Here is a **complete working example** of **Custom Authentication (Programmatic Login)** in a Tomcat web application using HttpServletRequest.login().

This avoids <login-config>-based declarative security and handles login manually (e.g., for custom login forms or REST APIs).

**✅ 1. web.xml Configuration**

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

version="4.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee

http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd">

<servlet>

<servlet-name>LoginServlet</servlet-name>

<servlet-class>com.example.LoginServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>LoginServlet</servlet-name>

<url-pattern>/login</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>SecureServlet</servlet-name>

<servlet-class>com.example.SecureServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>SecureServlet</servlet-name>

<url-pattern>/secure</url-pattern>

</servlet-mapping>

<security-constraint>

<web-resource-collection>

<web-resource-name>Secure Area</web-resource-name>

<url-pattern>/secure</url-pattern>

</web-resource-collection>

<auth-constraint>

<role-name>user</role-name>

</auth-constraint>

</security-constraint>

<security-role>

<role-name>user</role-name>

</security-role>

</web-app>

**✅ 2. tomcat-users.xml (Tomcat Configuration)**

Located in: TOMCAT\_HOME/conf/tomcat-users.xml

<tomcat-users>

<role rolename="user"/>

<user username="admin" password="admin123" roles="user"/>

</tomcat-users>

⚠️ Ensure that Tomcat is **restarted** after editing this file.

**✅ 3. LoginServlet.java**

package com.example;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.\*;

@WebServlet("/login")

public class LoginServlet extends HttpServlet {

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

String username = request.getParameter("username");

String password = request.getParameter("password");

try {

request.login(username, password);

response.sendRedirect("secure");

} catch (ServletException e) {

response.getWriter().println("Login failed: " + e.getMessage());

}

}

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html");

response.getWriter().println("<form method='post'>");

response.getWriter().println("Username: <input type='text' name='username'><br>");

response.getWriter().println("Password: <input type='password' name='password'><br>");

response.getWriter().println("<input type='submit' value='Login'>");

response.getWriter().println("</form>");

}

}

**✅ 4. SecureServlet.java**

package com.example;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.\*;

@WebServlet("/secure")

public class SecureServlet extends HttpServlet {

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.getWriter().println("Welcome, " + request.getRemoteUser());

response.getWriter().println("<br><a href='logout'>Logout</a>");

}

}

**✅ 5. Optional: LogoutServlet.java**

package com.example;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.\*;

import java.io.IOException;

@WebServlet("/logout")

public class LogoutServlet extends HttpServlet {

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

request.logout();

response.sendRedirect("login");

}

}

**✅ Deployment Structure**

webapps/

└── CustomAuthApp/

├── WEB-INF/

│ ├── classes/

│ │ └── com/example/\*.class

│ └── web.xml

Compile the Java files into WEB-INF/classes/com/example/ with proper package structure.

**✅ Access in Browser**

1. Start Tomcat.
2. Go to: http://localhost:8080/CustomAuthApp/login
3. Use username/password: admin/admin123
4. After login → <http://localhost:8080/CustomAuthApp/secure>

**✅ Bonus Tip: View the Logged-In User’s Roles**

Add this to SecureServlet.java to debug roles:

response.getWriter().println("User: " + request.getRemoteUser());

response.getWriter().println("<br>Has role 'user': " + request.isUserInRole("user"));

this is an important distinction in how **container-managed security** (like Tomcat with <security-constraint>) works with:

* response.sendRedirect(...) ✅ browser-visible redirect
* request.getRequestDispatcher(...).forward(...) ✅ server-side forward

**✅ Why RequestDispatcher.forward(...) Works (but sendRedirect(...) Does Not)**

When you use **request.login(username, password)**, it authenticates the session *on the current request*. If you immediately call:

response.sendRedirect("secure/Welcome.jsp");

...you create a **new request**, and the container may not carry over the authenticated session yet — especially if you're using FORM-based login or **programmatic login** (request.login()).

But if you use:

request.getRequestDispatcher("secure/Welcome.jsp").forward(request, response);

You're forwarding **within the same request context**, so the authenticated identity and roles are already available.

**🛠 Best Practice (Secure Post-login Navigation)**

After programmatic login:

try {

request.login(username, password); // authenticates user

request.getRequestDispatcher("/secure/Welcome.jsp").forward(request, response);

} catch (ServletException e) {

request.setAttribute("error", "Login failed: " + e.getMessage());

request.getRequestDispatcher("/login.jsp").forward(request, response);

}

✅ This ensures the **authenticated user** is available to the secure JSP or servlet.

**🔁 Optional Redirect (if session propagation is handled)**

If you *really* need to use sendRedirect, you must ensure:

1. A valid session is created
2. Cookies are enabled
3. The session is propagated correctly

HttpSession session = request.getSession(true); // ensure session exists

request.login(username, password);

response.sendRedirect(request.getContextPath() + "/secure/Welcome.jsp");

But this is **less reliable** than using forward() immediately after login().