## Adding Security to an Application

Some applications are just used by their developers, but others are made available to a number of clients. These people may either be in the same company or somewhere on the Internet. For these applications, it is often useful to have levels of access or at least a login involving a username and password.

There are several ways to handle this. The most complete is to develop a custom login and use encryption, such as Secure Socket Layer (SSL). Here usernames and passwords are kept in a secure database, often with encryption. And they are sent over a secure network. This level of security is necessary for financial sites such as banks and brokerage houses.

Other sites require security only when final ordering information, including credit card numbers, is gathered. Up until that point, shoppers or other visitors are free to investigate the site. Some also have registration and login requirements for visitors. These are also usually custom designed.

But a web application can also have levels of security so that, for example, managers could have greater access to web pages than clerks. This can be built into the application using web.xml, the web application deployment descriptor. The Tomcat server can have *roles* assigned to different users so that a manager's role would have greater access than a clerk's role.

**tomcat-users.xml**

The file, tomcat-users.xml, is contained in the *conf* folder of Apache Tomcat. It allows the manager of the server to set up roles for clients.

```
<!--
      NOTE:  By default, no user is included in the "manager" role required
      to operate the "/manager" web application.  If you wish to use this app,
      you must define such a user - the username and password are arbitrary.
-->
<tomcat-users>
      <user name="tomcat" password="tomcat" roles="tomcat" />
      <user name="role1"  password="tomcat" roles="role1"  />
      <user name="both"   password="tomcat" roles="tomcat,role1" />
      <user name="Diana Chen" password="abc123" roles="manager" />
</tomcat-users>
```

The manager role has been added here. It was not in the original file. With it, Diana Chen will have access to material that someone, who is not a manager will not. However, she must know the password, here just the simple string, "abc123".

**web.xml**

For the manager to be recognized by the web application, several lines should be added to web.xml. These are added at the end, after the error-page tags.

```
<web-app>
      …
      <servlet>
            <servlet-name>ManagerServlet</servlet-name>
            <servlet-class>manage.ManagerServlet</servlet-class>
      </servlet>
      …
```

```
<servlet-mapping>
        <servlet-name>ManagerServlet</servlet-name>
        <url-pattern>/manager/*</url-pattern>
</servlet-mapping>
…
<error-page>
        <error-code>404</error-code>
        <location>/notfound.html</location>
</error-page>

<!--  The Security Constraint for this Application.  -->
<security-constraint>
        <web-resource-collection>
                <web-resource-name>Application Manager</web-resource-name>
                <url-pattern>/manager/*</url-pattern>
                <http-method>POST</http-method>
                <http-method>GET</http-method>
        </web-resource-collection>
        <auth-constraint>
                <role-name>manager</role-name>
        </auth-constraint>
</security-constraint>

<!--  The Login Configuration for this Application  -->
<login-config>
        <auth-method>FORM</auth-method>
        <realm-name>Application Manager</realm-name>
        <form-login-config>
                <form-login-page>/managerLogin.html</form-login-page>
                <form-error-page>/managerLoginError.html</form-error-page>
        </form-login-config>
</login-config>
```

The web-resource-collection contains the servlets that will be constrained.  Here there is only one.  The <auth-constraint> provides the role name of the client that will have access to these servlets.  All the web pages supplied to the manager should be dynamic, that is they should be created by the servlets, rather than having the pages stored statically in the main application directory.

Because the web pages have to be created by servlets, you have to call a servlet in order to access the login.  Something similar to the following code should be placed in the index file.

```
<form method = "post" action="../application_name/manager">
        <input type = "submit" value = "Manager Login" /></td>
</form>
```

There are two kinds of login configurations.  The one above is for a login form.  The names used in the form are defined by the server.  The action value must be j_security_check, the username, j_username, and the password, j_password.  The login form looks like the following after Diana Chen enters her username and password.


```
<html>
<head><title>Login Form</title></head>

<body>
	<center><h2>Please Login</h2>
	<form method = "post" action = "j_security_check">
		Username: <input type = "text" name = "j_username" /><br />
		Password: <input type = "password" name = "j_password" /><br />
		<br /><input type = "submit" value = "Login" />
		<input type = "reset" value = "Reset" />
	</form></center>
</body></html>
```

If the login is incorrect, managerLoginError.html will be displayed.

```
<html>
<head><title>Incorrect Login Form</title></head>

<body>
	<center><h2>Your login was incorrect.  Please return to the login page.</h2>
	<p><form method = "post" action="../application_name/manager">
	<input type = "submit" value = "Manager Login" /></p></center>
</body></html>
```

Note that you cannot use a direct reference back to the manager's login page here.

The ManagerServlet is used to display forms that the manager can use to find and change values. An example of one of the methods follows:

```
	private void changeForm (PrintWriter out)
	{
		out.println ("<form method = 'get' action = '../application_name/managerChange'>");
		out.println ("<input name = 'id' type = 'text' value = '' size = '15' />ID<br />");
		out.println ("<input name = 'price' type = 'text' value = '' size = '10' />New Price");
		out.println ("<p><input type = 'submit' value = 'Change Price' /></p>");
	} // changeForm
```

This method creates a form that then calls another servlet when it is submitted.

Tomcat will also provide its own form if web.xml contains the following:

```
	<login-config>
```

```
        <auth-method>BASIC</auth-method>
        <realm-name>Application Manager</realm-name>
    </login-config>
```
Using this gives you less control over the appearance of the page. Both forms encrypt the username and password, but the encryption is very weak.

**More for web.xml**

In addition to the security constraint described above, web.xml allows you to designate particular servlets that will be protected. While clerks may be given permission to do a number of things, they probably are not allowed to change prices. We can put a constraint on the servlet that does the price change in the web.xml file.

```
<servlet>
    <servlet-name>ChangeServlet</servlet-name>
    <servlet-class>manage.ChangeServlet</servlet-class>
    <security-role-ref>
        <role-name>mgr</role-name>
        <role-link>manager</role-link>
    </security-role-ref>
    <security-role-ref>
        <role-name>clerk</role-name>
        <role-link>clerk</role-link>
    </security-role-ref>
</servlet>
```

The role-name and role-link entries allow for different names to be used in the servlet and the <auth-constraint> entry. Here "mgr" will be used in the servlet while "manager" is used in the authentication constraint entry. The servlet can ask whether a user is in the role of a manager or a clerk. It can them differentiate between what each is allowed to do.

The code that checks for the role is

```
    boolean manager = request.isUserInRole ("mgr");
```

If the user that logged in was listed as a manager, the transaction will be allowed. Otherwise it will not be authorized. Note that here "mgr" is used rather than "manager". The <role-link> tag provided this connection.

```
//  If the user is a manager, ChangeServlet gets data from the request and changes the product's price.

package manage;

import java.sql.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/*  If the user is a manager, ChangeServlet finds a specific product and changes the price. */
public class ChangeServlet extends HttpServlet
{
    public void doGet (HttpServletRequest request, HttpServletResponse response)
    {
        PrintWriter out = null;
```

```
            int success = 0;
            String table = null;
            try
            {
                  out = response.getWriter ();
                  Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
                  Connection con = DriverManager.getConnection ("jdbc:odbc:store");
                  Page.createHeader (out, "Change Price");

                  String id = request.getParameter ("id");
                  double newPrice = Double.parseDouble (request.getParameter ("price"));
                  // This application has several tables.
                  String [] tables = request.getParameterValues ("table");
                  for (int count = 0; count < tables.length; count++)
                        if (tables [count] != null) table = tables [count];

                  // This checks whether the user is authorized to make this transaction.
                  boolean manager = request.isUserInRole ("mgr");
                  if (manager)
                  {
                        success = changePrice (out, con, id, table, newPrice);
                        if (success != 0) out.println ("Price Changed.");
                        else out.println ("Error, price not changed.");
                  }
                  else out.println ("Transaction not allowed.");

                  Page.createFooter (out);
            }  catch (IOException ex) {out.println ("<h3>IO Exception.</h3>");}
            catch (ClassNotFoundException e) {System.out.println ("Class Not Found Exception.");}
            catch (SQLException es) {System.out.println ("SQL Exception");}
      } // doGet

      // changePrice locates a product in the database and changes the price.
      public int changePrice (PrintWriter out, Connection con, String id, String table, double newPrice)
      {
            int success = 0;
            try
            {
                  Statement stmt = con.createStatement ();
                  String query =
                  "Update " + table + " Set price = '" + newPrice + "' Where id = '" + id + "'";
                  success = stmt.executeUpdate (query);
                  stmt.close ();
            } catch (SQLException es) {out.println ("SQL Exception");}
            return success;
      } // changePrice
} // ChangeServlet
```

As you can see, Tomcat provides some security for an application or even for particular servlets in an application. The username and password are encrypted when you use the special login form, but this is very weak encryption. For a real store, particularly one that operates over the Internet, the company

should invest in SSL. SSL uses public key encryption, and the key must be registered with some company such as Verisign.

Reference

1. Karl Moss, *Java Servlets Developer's Guide*, chapters 4 and 5, McGraw-Hill/Osborne, 2002.