

Form-Based Web Authentication

December 10, 2004

Louis E. Mauget
lmauget@crosslogic.com
CrossLogic Corp

Contents

Form-Based Protected Web Resources.....	3
Configuring Protected Resources	3
Basic Authentication.....	4
Form-Based Authentication	6
Example Application	7
Demonstration.....	8
Summary	13
Appendix.....	14
Environment.....	14
Web Application	14
/WEB-INF/web.xml.....	14
/index.jsp.....	15
/login.jsp.....	16
/error.jsp.....	16
/logout.jsp.....	16
/protected/index.jsp.....	16
References.....	18

Form-Based Protected Web Resources

This paper is about protecting access to Web resources by using facilities present in all J2EE Web containers. A resource is any file or program accessible through a URL using and HTTP method. We present a simple demonstration example that consists of five JSPs and no explicit servlets. We created it using only Windows Notepad, the Java SDK jar command, and Apache Tomcat. The appendix contains the source XML and JSP files that form the application.

The example presents a start page that invites the user to access a protected page. The Web container intercepts the request for the protected page and authenticates the user. An authenticated user has access to the protected page until she logs out. The page is not accessible by a non-authenticated user nor by an authenticated user who is not a member of a stipulated role.

We discuss how to use form-based authentication callbacks to enable the Web container to orchestrate authentication and access while allowing the application developer to create the login and logout pages. This is called form-based authentication.

Configuring Protected Resources

The Web application is configured in `WEB-INF/web.xml` as usual. A security-constraint XML element declares a collection of protected resources, constraints to user roles, which user roles can access what resources, the kind of authentication used, and the transport security level used to request access.

The following security-role elements define the configured roles that our example application will use in security constraints.

```
<!-- Security roles referenced by this web application -->
<security-role>
  <role-name>manager</role-name>
</security-role>
<security-role>
  <role-name>role1</role-name>
</security-role>
<security-role>
  <role-name>tomcat</role-name>
</security-role>
```

Whenever a user tries to access a resource protected by a security-constraint, the Web container intercepts that request, and verifies that the current user is a member of a role allowed to access that resource. If the user has not been identified, the Web container obtains a user (principal) identity through a login process, checks if the user presented the correct password, and obtains the roles defined for that user within Tomcat's Web container. The Web container refers to a server-wide principal registry to look for the user ID, password, and role. The kind of registry is configurable by the server

Form-Based Authentication

administrator. Tomcat defaults to an XML-defined in-memory registry but can alternatively use a relational database, an LDAP directory service, or a Kerberos registry.

The following is the security-constraint we used in our example. Notice that any URL pattern of **/protected/*** is constrained. Thus <http://localhost:8080/demo/protected/index.html> would be constrained for an application running on an ordinary Tomcat server with Web application root **demo**. Notice the optional granularity down to the HTTP method.

```
<security-constraint>
  <display-name>Demo Security Constraint</display-name>
  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <!-- Define the context-relative URL(s) to be protected -->
    <url-pattern>/protected/*</url-pattern>
    <!-- If you list http methods, only those methods are protected -->
    <http-method>DELETE</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
    <http-method>PUT</http-method>
  </web-resource-collection>
  <auth-constraint>
    <!-- Anyone with one of the listed roles may access this area -->
    <role-name>role1</role-name>
    <role-name>tomcat</role-name>
    <role-name>user</role-name>
    <role-name>manager</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Whenever we send a user ID and password over the network, the password is sent in base 64 encoding. This is considered to be clear text, so we should encrypt the transport connection. The security constraint requests an SSL connection by specifying the CONFIDENTIAL transport guarantee element that appears near the end of the security constraint shown previously.

The Web container needs a way to present a login prompt to the user. There are several options available that include basic authentication, form-based authentication, digest authentication, or exchanging public key certificates.

Basic Authentication

With basic authentication, the browser presents its own popup window to begin a basic authentication sequence. This is triggered by the Web container returning an HTTP status of 401 – “This request requires HTTP authentication.” The browser swallows this status and displays the login popup instead. If the user cancels the popup without logging in, the raw 401 status page will render unless an error page was defined for that status.

Form-Based Authentication

The user ID and password are sent as base 64 clear text. The browser holds the logged-in state, even across server restarts. When all instances of that browser close, the user is logged out. The following `web.xml` stanza specifies basic authentication.

```
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name> Basic Authentication Area</realm-name>
</login-config>
```

Since the login window is created by the browser implementation, not the application, the window appearance differs across browser implementations. Figure 1 shows an Internet Explorer login window on Windows XP service pack 2, while Figure 2 shows the login window for the same application as presented by Mozilla Firefox release 1.0. Notice how the configured realm name is used to indicate the target secured area.



Figure 1



Figure 2

Form-Based Authentication

If the user cancels the BASIC login, a 401 HTTP status – This request requires HTTP authentication – is returned to the browser from the Web container. This is because the resource is protected by a security constraint that requires that the user be in a particular role, but the user is anonymous and not in any role. The ugly error screen can be replaced by an application-supplied 401 error screen, but there are other downsides to basic authentication:

1. There is no way for the user to log out except to close all instances of the browser. This is a security issue on shared machines or public machines. It is a problem if the user has other identifiers for other roles. For example, she may need to log out of a user role and then log into a manager role.
2. The application has no control over the number of invalid login attempts or at the kind of recovery carried out when an attempt fails
3. The login token is not held in the HTTP session so if the session times out or is otherwise invalidated, the user is still logged in.
4. The application has no control over the appearance of the login window. Its appearance varies across browsers. This may matter in training scenarios or in places where a cohesive theme is desired, or where other information needs to be presented on the login screen.

Form-Based Authentication

In the example will show form-based authentication, the goal of this paper. We specify FORM instead of BASIC and then give the relative path to a login JSP and error JSP that will be invoked if login fails.

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>Form-Based Authentication Area</realm-name>
  <form-login-config>
    <form-login-page>/login.jsp</form-login-page>
    <form-error-page>/error.jsp</form-error-page>
  </form-login-config>
</login-config>
```

The application supplies the form-login-page and the form-error-page page. The error page has no dictated form. The example's error page simply displays that authenticated failed and presents a link to the start page. The login page has to have an HTML form that has a POST action named **j_security_check**, an input element named **j_username**, and a password element called **j_password**. These are highlighted in the following example of a login form:

```
<form method="POST" action="j_security_check">
  <table border="0">
    <tr>
      <td> User ID: </td>
      <td><input type="text" name="j_username"></td>
    </tr>
    <tr>
      <td> Password: </td>
      <td><input type="password" name="j_password"></td>
    </tr>
```

Form-Based Authentication

```
<tr>
  <td colspan="2"><input type="submit" name="logon" value="Login"></td>
</tr>
</table>
</form>
```

The Web container sends the login page to the browser whenever it intercepts a request to a protected resource from an unknown user.

Form-based authentication gives us the following advantages compared to basic authentication:

1. The application controls the look-and-feel of the login screen because it stipulates a JSP of its choosing to be called for login. The JSP need only provide a form supporting three well-known element names.
2. The user can log out and then log in as a different user, perhaps in a different role.
3. The login page and the error page are supplied by the application, so the application has control over recovery attempts.
4. The HTTP session holds the security token, not the browser.
5. If the session times out the user is logged out.
6. The application can log the user out by simply invalidating the HTTP session.

Example Application

Figure 3 shows the classifications of the pages used by our example Web application. The top row of the figure shows two pages available to any user and one protected page available only to any user that is a member of a configured role or roles. The application can invoke those pages through links or forms as usual, except that the container intercepts requests to the protected page to first check if the user is in a configured role.

The two pages in the bottom row of the figure are invoked automatically, as needed, by the Web container whenever an anonymous user tries to access a protected page. The application supplies those two pages but does not invoke them directly. Thus the login and error pages function as callbacks.

After the user is authenticated to the configured role, requests for the protected page flow unimpeded until the user logs out. The state of “logged in” is remembered in the HTTP session, so the user is logged out whenever the session times out or is invalidated.

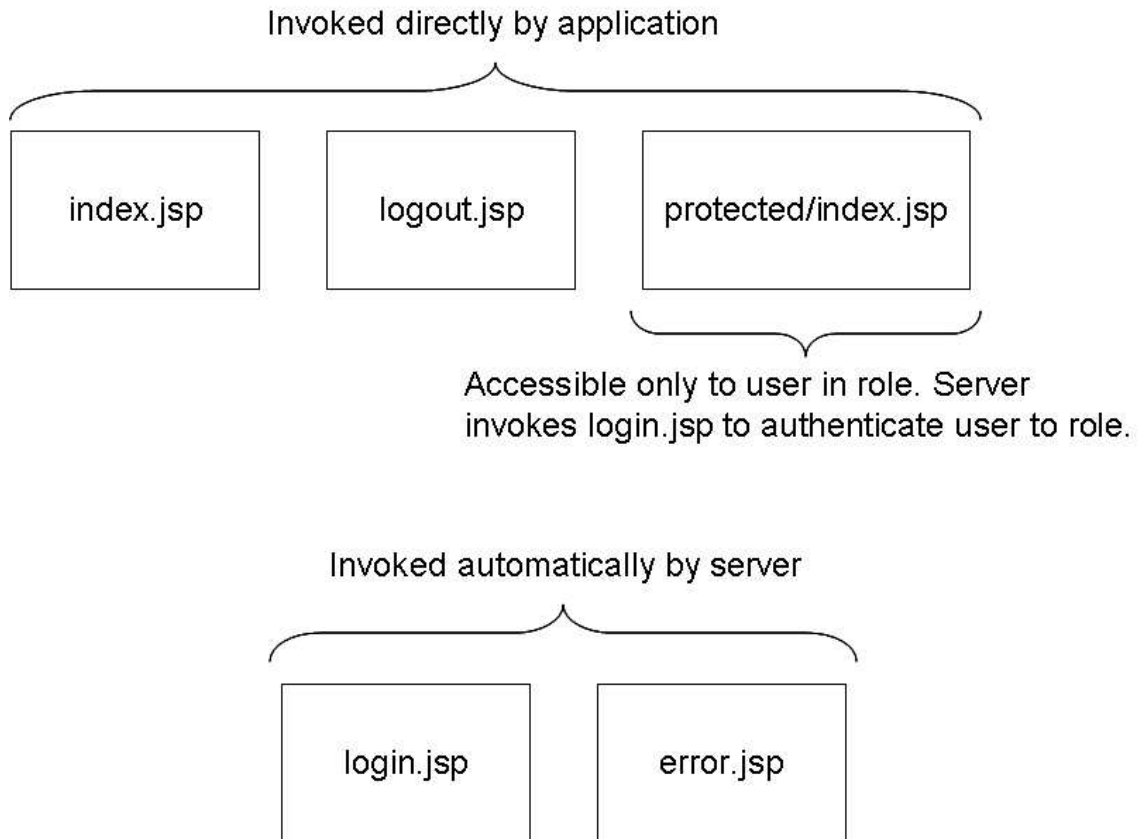


Figure 3

Demonstration

Let us step through the screens of the forms-based authentication example from the Appendix. The context root of the example is **demo** so its access URL on a standard local Tomcat server would be <http://localhost:8080/demo/>. The response is the start page shown in Figure 4. Now we click the link named Enter secured area.



Figure 4

The security information configured in `web.xml` causes the Web container to return the login page defined for the secured area. The container interposes the login page because the user is anonymous and thus not known to be in a role allowed to access the secured area. See Figure 5. Notice the lock icons. The connection is secured by SSL (the browser certificate verification popup is not shown here). We specified SSL so that the user ID and password would not be exposed in clear text on the transport in clear text. The security constraint URL pattern in `web.xml` requested a secure connection with the following element:

```
<transport-guarantee>CONFIDENTIAL</transport-guarantee>
```

This page contains the HTML form having well-known element names that is called by the Web container during form-based authentication. Let us proceed by first clicking the login button without entering a user ID and password.

Form-Based Authentication

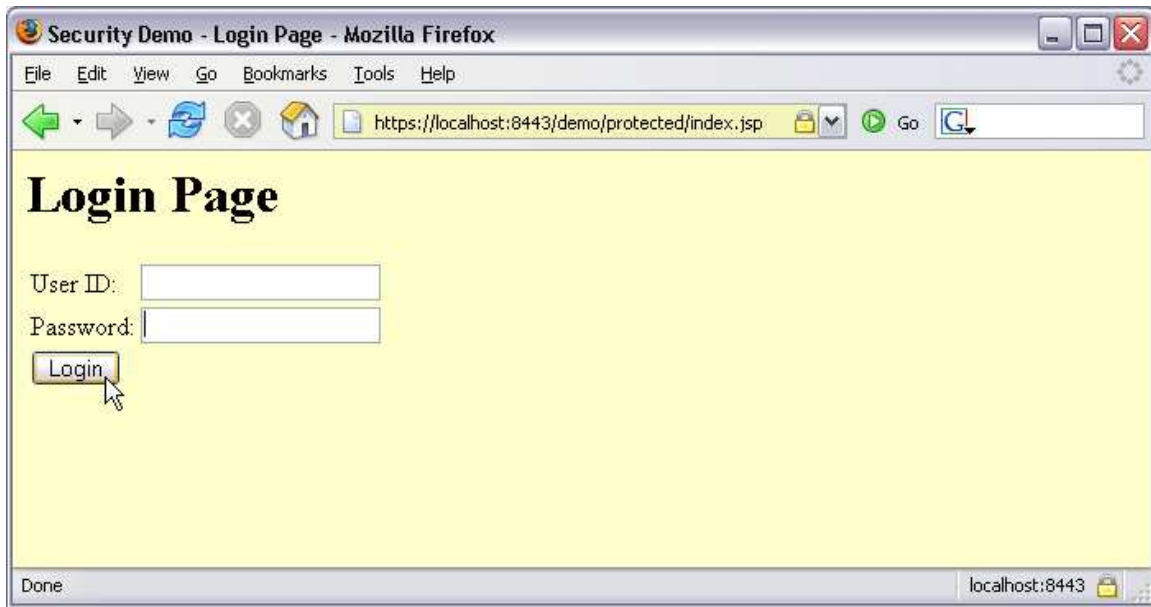


Figure 5

The demo Web application defines an error page for form-based authentication as rendered in Figure 6. The Web container automatically invokes this page if the user ID and password that the user supplied in the login form are not found in the server's principal registry. This internal container action happens in response to an HTTP error status 401.

Our example error page invites the user to return to the start page. It could have been coded to silently redirect to the login page or to carry out any desired recovery tactic.

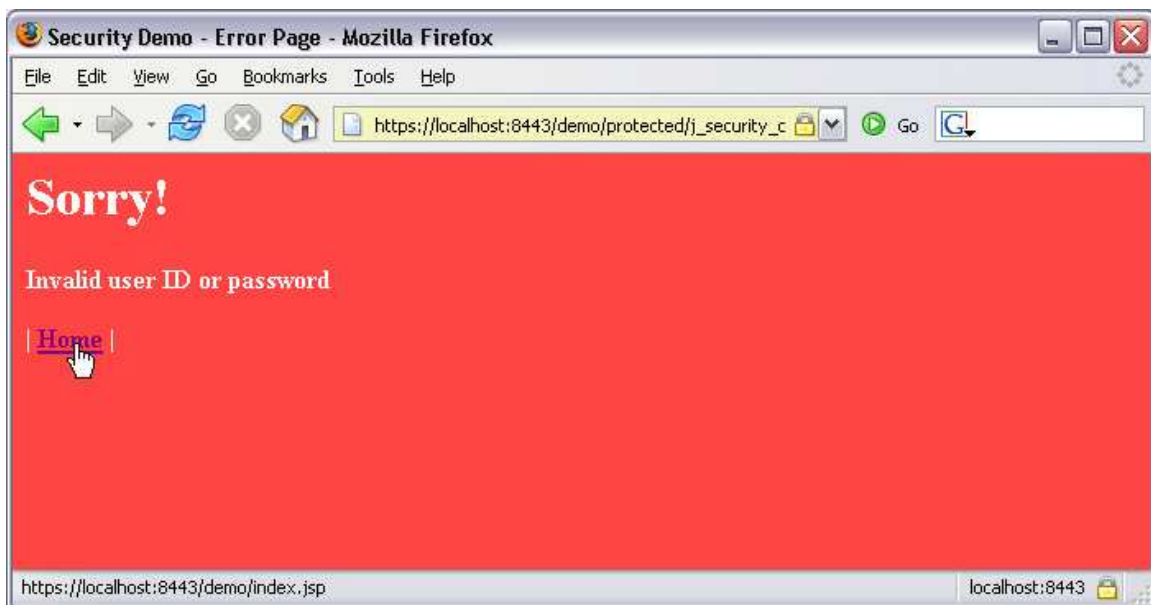


Figure 6

Form-Based Authentication

We decide to click the [Home](#) link to return to the start page for another try. Figure 7 shows the start page again. Notice that the connection remains secured by SSL.



Figure 7

Let us click the [Enter secured area](#) link again (this tiny example presents no other choice). The Web container again imposes the login page before letting us into the secured area. This time we shall enter a valid user ID and password as shown in Figure 8.



Figure 8

Voila! We've been allowed into the secured area. Remember this was defined by a URL matching pattern in `web.xml`. Refer to Figure 9. Notice that the page shows the user name

Form-Based Authentication

and privilege role, and that we were authenticated by form-based authentication. This information came from the *HttpServletRequest* object supplied by the Web container.



Figure 9

At this point, we click the Home link to return to the start page shown in Figure 7, then click Enter secured area again. This time the Web container does not interpose the login page because we are still logged in as a known user in an allowed role. This state is held in the HTTP session within the Web container. Whenever the session becomes invalidated, we will no longer be logged in.

The Log out link invokes a `logout.jsp` that silently invalidates the session and then redirects to the start page using a URL that contains a logout message parameter. Figure 10 shows the result. Notice that the page renders the message and that it is still using SSL. If we were to enter the secured area, the container would now interpose the login form because we are anonymous again.

This concludes the demonstration description. The following section summarizes what we've covered.



Figure 10

Summary

- We discussed how a J2EE Web application can protect some of its resources from access by other than an authenticated user.
- The authenticated user must be a member of a configured role to gain access to the resource.
- Security constraints defined in WEB-INF/web.xml specify what is protected, what user roles can access it, whether to secure the transport, and how the user is identified or authenticated to a role.
- We compared form-based authentication to basic authentication. Form-based authentication means that the Web container interposes an application-specified page to present the login form to the user whenever that user tries to access a secured resource.
- Finally, we stepped through a simple example of forms-based authentication, discussing the action as we proceeded.
- The appendix contains the source listings for the files that comprise the example.

Appendix

This appendix contains the source code for the example application.

Environment

The application was tested on Apache Tomcat. The application files were created using a simple text editor. The environment configuration follows:

1. Java SDK 1.4.2.
 - The `keytool` command creates the `.keystore` file for ApacheTomcat in the as described in Tomcat's `/config/server.xml` file.
 - The `jar` command created the `demo.war` file for deployment to Tomcat.
 - Apache Tomcat used the 1.4.2 JSDK. A JRE alone won't work because the Web container must compile the JSPs.
2. Apache Tomcat 5.0 server. Almost any 4 or 5 version of Tomcat should work.
3. Follow the instructions in Tomcat's `/config/server.xml` file to enable SSL port 8443for the secure transport.

Web Application

The following sections list the files used to create the example application. We built them in a directory named `c:\demo` and then used the following command issued from `c:\`

```
jar -cvf demo.war -C demo .
```

This produces the `demo.war` file. Copy this into Tomcat's `/webapps` directory to deploy the demo application. You can start Tomcat from its root directory by issuing

```
bin\startup.bat
```

or

```
bin/startup.sh
```

Access the application through the following URL:

```
http://localhost:8080/demo/
```

/WEB-INF/web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>

  <display-name>Security Demo</display-name>
  <description>Forms-based authentication demo</description>
```

Form-Based Authentication

```
<security-constraint>
  <display-name>Demo Security Constraint</display-name>

  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <!-- Define the context-relative URL(s) to be protected -->
    <url-pattern>/protected/*</url-pattern>
    <!-- If you list http methods, only those methods are protected -->
    <http-method>DELETE</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
    <http-method>PUT</http-method>
  </web-resource-collection>

  <auth-constraint>
    <!-- Anyone with one of the listed roles may access this area -->
    <role-name>role1</role-name>
    <role-name>tomcat</role-name>
    <role-name>user</role-name>
    <role-name>manager</role-name>
  </auth-constraint>

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

<!-- Default login configuration uses form-based authentication -->
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>Form-Based Authentication Area</realm-name>
  <form-login-config>
    <form-login-page>/login.jsp</form-login-page>
    <form-error-page>/error.jsp</form-error-page>
  </form-login-config>
</login-config>

<!-- Security roles referenced by this web application -->
<security-role>
  <role-name>user</role-name>
</security-role>
<security-role>
  <role-name>manager</role-name>
</security-role>
<security-role>
  <role-name>role1</role-name>
</security-role>
<security-role>
  <role-name>tomcat</role-name>
</security-role>
</web-app>
```

/index.jsp

```
<html><head><title>Security Demo</title></head>
  <body bgcolor="#ffffcc">
    <h1>Security Demo</h1>
    <% if (request.getParameter("msg") != null) { %>
      <p><b><%=request.getParameter("msg")%></b></p>
    <% } %>
    <p><a href="/protected/index.jsp">Enter secured area</a></p>
  </body>
```

Form-Based Authentication

```
</html>
```

/login.jsp

```
<html><head><title>Security Demo - Login Page</title></head>
  <body bgcolor="#ffffcc">
    <h1>Login Page</h1>
    <p>
      <form method="POST" action="j_security_check">
        <table border="0">
          <tr><td> User ID: </td><td>
            <input type="text" name="j_username">
          </td></tr>
          <tr><td> Password: </td><td>
            <input type="password" name="j_password"></td></tr>
          <tr><td colspan="2">
            <input type="submit" name="logon" value="Login">
          </td></tr>
        </table>
      </form>
    </p>
  </body>
</html>
```

/error.jsp

```
<html><head><title>Security Demo - Error Page</title></head>
  <body bgcolor="#ff4444" text="white">
    <h1>Sorry!</h1>
    <p><b>Invalid user ID or password</b></p>
    <b>|&nbsp;  <a href=" ../index.jsp">Home</a>&nbsp;  |</b>
  </body>
</html>
```

/logout.jsp

```
<%
  session.invalidate();
  response.sendRedirect("index.jsp?msg=You%20are%20logged%20out.");
%>
```

/protected/index.jsp

```
<html><head><title>Security Demo</title></head>
  <body bgcolor="#ccffcc">
    <h1>Security Demo</h1>
    <p>Hello
    <b><%=request.getUserPrincipal().getName()%></b>!

    Welcome to the inner circle.</p><p>You are authorized
    to have

    <%
    if (request.isUserInRole("manager")) {
      %>
      <b>Manager</b>
      <%
    } else {
      %>
      <b>User</b>
      <%
    }
    %>
    privileges.
```

Form-Based Authentication

```
</p>
<p>We used <b><%= request.getAuthType() %></b>
authentication.</p>
|&nbsp;<a href=" ../index.jsp">Home</a>&nbsp;<br>
|&nbsp;<a href=" ../logout.jsp">Log Out</a>&nbsp;<br>
</body>
</html>
```

References

- **Implementing WebSphere security through LDAP;** WebSphere Developers' Journal, May-June, 2003; Louis E. Mauget; a two-part article featuring form-based authentication with an LDAP registry;
http://www.findarticles.com/p/articles/mi_m0MLX/is_5_2/ai_102343180 and
http://www.findarticles.com/p/articles/mi_m0MLX/is_6_2/ai_104209580/pg_2
- **Core Servlets and JavaServer Pages, Vol 1: Core Technologies, 2nd Edition;** Prentice Hall; © 2003; Marty Hall and Larry Brown; ISBN: 0130092290; Covers the security subject and much more.
- **The J2EE™ Tutorial;** Web-Tier Security; Sun Microsystems;
http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Security4.html; Brief, but accurate
- **J2EE Form-based Authentication;** OnJava.com; Prabu Arumugam, 06/12/2002;
<http://www.onjava.com/pub/a/onjava/2002/06/12/form.html> ; An excellent tutorial