
  </user>
</users>
```

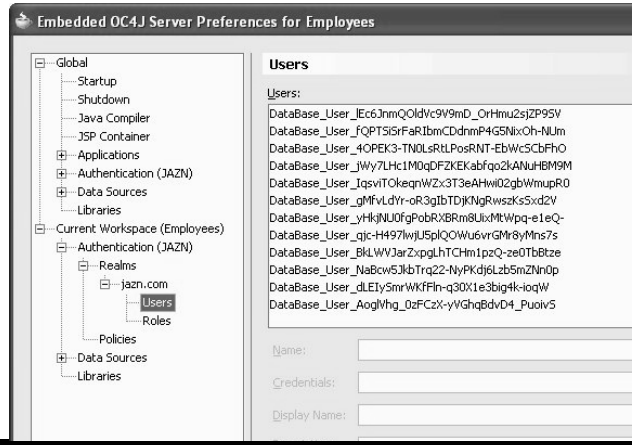
password obfuscation

Users in Role

```
<role>
  <name>admin</name>
  <members>
    <member>
      <type>user</type>
      <name>SKING</name>
    </member>
    <member>
      <type>user</type>
      <name>AHUNOLD</name>
    </member>
  </members>
</role>
```

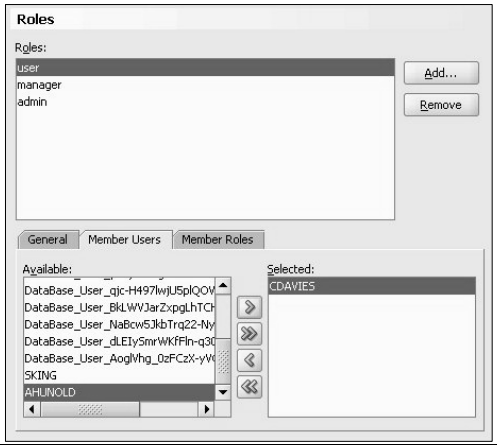

Users and Roles in JDeveloper

- **Tools | Embedded OC4J Preferences** after selecting the application
  - Current Workspace\Authentication\ realms\jazn.com
- Users node
  - Click Add
  - Define name and password
  - Password is obfuscated
- Roles
  - Click Add
  - Enter name, description



Enroll Users in Roles

- Members Users tab on Roles page
  - Shuttle users to Selected area.



Demo

Agenda – Part 1

- Why security?
- OC4J security
- Set up the user repository
- Set up web descriptor security
- Define View layer security

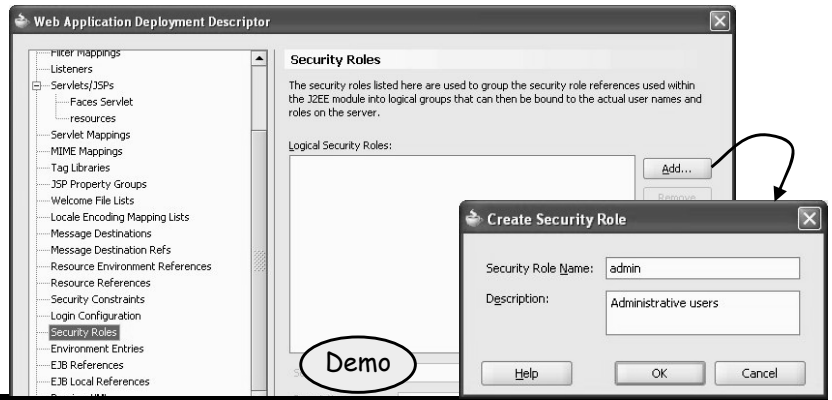
Set Up Logical Application Roles

- In **`web.xml`** (web application deployment descriptor)
- Standard J2EE XML file – standard contents
- Abstracts the roles required by the application from the user repository roles

```
<security-role>
  <description>Administrative users</description>
  <role-name>admin</role-name>
</security-role>
<security-role>
  <description>Management users</description>
  <role-name>manager</role-name>
</security-role>
```

Logical Application Roles

- On web.xml node in ViewController\Web Content\WEB-INF, select **Properties**
  - Web Application Deployment Descriptor dialog
  - On Security Roles page, click Add



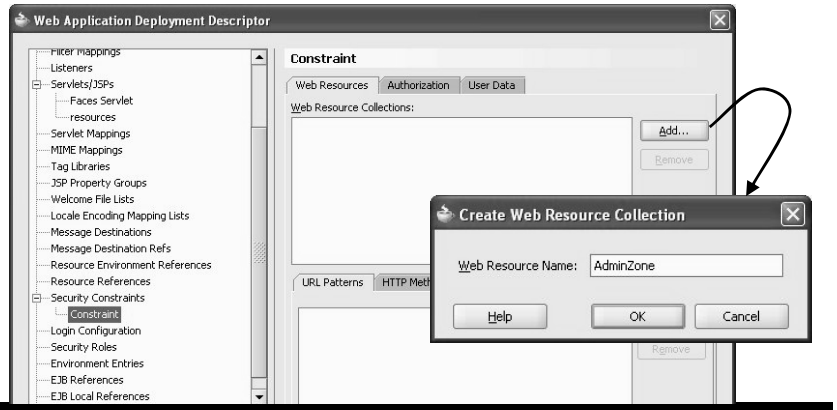
Define Security Constraints

- Used to map logical roles to URL patterns
- Restricts access to a set of files based on role
- URL pattern represents a directory and file names

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>UserZone</web-resource-name>
    <url-pattern>faces/pages/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>user</role-name>
    <role-name>admin</role-name>
    <role-name>manager</role-name>
  </auth-constraint>
</security-constraint>
```

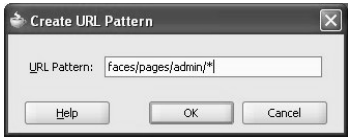
Security Constraints

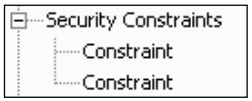
- On Security Constraints node (web.xml), click New
  - A Constraint child node will appear
- Click Add and name the constraint
  - Order matters - start with most restrictive



Define the Constraint

- Select Web Resource Collection (AdminZone)
  - On Authorization tab, select the roles
  - These roles will be constrained to the URL patterns you define next
- On Web Resources tab, select collection
  - Click Add and Enter path and file names (or "*" for all)
- Repeat creation of constraint for all other URL patterns needed
  - E.g., UserZone constraint for "faces/pages/*" URL pattern





Constraint Principles

- Security constraints can be defined for any number of roles.
  - Users can be members of any number of roles
  - Roles can contain any number of users.
- Security constraints protect files and directories.
  - Allow files and directories to be accessed by specific users (roles).
- Pages not protected by a security constraint are accessible to any user
- Security constraints are processed in the order in which they appear in the web.xml file.
  - Access allowed if the servlet finds the first security constraint for the user's role where the page matches the URL pattern
  - URL patterns can include the asterisk ("*") wildcard character
    - Match file names, for example, "*Emp.jsp"
    - Match all files in all subdirectories
      - For example, "/faces/pages/*"

Wildcard Gotcha

- Wildcard "*" stands for "all files and files in all subdirectories"
- E.g., you define URL patterns for "/faces/*" and "/faces/admin/*"
  - User role assigned "/faces/*"
  - Admin role assigned "/faces/admin/*"
  - User role then has access to /faces/admin pages – Not intentionally, however
- Solution: define specific patterns:
  - User role assigned "/faces/*.jsp"
  - Admin role assigned "/faces/admin/*"

Navigation Gotcha

- *Redirect* property is “false” by default
  - This indicates a “forward”
    - Controller calls the page directly
  - Problem: no URL is used so the URL pattern cannot be matched
- Set *Redirect* on navigation case to “true”
  - That way, the browser will request the page using the URL pattern
  - “Redirect” requests browser to send URL of the new page
  - Problem: ADF dialog does not work using redirect

Define Application Login

- Set login method
  - Basic or form-based authentication
  - Set in **`web.xml`**

Basic

```
<login-config>
  <auth-method>BASIC</auth-method>
</login-config>
```

Form-based

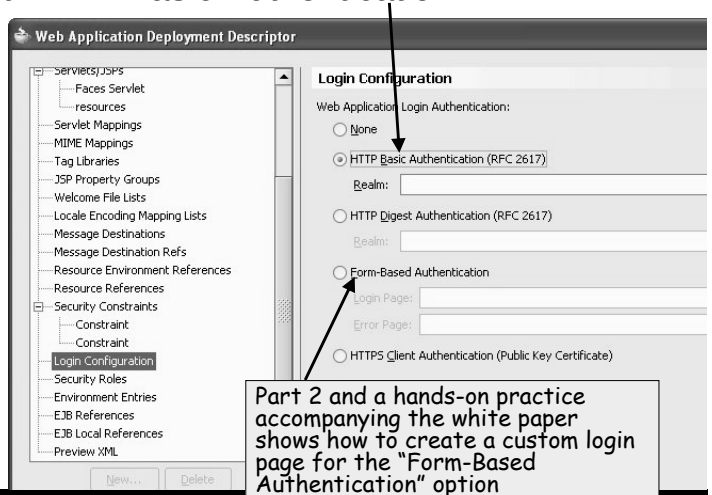
```
<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>security/login.jsp</form-login-page>
    <form-error-page>security/login.jsp</form-error-page>
  </form-login-config>
</login-config>
```

Specify a login and error page.

Demo

Define Login Method

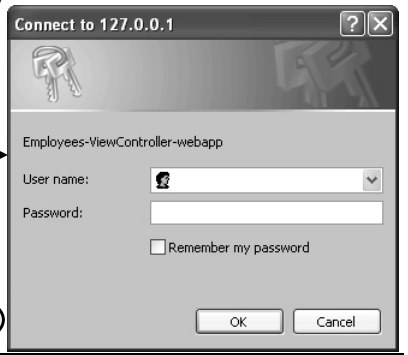
- Login Configuration page (web.xml)
  - Select HTTP Basic Authentication



Part 2 and a hands-on practice accompanying the white paper shows how to create a custom login page for the “Form-Based Authentication” option

Testing Basic Authentication

- Reminder:
  - admin can access faces/pages/admin/*
  - user and admin can access faces/pages/*
- Define pages for admin and user
  - One page in each directory
- Test each page
- Basic authentication dialog will appear when you run the page
- Test password protection



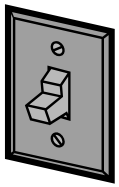
Demo

Switching User Repositories

- XML user repository is handy for development
  - Stored in **`<appname>-jazn-data.xml`** in the application root directory – edit it manually
  - Can manage this locally for application development
- LDAP is used for enterprise production systems
- Switch it in **`<appname>-oc4j-app-data.xml`**

From:
```
<jazn provider="XML"
      location="jazn-data.xml"
      default-realm="jazn.com"/>
```

To:
```
<jazn provider="LDAP"
      location=
      "ldap://ldap.tuhra.com:389"/>
```



Agenda – Part 1

- Why security?
- OC4J security
- Set up the user repository
- Set up web descriptor security
- Define View layer security

Who is Running the App?

- Get user role from FacesContext

```
public boolean isAdmin() {
  FacesContext ctx =
               FacesContext.getCurrentInstance();
  ExternalContext ectx = ctx.getExternalContext();
  return (ectx.isUserInRole("admin"));
}
```

- This requires writing code in some utility class
- Alternative: use JSF-Security
  - Adds an EL scope: *securityScope*

JSF-Security

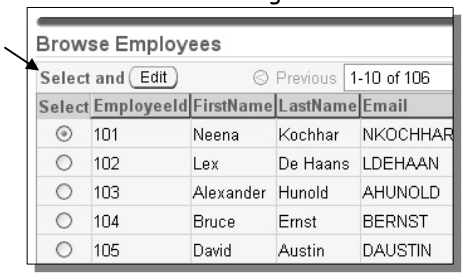
- Open source framework for exposing security settings to application
  - jsf-security.sourceforge.net
- Download library file and add it to the project
  - WEB-INF\lib
- Then role can be queried for value of properties on components
  - *Disabled*
  - *Rendered*
  - *Read-only*

Example 1

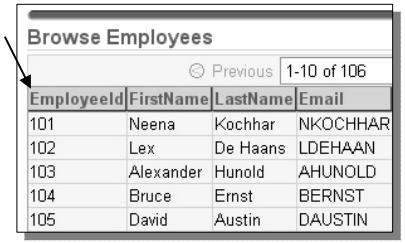
- Hide container (af:tableSelectOne) for all but admin and manager roles

```
<af:tableSelectOne text="Select and"
    rendered=
   "#{securityScope.userInRole['admin,manager']}">
```

admin and manager users



other users

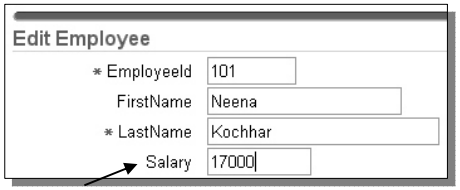


Example 2

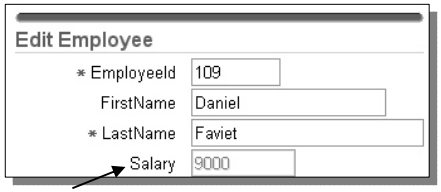
- Disable Salary item for all but admin roles

```
<af:inputText value="#{bindings.Salary.inputValue}"
   label="#{bindings.Salary.label}"
   required="#{bindings.Salary.mandatory}"
   columns="#{bindings.Salary.displayWidth}"
   disabled="#{ !securityScope.userInRole['admin']}"
/>
```

admin users



other users



Agenda - Part 2

- Define Model layer security
- Create login and logout pages
- Protect against SQL injection
- Log audit information
- Display the user name on the page
- Use the ADF Security framework

Review: Security Tasks

Administrator

- ☑ Select a security system
  - JAZN here
- ☑ Set up user repository roles and users
- ☑ Enroll users in roles in the user repository
- ☑ Switch user repositories
  - Before production

Developer

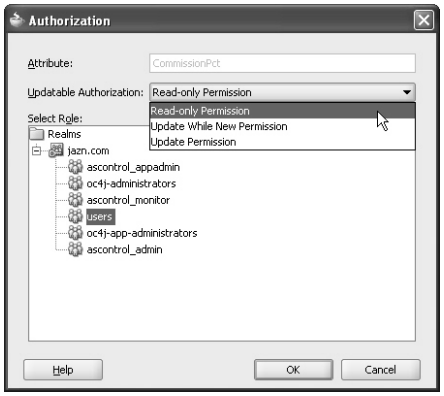
- ☑ Set up logical application roles for the application
- ☑ Configure a login method for the application
- ☑ Set up security constraints to protect pages based on roles
- ☑ Protect items based on roles

To Do:

- ❑ Secure Model level attributes
- ❑ Create login and logout pages
- ❑ Protect against SQL injection attacks
- ❑ Log data modifications
- ❑ Display the logged-in user
- ❑ Use ADF Security

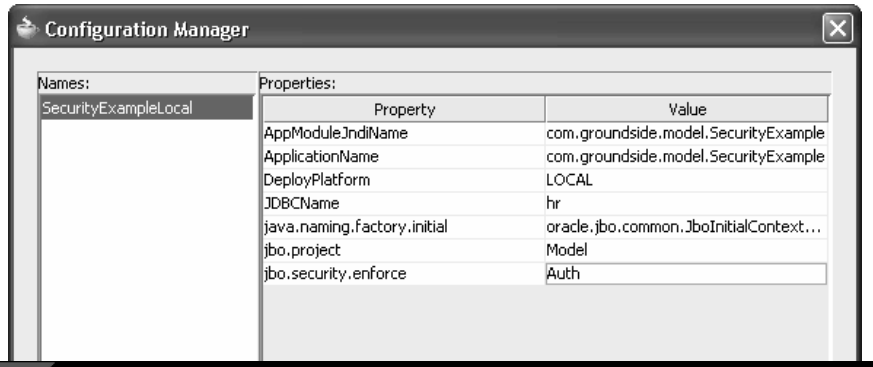
Securing Model Layer ADF BC Attributes

- ADF BC can read the role of an authenticated user
- Used to secure entity attributes
  - Mark them as
    - Read-only
    - Updateable while new
    - Always Updatable
- Automatically reflected by the UI



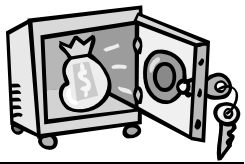
Secure Model Attributes

1. Tell ADF BC to worry about security
   - Set the configuration param **jbo.security.enforce=Auth**



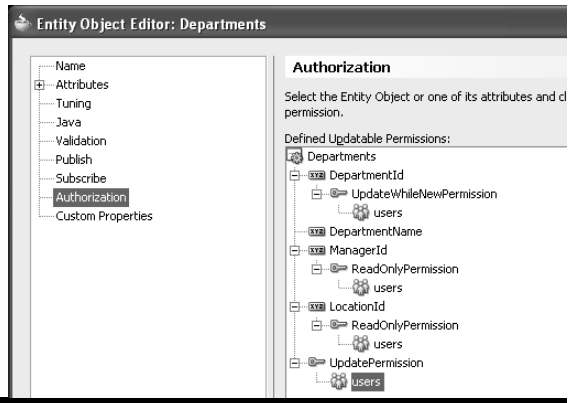
Secure Model Attributes

1. Tell ADF BC to worry about security
2. Propagate jazn-data.xml data
   - Make sure that the following files contain the same users and roles:
     - %JDEV%/j2ee/home/config/system-jazn-data.xml
     - %JDEV%/jdev/system/oracle.j2ee.10.1.3.n.n/embedded-oc4j/config/system-jazn-data.xml
     - %workspace%/workspace-jazn-data.xml
   - This is just for design time



Secure Model Attributes

1. Tell ADF BC to worry about security
2. Propagate the jazn-data.xml data
3. Edit the Entity Object
   - Select the Authorization node
   - Select an attribute
   - Click Add



Agenda - Part 2

- Define Model layer security
- Create login and logout pages
- Protect against SQL injection
- Log audit information
- Display the user name on the page
- Use the ADF Security framework

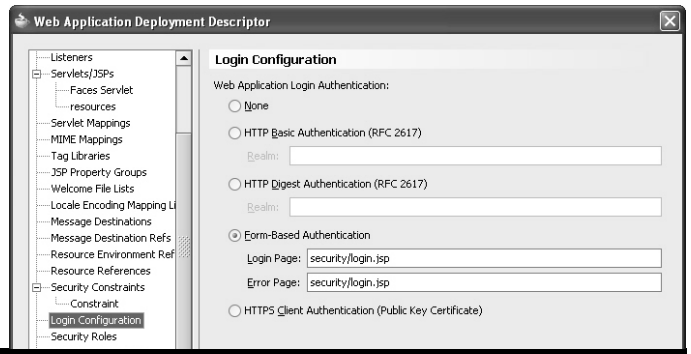
Login Page

1. Create non-JSF JSP
   - \security subdirectory
   - JSF is processed after the JSP authentication takes place
2. Add standard HTML items
   - Form
     - Name: j_security_check
     - Method: post
   - Fields
     - Names: j_username, j_password
   - Button
     - Name: login
     - Value: Login

Login Page

3. In the web.xml editor, set login page as security/login.jsp
4. This page will be used instead of the basic authentication page

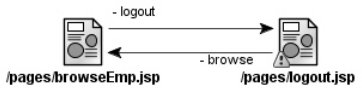
Hands-on practice shows how to display error message for invalid login.



Logout Page

- Need to invalidate session and navigate to the login page

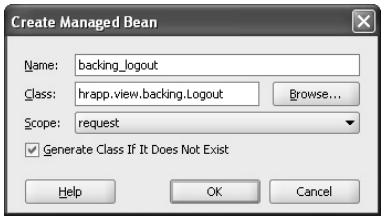
1. Define page and navigation



2. Add a button on the browse page
   - Text: Logout
   - Action: browse
     - Will navigate to the logout page
   - Immediate: true
     - So validation is not performed

Logout Page

3. Add items to the page
   - PanelHeader – for message
   - Button
     - Text: Yes
   - Button
     - Text: No
     - Action: browse
4. Add backing bean code to Yes button
   - Invalidate session
   - Navigate to the browse page
     - This will activate the login page
   - Double click the button to create the bean
     - Create the bean (backing_logout)
     - Rename the method (logoutButton_action)



Logout Page

- In new backing bean:

```
public String logoutButton_action() throws IOException
{
  ExternalContext ectx = FacesContext.
            getCurrentInstance().getExternalContext();
  HttpServletResponse response =
            (HttpServletResponse)ectx.getResponse();
  HttpSession session = (HttpSession)
            ectx.getSession(false);
  session.invalidate();
  response.sendRedirect("./browseEmp.jsp");
  return null;
}
```


Agenda - Part 2

- Define Model layer security
- Create login and logout pages
- Protect against SQL injection
- Log audit information
- Display the user name on the page
- Use the ADF Security framework

SQL Injection

- A technique used to insert unintended SQL text inside query forms, e.g.,

```
SELECT * FROM employees
WHERE last_name LIKE '<field_value>%'
```

- User is supposed to enter something like “Kin” in the Last Name query field
  - The SQL would then be:

```
SELECT * FROM employees
WHERE last_name LIKE 'Kin%'
```

SQL Injection Attempt

- The user could, instead, enter:

```
%' and salary > 10000 --
```

- This turns into:

```
SELECT * FROM employees
WHERE last_name LIKE '%' and salary > 10000 -- %'
```

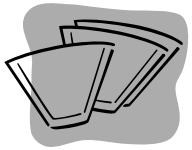
- The user will be able to see all employees with salaries over 10,000
  - Could be a problem
- Smart hackers could potentially enter function calls or other code that changes data as well

SQL Injection Solutions

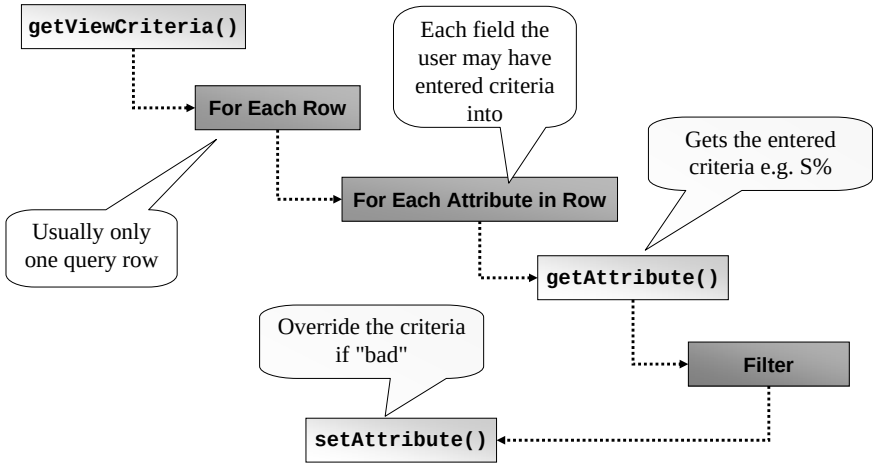
- Do not use Find mode (in ADF Faces)
  - However, it is very convenient
- Instead, use bind variables (parameterized queries)
  - Nearly 100% solution
  - Database matches datatypes
  - Does not construct SQL clause predicates
- Like the Oracle Reports bind parameter
  - QBE uses parameters like the lexical parameters: part of the SQL statement
- You can also filter query parameters before processing
  - See hands-on practice on website

Logic for Query Criteria Filter

- Intercept and filter the search criteria
  - Implement a Impl class for the VO
  - Override getViewCriteriaClause(boolean)
- Filter can check for "warning" strings
  - Operators
  - Column names
  - Pseudo columns
- Regular Expressions are excellent for this



Filtering the Criteria



A Simple Filter

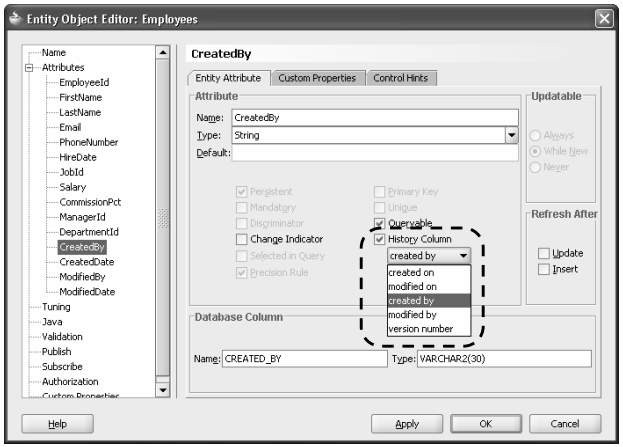
```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
…
protected String detectInjection(String criteria) {
  boolean reject = false;
  String testPattern =
       "^(>=|<=|=<|=>|<|>|<>|!=|=|BETWEEN|IN|LIKE|IS)";
  String testCriteria = criteria.trim().toUpperCase();
    if (testCriteria != null &&
        testCriteria.length() > 0) {
      Pattern pattern = Pattern.compile(testPattern);
      Matcher matcher = pattern.matcher(testCriteria);
      if (matcher.find())
        reject = true;
    }
    return reject?null:criteria;
  }
```


Agenda - Part 2

- Define Model layer security
- Create login and logout pages
- Protect against SQL injection
- Log audit information
- Display the user name on the page
- Use the ADF Security framework

Audit Columns in ADF BC

- Entity object history column property
  - created on
  - created by
  - modified on
  - modified by
  - version
- Requires use of JAZN security



More Advanced Audit

- History columns mechanism may not be enough for some uses
  - Saving old versions of records
  - Writing audit trails to a different table
  - Non ADF BC applications also update the tables
- Use table triggers to write the audit info
- The problem of "identity"
  - Database account is most likely shared
  - Need to push the J2EE identity into the DB…

Steps for Propagating Identity

1. Set up application context to store this additional metadata
2. Use the context information from the table triggers
3. Set the context information at runtime from ADF BC

Application Context

- A namespace containing name=value pairs
  - "Session State" in the database
  - USERENV is one such

```
SELECT SYS_CONTEXT('USERENV','NLS_DATE_FORMAT')
FROM DUAL;
```

- Populated by a defined (trusted) package

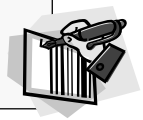
```
CREATE CONTEXT hr_context USING security_pkg;
```

- Used for VPD as well

The Context Package

```
CREATE OR REPLACE PACKAGE security_pkg
IS
   PROCEDURE set_security_context (
      p_username    IN VARCHAR2,
      p_application IN VARCHAR2 DEFAULT 'TUHRA');
END security_pkg;
```

```
CREATE OR REPLACE PACKAGE BODY security_pkg
IS
   PROCEDURE set_security_context (
      p_username    IN VARCHAR2,
      p_application IN VARCHAR2 DEFAULT 'TUHRA')
   IS
   BEGIN
      -- Write the user info into the context area
      SYS.DBMS_SESSION.set_context ('HR_CONTEXT',
         'APP_USERNAME', p_username);
   EXCEPTION
      -- exception handling code
   END;
END security_pkg;
```



Using The Context

```
CREATE OR REPLACE TRIGGER employees_audit_biu
   BEFORE INSERT OR UPDATE
   ON employees
   FOR EACH ROW
DECLARE
   v_user   VARCHAR2(30);
BEGIN
   v_user := UPPER(SYS_CONTEXT('HR_CONTEXT',
               'APP_USERNAME'));

   IF INSERTING
   THEN
      :NEW.created_by := v_user;
      :NEW.created_date := SYSDATE;
   ELSIF UPDATING
   THEN
      :NEW.modified_by := v_user;
      :NEW.modified_date := SYSDATE;
   END IF;
END;
```

Setting the Context from ADF

- Application module prepareSession() method

```
public void prepareSession(SessionData sessionData)
{
  super.prepareSession(sessionData);
  // Retrieve the J2EE user ID
  String authenticatedUser = getUserPrincipalName();
  DBTransactionImpl dbTrans =
              (DBTransactionImpl)getDBTransaction();
  CallableStatement cStmt =
              dbTrans.createCallableStatement(
      ("BEGIN " +
         "security_pkg.set_security_context(?); " +
       "END;"), 0);
  try {
    callableStmt.setString(1, authenticatedUser);
    callableStmt.execute();
  }
...
```

Full example in the hands-on practices.

Agenda - Part 2

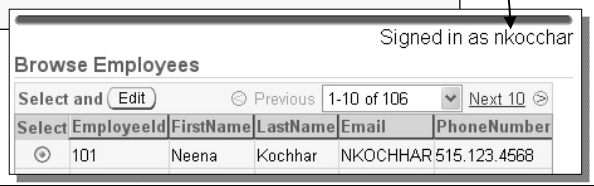
- Define Model layer security
- Create login and logout pages
- Protect against SQL injection
- Log audit information
- Display the user name on the page
- Use the ADF Security framework

Display the Username

- The user name is in the securityScope

1. Add an outputFormatted item
   - This should be part of the template
2. Set *Value* property
   - Signed in as "#{securityScope.remoteUser}"

```
<f:facet name="infoUser">
  <af:outputFormatted value=
        "Signed in as #{securityScope.remoteUser}"/>
</f:facet>
```



Agenda - Part 2

- Define Model layer security
- Create login and logout pages
- Protect against SQL injection
- Log audit information
- Display the user name on the page
- Use the ADF Security framework

ADF Security

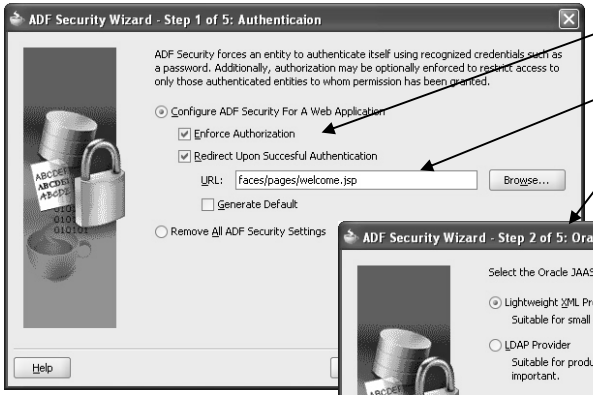
- Alternative to container security method explained before
- ADF feature that allows you to control access on the binding level
- Affects the View layer code
- Use this for any Business Service
  - ADF BC, EJB, web services, POJO
  - Compared to Model security which is for ADF BC only
- Works with JAAS and OC4J
- **Requirement**: currently, all bindings must have authorization defined

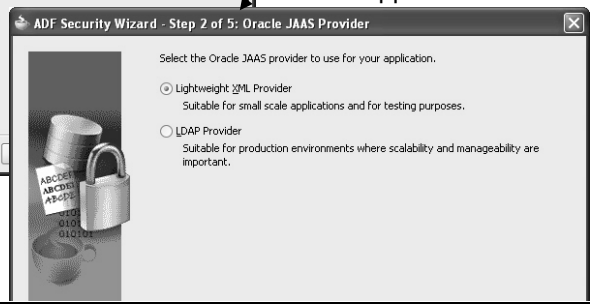
ADF Security - Steps

1. Configure application for ADF Security
   - Use the ADF Security Wizard
   - Available in JDev 10.1.3.2 (not 10.1.3.1)
2. Define OC4J container security
3. Define security on all bindings
   - Use the Edit Authorization right-click menu item
   - PageDef file
   - Individual bindings
4. Optionally restrict access to components

1. Configure the Application - A

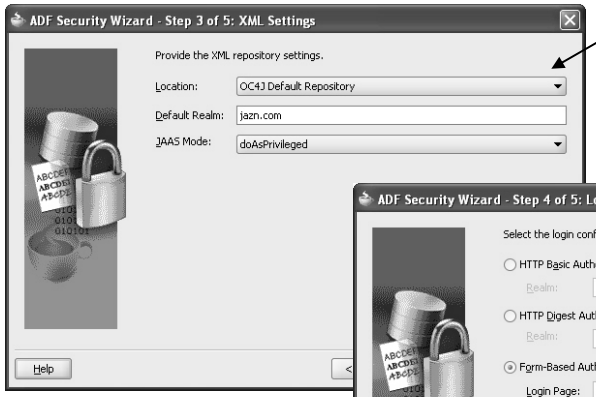
- **Tools | ADF Security**





- ADF Security requires definitions for all bindings
- Add “faces/” to Browse results
- Writes provider to orion-application.xml

1. Configure the Application - B

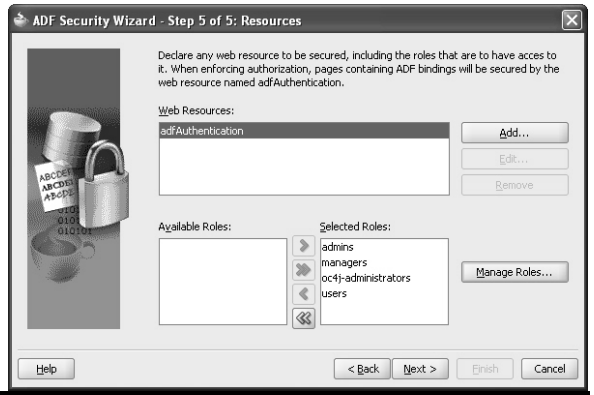


- Specifies system-jazn-data.xml
- Writes this to web.xml as with previous method

- Can have the wizard create these pages automatically – nice!

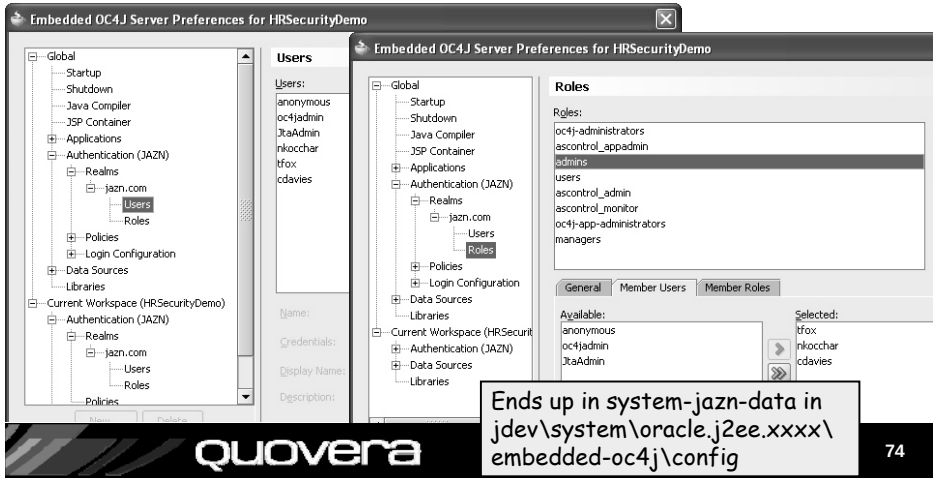
1. Configure the Application - C

- adfAuthentication is the constraint name
- Set up roles
- This ends up in web.xml
- Need to modify web.xml afterwards to add URL patterns to this constraint



2. Define OC4J Container Security

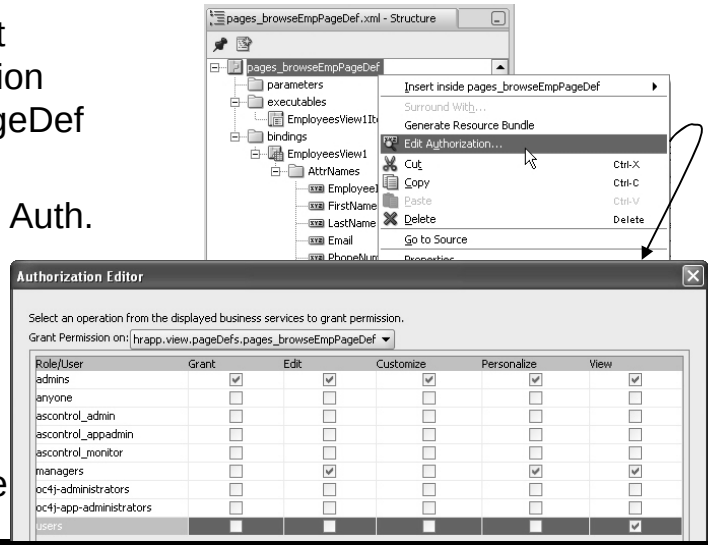
- Set up users and roles and map users to roles, as before but use Global area



Ends up in system-jazn-data in jdev\system\oracle.j2ee.xxxx\embedded-oc4j\config

3. Define Security on the Bindings

- Select Edit Authorization on the PageDef node
- Fill out the Auth. Editor
- Repeat for each binding and executable

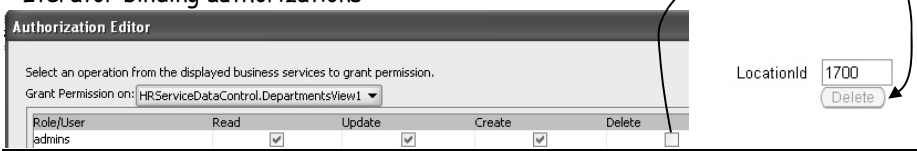


4. Optionally Restrict Access to Components - A

- Set Disabled, Rendered properties on components based on the PermissionInfo object
  - This reads the binding grants you defined in the Edit Authorization dialog for that user's role

```
<af:commandButton actionListener="#{bindings.Delete.execute}"
   text="Delete"
   disabled=
"#{!bindings.DepartmentsView1Iterator.permissionInfo.delete}"
/>
```

Iterator binding authorizations



4. Optionally Restrict Access to Components - B

- Access the grants programmatically in backing bean code

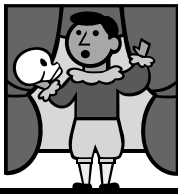
```
public String createDepartmentsAction()
{
  final FacesContext fc = FacesContext.getCurrentInstance();
  final Application fapp = fc.getApplication();
  //get binding
  OperationBinding obind = (OperationBinding)
     fapp.createValueBinding("#{bindings.CreateDepartment}").
       getValue(fc);

  if (ADFContext.getCurrent().getSecurityContext().hasPermission
      (new RowSetPermission("AppModuleDataControl.DepartmentsView1",
        "create")))
  {
    obind.execute();
    if (obind.getErrors() == null || obind.getErrors().size()==0)
    {
    return "editDepartments";
    }
  }
 return null;
}
```

Example from Frank Nimphius' article cited later.

To Use or Not To Use?

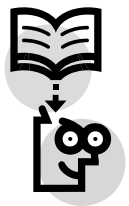
- It's actually more complex, but easier to define
  - Huh?
  - Easy: Wizard and authorization dialogs help
  - Complex: It requires granting every binding
- Use it if your business service is not ADF BC
- Use it if you need low-level control of elements
  - More options than ADF BC authorization control
- Use it if you prefer "easy" declarative screens
- Wait, if you can, for an upcoming version that allows you to declare permissions for only bindings that need permissions

Stay tuned for JDev R 11



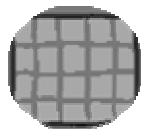
Other Resources

- *Declarative J2EE authentication and authorization with JAAS,* Frank Nimphius and Duncan Mills
  - Google search that title
- *Oracle Application Server Containers for J2EE Security Guide 10g Release 3 (10.1.3)*
  - download-east.oracle.com/docs/cd /B25221_04/web.1013/b14429/toc.htm
- *Introduction to ADF Security in JDeveloper 10.1.3.2*, Frank Nimphius on OTN
  - Google search that title
- White paper for this talk
  - Hands-on practices
  - On the IOUG SELECT and Quovera websites



Summary

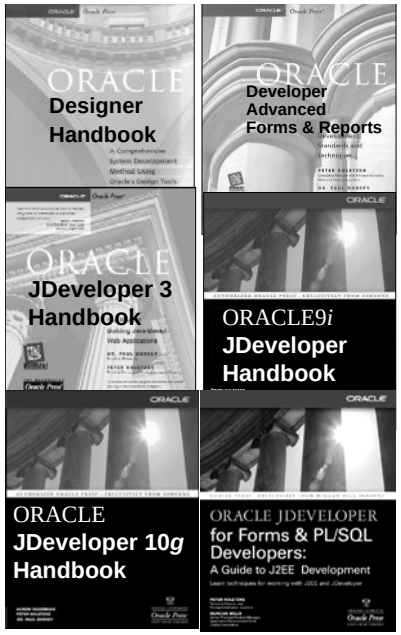
- You need to design application security
- OC4J offers easy access to standard JAAS security features (JAZN)
- JAZN supports user repositories in XML and LDAP
- JDeveloper can help you define XML user repositories and hooks into the app
- Design and test for all security breach scenarios



Out of Business

We will bankrupt ourselves
in the vain search
for absolute security.

—Dwight David Eisenhower, (1890-1969)



- Please fill out the evals
- Books co-authored with Dr. Paul Dorsey, Avrom Roy-Faderman, & Duncan Mills
- Personal web site:
  `http://ourworld.compuserve.com/homepages/Peter_Koletzke`

quovera

http://www.quovera.com

- Founded in 1995 as Millennia Vision Corp.
- Profitable for 7+ years without outside funding
- Consultants each have 10+ years industry experience
- Strong High-Tech industry background
- 200+ clients/300+ projects
- JDeveloper Partner
- More technical white papers and presentations on the web site