Adv. Java means DURGA SIR..

# ADV.JAVA With SCWCD / OCWCD

## Servlets Material

## 5. Web Application Security

DURGA M.Tech

(Sun certified & Realtime Expert)

Ex. IBM Employee

Trained Lakhs of Students for last 14 years across INDIA

India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

# Web Application Security

1) Basic Terminology
    - Authentication
    - Authorization
    - Data Integrity
    - Confidentiality

2) Types of Authentication
    - BASIC
    - DIGEST
    - Form-based
    - Https client-cert

3) Declarative Security
4) Programmatic Security

## Basic Terminology :

Based on Servlet specification compare the following Security mechanisms

1) Authentication:
   It is process of validating the user, we can implement authentication by using userName and Password
   Ex: providing userName and password to login into bank site is called Authentication.

2) Authorization :
   It is process of validating access permissions of a user i.e, It is process of checking whether user allowed to access a perticular resource or not after authentication we have to perform authorization by using Access Control List (ACL) we can implement authorization
   Ex: we are not allowed to access some other accounts information even though we are valid member of the bank i.e., we are not authorized to access some other account information.

3) Data Integrity:
   It is process of ensuring that data should not be changed in transformation from client to server by using Secure Socket Layer(SSL) , we can implement Data Integrity .

4) Confidentiality :
   It is process of ensuring that no one except intended user is able to understand our information. We can achieve this confidentiality by using encryption mechanisms .

**What is difference between Authorization and Confidentiality ?**

**Authorization prevents information reaching and unintended users at beginning only , where as Confidentiality ensure that even though information falls in wrong hand it stills remains unreachable.**

**Ex:Beer website application**

1. **should have userName and password to access this application(authentication).**
2. **Only premium users can get 10% discount (authorization) .**
3. **When ever user places an order some sort of confermation is required(Data Integrity).**
4. **When ever customer places or type credit card information should send encrypted form and should not be misused (Confidentiality).**

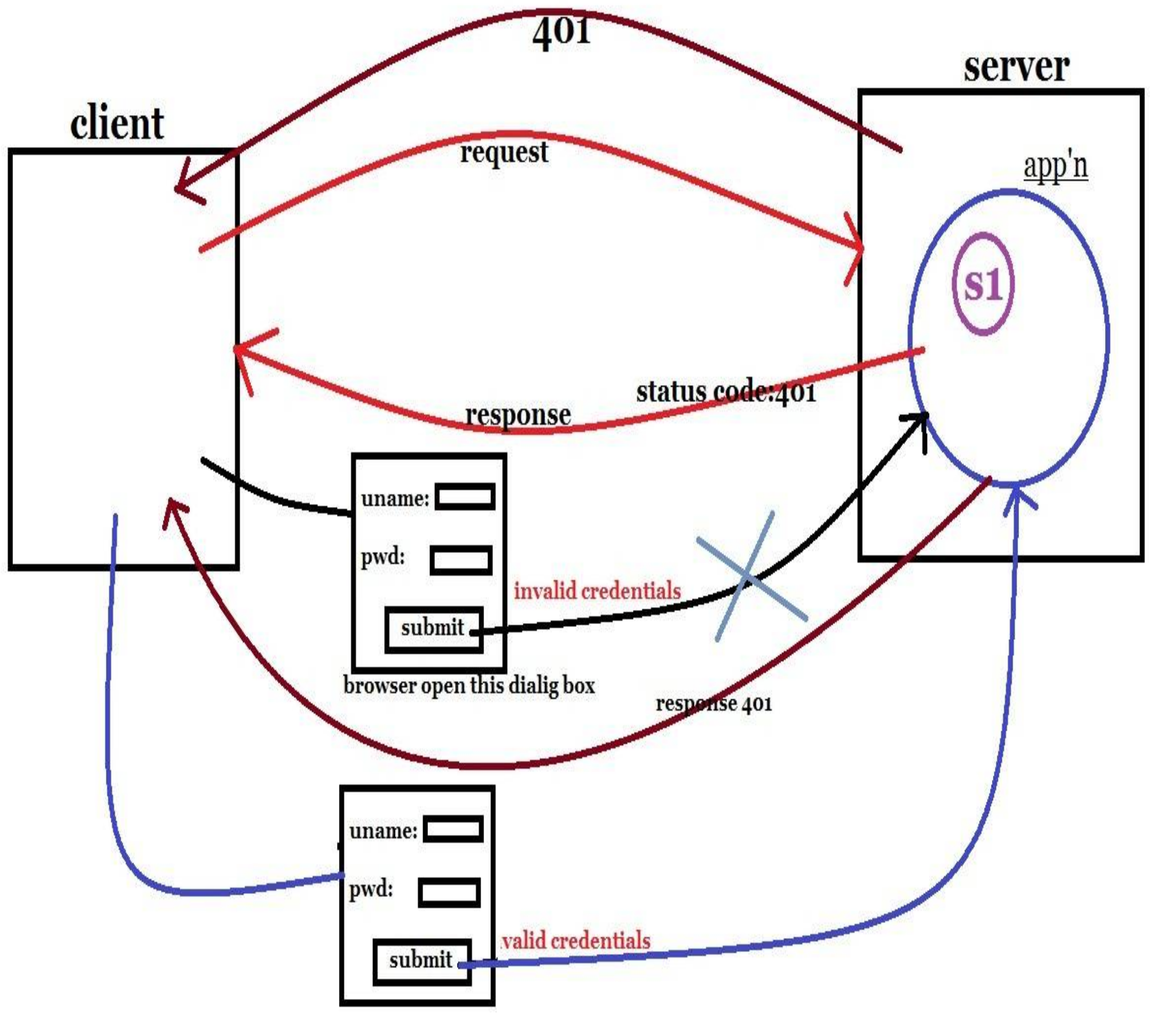
## *Types of Authentication Mechanism :*

**According to servlet specification 4 types of authentication mechanism possible.**

### HTTP BASIC Authentication Mechanism :

**It is the most simplest and commonly used Authentication mechanism.**
**The Basic Authentication Mechanism introduced in http 1.1 specification process.**

**Basic Authentication Mechanism graphical representation:**



1) **Browser sends a request to the server at this time browser don't know whether the request is protected or not , hence it sends normal request .**

2) Server identifies the requested resource is secure hence instead of sending required response hence it will send 401 status code saying it requires authentication.
3) By receiving the 401 status code browser opens a dialog box prompting userName and password , once the end user enter userName and password browser resend the request to server with credentials.
4) When the server receives the request it validate the userName and password the credentials are valid server sends proper required response otherwise it will send 401 status code application. (this process may happen maximum 3 number of times if still if your giving wrong userName and password this time 401 status code sending to the end user . )

ADVANTAGES :
1) It is very easy to implement and setup.
2) All browsers and all web servers can provide support for this mechanism.

LIMITATIONS :
1) user-name and password will send in plain text form from client to server , hence security is very less in this authentication mechanisms, this authentication mechanism follows BASE64 encoding technique/algorithum
2) We can't customize LookUP and Feel of dialog box

## DIGEST AUTHENTICATION MECHANISM:

It is exactly similar to Basic except the password is sending encrypted form, this encryption makes more secure.

ADVANTAGE:
when compare to Basic Authentication mechanism DIGEST Authentication mechanism is more secure.

LIMITATIONS:
1) Only few browsers can provide support for this mechanism
2) Browser is responsible for this mechanism
3) Most of the browsers doesn't provide support for this DIGEST mechanism because servlet specification doesn't tell it is mandatory tecnic .
(My web server doesn't know which encryption algorithm followed by browser , so that it gives decryption problems )
4) We can't customize Look and Feel of dialog box.

## FORM-BASED AUTHENTICATION MECHANISM:

1) This mechanism is exactly similar to Basic authentication mechanism except that insteadof depending on browser's dialog box , we can provide our own login frm or our own dialog box.
2) Developer is responsible to provide login and error pages , so that we can customize LOOK and FEEL based on our requirement.
3) The only requirement for the login form is :
   - The value of action attribute should be 'J_Security_Check'

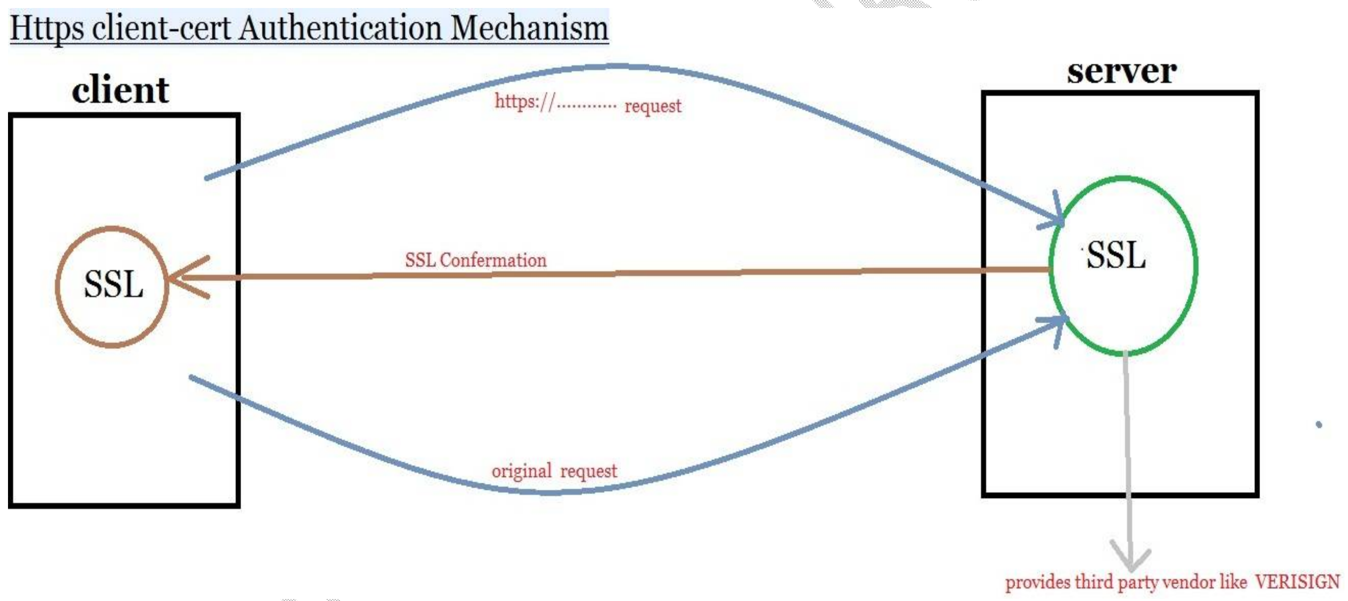
- The form compulsory should contains 2 text fields with the names 'J_username' and 'J_password' except these all the remaining things are customizable.

**ADVANTAGES:**
It is very easy to setup ,All browsers and all web servers can provide support for this mechanism. we can customize login form based on our requirement.

**LIMITATIONS:**
In this case also user-name and password will send in plain text form from client to server , hence security is very less .

## HTTPS CLIENT-CERT AUTHENTICATION MECHANISM:

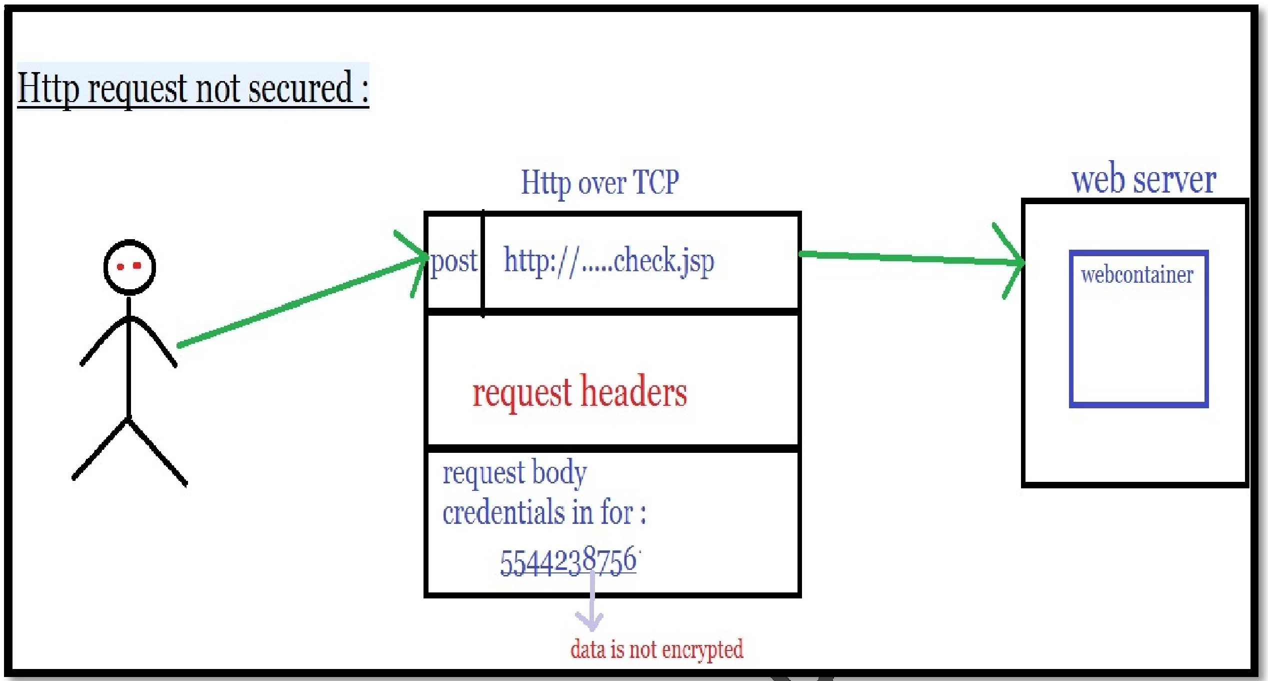
This is most secure type of authentication and it is most commonly used type of real time.
HTTP means Http over secure socket layer.

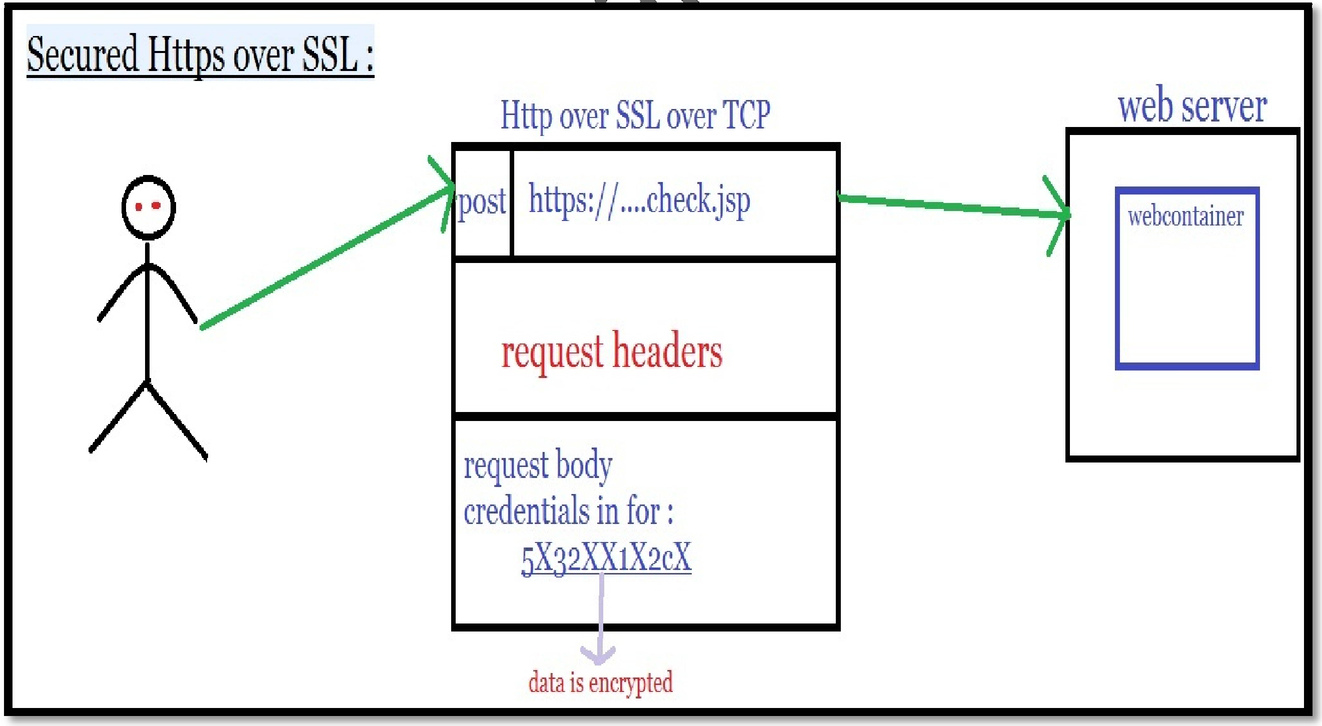

Https client-cert Authentication Mechanism
client
server
https://........... request
SSL Confermation
SSL
SSL
original request
provides third party vendor like VERISIGN

SSL is a protocol to ensure the privacy of sensitive data transmitted over the internet.
In the mechanism Authentication is perform when SSL connection is established between client and server.

**ADVANTAGES:**
It is most secure type of authentication.

**DIS ADVANTAGES:**
It is costly to implement and maintain.
It receives a certificate from 3rd party vendors like verisign.


Http request not secured :
Http over TCP
post
http://.....check.jsp
request headers
request body
credentials in for :
5544238756
data is not encrypted
web server
webcontainer


Secured Https over SSL :
Http over SSL over TCP
post
https://....check.jsp
request headers
request body
credentials in for :
5X32XX1X2cX
data is encrypted
web server
webcontainer

| Type | Specification | DataIntegrity | Comments |
|---|---|---|---|
| BASIC | Http | Base 64-encoding | All browsers can provide support. |
| DIGEST | Http | Strong - but not SSL | Browsers and servers may not be support. |
| FORM BASED | J2EE | Very less security/weak encryption | We can customize login page . |
| CLIENT-CERT | J2EE | Strong-SSL(public key) | Strong , Users must have certificates . |

## DECLARATIVE SECURITY:

objective: In the deployment descriptor declare ......

1) < security-constraint >
2) < login-config >
3) < security-role >

We can implement web security mainly by using following 3 tags.

1) < security-constraint >
It defines which resource has to be protected,
which roles are allowed to access and how to resource is transported across the network

2) < login-config >
It defines the type of authentication what we are using

3) < security-role >
It defines Security roles which are allowed in web application

All the above 3 tags are direct child tags of < web-app >
hence we can take any where with in the < web-app > tag but it is convention to take the above order only.

## < security-constraint >

It defines the following 3 child tags.

1) <web-resource-collection >
It defines which resource has to be protected

2) <auth-constraint >
Authorization constraints which determines what roles are allowed to access these resources .

3) < user-data-constraint >
What type of production is required when transport the resource across network.

## <web-resource-collection >

It contains there are 4 child tags.

1) <web-resource-name >
2) < description >
3) < url-pattern >
4) < http-method >
   If we are specifying this tag then security constraint is applicable for all http methods (like GET, POST , HEAD , DELETE , TRACE , PUT , OPTIONS )

## < auth-constraint >

It defines the following 2 child tags.

1) < description >
2) < role-name >

## < user-data-constraint >

It defines the following 2 child tags.

1) < description >
2) < transport-guarantee >

This tag specifies what type of guarantee we have to provide while transporting the resource across network.
The allowed values for these tags are

NONE: it means the data is transported in plain-text form , it is default value .
INTEGRAL: it means the data should not be changed in transmission .
CONFIDENTIAL: it means the data is transported in encrypted form.

The priority order is : CONFIDENTIAL > INTEGRAL > NONE

Note: we have only one < user-data-constraint > per < security-constraint >

## < login-config >

This tag specifies what type of authentication , we are using it contain the following 3 child tags.

1) < auth-method >
   It represents authentication methods the allowed values are BASIC , DIGEST , FORM , CLIENT-CERT

2) < relian-name >
   It represents location where we are storing authentication information. It is required only for BASIC authentication.

3) < form-login-config >
   This tag is required to specify login page and error page in case of form based authentication mechanism.
   This tag contains 2 child tags .

- < form-login-page >
- < form-error-page >

**< security-role >**

It can be used to defines security roles in web-applications
This tag contains the following 2 child tags.

- < description >
- < role-name >

**Note:** if we want to configure multiple roles that time each role a separate < security-role > tag is required.

**Example:**

```
< security-role >
        < role-name > jobs4times < / role-name >
< / security-role >
< security-role >
        < role-name > ashok < / role-name >
< / security-role >
```

**Write a program BASIC authentication mechanism:**

http://localhost:7001/scwcd/login.jsp

**login.jsp**

```
<form action="/BasicAuthentication" method="post">
    Enter name : <input type="text" name="uname"><br>
    <input type="submit" value="submit">
</form>
```

**BasicAuthentication**

```
package com.jobs4times;
public class BasicAuthentication implements HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
                                                            ServletException, IOException {
        System.out.println("We are in doGet( ) method ");
        System.out.println("Get: After Authentication only we can access this servlet " );
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws
                                                            ServletException, IOException {
            System.out.println("We are in doPost( ) method ");
            System.out.println("Post: After Authentication only we can access this servlet " );
    }
}
```

**web.xml**

```
<web-app>

     <servlet>
          <servlet-name>BasicAuthentication</servlet-name>
          <servlet-class>com.jobs4times.BasicAuthentication</servlet-class>
     </servlet>

     <servlet-mapping>
           <servlet-name>BasicAuthentication</servlet-name>
           <url-pattern>/BasicAuthentication</url-pattern>
     </servlet-mapping>

     <security-constraint>
          <web-resource-collection>
                <web-resource-name>basic</web-resource-name>
                <url-pattern>/BasicAuthentication</url-pattern>
                <http-method>GET</http-method>
                <http-method>POST</http-method>
          </web-resource-collection>
          <auth-constraint>
                 <role-name>adminrole</role-name>
                 <role-name>ashokrole</role-name>
          </auth-constraint>
          <user-data-constraint>
                <transport-guarantee>CONFIDENTIAL</transport-guarantee>
          </user-data-constraint>
    </security-constraint>

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

   <security-role>
        <role-name>ashokrole</role-name>
   </security-role>

   <security-role>
        <role-name>adminrole</role-name>
    </security-role>

</web-app>
```

**tomcat-users.xml**

```
<tomcat-users>
       <role rolename="adminrole"/>
       <role rolename="ashokrole"/>
       <user name="admin" password="scjp" roles="adminrole" />
      <user name="ashok" password="scwcd" roles="ashokadmin" />
       <!-- <user name="admin" password="scjp" roles="adminrole,ashokarole" />
              //both roles same username , pwd     -->
</tomcat-users>
```

**Rules for auth-constraint :**

1) With in a security-constraint element auth-constraint element is an optional.
2) If an auth-constraint exists the container must perform authentication for associated URL's
3) If an auth-constraint exists doesn't exist the container must allowed unauthenticated access for the URL's

**Ex:**

1) < auth-constraint >
   < role-name > guest < /role-name >
   < role-name > admin < /role-name >
   < /auth-constraint >
   Note: guest-role and admin-role can access
2) < auth-constraint >
   < role-name >*< /role-name >
   < /auth-constraint >
   Note: any role can access.
3) < auth-constraint / >
   Note: no body can access.
4) If no < auth-constraint > in web.xml , that time every body can access , container must allow unauthenticate access for the URL's .

**Rules for < role-name > tag :**

1. With in the < auth-constraint > element < role-name > element is optional.
2. If < role-name > element exists, they tell to the container , which roles are allowed.
3. If < auth-constraint > element exists with no < role-name > element , then no users are allowed.
4. if < role-name > * <role-name > Then all users are allowed.

**Key points for <web-resource-collection> :**

1. The <web-resource-collection> element to primary sub elements
   1. url-pattern
   2. http-method(optional , zero or more)

2. The <url-pattern> and <http-method> together define resource request that are constraints to be accessible by the only those roles defined in <auth-constraint>
3. web-resource-name tag is mandatory (even though you probably won't use it , but it is using web-container )
4. description tag is optional.
5. The <url-pattern> element follows same servlet standard naming conventions and rules.
6. You must specify at least one url-pattern but you can have meaning
7. Valid methods for http-method is GET,POST , PUT , HEAD, DELETE, TRACE, OPTION.
8. If no http-method's are specified then all methods will be constraints which means they can be accessed by only roles in <auth-constraint>
9. we can has more than one <web-resource-collection> with in the same <security-constraint> element.
10. The <auth-constraint> element applies all <web-resource-collection> element in the <security-constraint>

## Writa a Program for FORM-BASED AUTHENTICATION MECHANISM:

**login.html**

```
<h3>Form Based Authentication Example :</h3>
<form action="j_security_check" method="post">
      <table border="2" bgcolor="lightgrey"><tr><td>
              UserName: <input type="text" name="j_username"><br></td></tr><tr><td>
              Password:<input type="password" name="j_password"><br></td></tr>
              <tr><td align="center"><input type="submit" value="Submit"></td></tr>
       </table>
</form>
```

**error.html**

```
<body>
     <pre>Your credentials are not correct<br>
              please provide valid credentials</pre>
<body>
```

**post.html**

```
<form method="post" action="./post">
        <input type="submit" >
</form>
```

**FormServlet.java**

```
package com.jobs4times;
public class FormServlet implements HttpServlet {
            public void doGet(HttpServletRequest request, HttpServletResponse response) throws
                                                                     ServletException, IOException {
                    System.out.println("We are in doGet( ) method ");
                    System.out.println("Get: After Authentication only we can access this servlet " );
```

```
          }
          public void doPost(HttpServletRequest request, HttpServletResponse response) throws
                                                                  ServletException, IOException {
                    System.out.println("We are in doPost( ) method ");
                    System.out.println("Post: After Authentication only we can access this servlet " );
          }
}
```

web.xml

```
<web-app>

     <servlet>
          <servlet-name>FormServlet</servlet-name>
          <servlet-class>com.jobs4times.FormServlet</servlet-class>
     </servlet>

     <servlet-mapping>
          <servlet-name>FormServlet</servlet-name>
          <url-pattern>/FormServlet</url-pattern>
     </servlet-mapping>

     <security-constraint>
           <web-resource-collection>
                  <web-resource-name>form</web-resource-name>
                  <url-pattern>/FormServlet</url-pattern>
                  <http-method>GET</http-method>
                  <http-method>POST</http-method>
           </web-resource-collection>
           <auth-constraint>
                <role-name>ashokrole</role-name>
           </auth-constraint>
      </security-constraint>

      <login-config>
           <auth-method>FORM</auth-method>
           <form-login-config>
                <form-login-page>/login.html</form-login-page>
                <form-error-page>/error.html</form-error-page>
           </form-login-config>
      </login-config>

     <security-role>
         <role-name>ashokrole</role-name>
     </security-role>

</web-app>
```

**tomcat-users.xml**

```
<tomcat-users>
	<role rolename="ashokrole"/>
	<user name="ashok" password="scwcd" roles="ashokadmin" />
</tomcat-users>
```

## Programatic Security:

Some times declarative security may not enough to meet programming requirement.

**Ex:**

Based on the user role we have to provide customized response , if the user is admin then we have to provide admin related response.
If the user is manager , then we have to provide manager related response.
To meet this requirement compulsary we should go for programmatic security and Declarative security is notenough.
We can implement programmatic security by using the following methods of HttpServletRequest(I).

1) public boolean isUserInRole(String roleName)
   If the authentication user belongs to specified role then this method returns true , if authenticated user not belongs specified role or if the user has not been authenticated then this method returns false.

2) public string getRemoteUser()
   This method returns authenticated userName(loginName) , if the user has been authenticated then this method returns null .

**Example:**

```
public class FirstServlet implements HttpServlet {
	public void doGet(HttpServletRequest request, HttpServletResponse response) throws
										ServletException, IOException {
		System.out.println("We are in doGet( ) method ");
		if(request.isUserInRole("admin")) {
			//this is admin related response
		} else {
				//this is general response
			}
	}
}
```

The main problem is this approach is ........
we are hardcode in servlet ,
there is any change in the role-name modify the servlet code is costly operation and creates maintanability problems ,
to resolve this we should go for <security-role-ref> in web.xml,
by using this tag we can configure/map hardcoded role-name with original value.

## Write a demo program for programmatic security:

**login.html**

```
<h3>programmatic security:</h3>
<form action="/test" method="get">
        <table border="2" bgcolor="lightgrey"><tr><td>
                UserName: <input type="text" name="uname"><br></td></tr><tr><td>
                Password:<input type="password" name="pwd"><br></td></tr>
                <tr><td align="center"><input type="submit" value="Submit"></td></tr>
          </table>
</form>
```

**FirstServlet.java**

```
public class FirstServlet implements HttpServlet {
      public void doGet(HttpServletRequest request, HttpServletResponse response) throws
                                                                        ServletException, IOException {
                  System.out.println("We are in doGet( ) method ");
                  if(request.isUserInRole("hero")) {
                        out.println("this is hero home page");
                  } else {
                              out.println("this is others home page");
                           }
       }
}
```

**web.xml**

```
<web-app>

      <servlet>
          <servlet-name>first</servlet-name>
          <servlet-class>com.jobs4times.FirstServlet</servlet-class>
          <security-role-ref>
                 <role-name>hero</role-name>     // hardcoded rolename in servlet
                 <role-link>adminrole</role-link>      //original roleName
          </security-role-ref>
      </servlet>

      <servlet-mapping>
             <servlet-name>first</servlet-name>
             <url-pattern>/test</url-pattern>
      </servlet-mapping>

      <security-constraint>
             <web-resource-collection>
                  <web-resource-name>checked</web-resource-name>
                  <url-pattern>/test</url-pattern>
```

```
                <http-method>GET</http-method>
                <http-method>POST</http-method>
            </web-resource-collection>
            <auth-constraint>
                 <role-name>adminrole</role-name>
                 <role-name>ashokrole</role-name>
            </auth-constraint>
    </security-constraint>

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

   <security-role>
         <role-name>adminrole</role-name>
    </security-role>

</web-app>
```

tomcat-users.xml

```
<tomcat-users>
        <role rolename="adminrole"/>
        <role rolename="ashokrole"/>
        <user name="admin" password="scjp" roles="adminrole" />
        <user name="ashok" password="scwcd" roles="ashokadmin" />
</tomcat-users>
```

**Multiple Securuty constraints elements with same name url-pattern ( or partially matching url-patrtern and http-method elements )**

web.xml

```
<web-app >
...................
<security-constraint>
       <web-resource-collection>
              <web-resource-name>display</web-resource-name>
              <url-pattern>/beer/updateReceipts/*</url-pattern>
              <url-pattern>/beer/displayReceipts/*</url-pattern>
              <http-method>POST</http-method>
       </web-resource-collection>

       <auth-constraint>
              //Authorization constraints with different roles
       </auth-constraint>
</security-constraint>
```

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>update</web-resource-name>
        <url-pattern>/beer/updateReceipts/*</url-pattern>
        <url-pattern>/beer/updateUser/*</url-pattern>
        <http-method>POST</http-method>
    </web-resource-collection>

    <auth-constraint>
        //Authorization constraints with different roles
    </auth-constraint>

</security-constraint>

</web-app>
```

How should container handle authorization when the same resource used by more than one < security-constraint >

| Content of A | Content of B | Whose has Access "Update Receipts" |
|---|---|---|
| < auth-constraint > <role-name>guest</role-name> < /auth-constraint > | < auth-constraint > <role-name>admin</role-name> < auth-constraint > | Both guest , admin |
| < auth-constraint > <role-name>guest</role-name> < /auth-constraint > | < auth-constraint > <role-name>*</role-name> </auth-constraint> | Every Body |
| < auth-constraint /> | < auth-constraint > <role-name>admin</role-name> </auth-constraint> | No Body |
| no < auth-constraint > specified | < auth-constraint > <role-name>admin</role-name> < /auth-constraint > | Every Body can Access |

<u>Note:</u> When 2 different non-empty auth-constraint elements apply to the same constraint resource access is granted to the union of all roles from both of the auth-constraint elements .

## Implementing Authentication:

4 login-config Examples :

web.xml

```
<web-app>
<login-config>
      <auth-method>BASIC</auth-method>
</login-config>
----------------OR----------------------
<login-config>
      <auth-method>DIGEST</auth-method>
</login-config>
----------------OR---------------------
<login-config>
      <auth-method>CLIENT-CERT</auth-method>
</login-config>
----------------OR---------------------
<login-config>
      <auth-method>FORM</auth-method>
      <form-login-config>
            <form-login-page>/login.html</form-login-page>
            <form-error-page>/error.html</form-error-page>
     </form-login-config>
</login-config>
</web-app>
```

---

Except for form once we have declared login-config element in the deployment description(web.xml) , implementing authenticatiuon is done(assume we have already configured userName or password or roles configured at server level).

web.xml

```
<web-app>
<security-constraint>
       <web-resource-collection>
                <web-resource-name>
                <url-pattern>
                <http-method>
                <description>
       </web-resource-collection>
       <auth-constraint>
                <description>
                 <role-name>
      </auth-constraint>
    <user-data-constraint>
             <transport-guarantee>
    </user-data-constraint>
</security-constraint>
<login-config>
```

```
        <auth-method>
        <relian-name>
        <form-login-config>
                <form-login-page>
                <form-error-page>
         </form-login-config>
</login-config>
<security-role>
        <role-name>
</security-role>
</web-app>
```