CORSO DI LAUREA IN INFORMATICA

Tecnologie Software per il Web

SECURITY

a.a. 2020-2021

Introduction

There are two major aspects to securing Web applications:

- **Preventing unauthorized users from accessing sensitive data.**
  This process involves *access restriction* (identifying which resources need protection and who should have access to them) and *authentication* (identifying users to determine if they are one of the authorized ones)
- **Preventing attackers from stealing network data while it is in transit.**
  This process involves the use of Secure Sockets Layer (SSL) to encrypt the traffic between the browser and the server

Access restriction

- The approaches to access restriction are the same regardless of whether or not you use SSL. They are:
- **Declarative security.** With declarative security none of the individual servlets or JSP pages need any security-aware code. Instead, both of the major security aspects are handled by the server (use web.xml)
- **Programmatic security.** With programmatic security, protected servlets and JSP pages at least partially manage their own security:
  - To prevent unauthorized access, each servlet or JSP page must either authenticate the user or verify that the user has been authenticated previously
  - *To safeguard network data, each servlet or JSP page has to check the network protocol used to access it* (request.isSecure()). *If users try to use a regular HTTP connection to access one of these URLs, the servlet or JSP page must manually redirect them to the HTTPS (SSL) equivalent*

Restricting Access to Web Pages

- **Main approach: "*declarative*" security via web.xml/annotation settings**
- **Alternative: programmatic HTTP**

1. Check whether there is Authorization header. If not, go to Step 2. If so, skip over word "basic" and reverse the **base64 encoding** of the remaining part. This results in a string of the form **username:password**. Check the username and password against some stored set. If it matches, return the page. If not, go to Step 2
2. Return a 401 (Unauthorized) response code and a header of the following form:

   WWW-Authenticate: BASIC realm="some-name"

   This instructs browser to pop up a dialog box telling the user to enter a name and password for some-name, then to reconnect with that username and password embedded in a single base64 string inside the Authorization header

Programmatic HTTP Security

- **Idea**
  - Each protected resource authenticates users and decides what (if any) access to grant
- **Advantages**
  - Totally portable
    - No server-specific component
  - Permits custom password-matching strategies
  - No **web.xml** entries needed (except maybe url-pattern)
- **Disadvantages**
  - Much harder to write and maintain
  - Each and every resource has to use the code
    - You can build reusable infrastructure (e.g., servlets that inherit from certain classes or custom JSP tags), but it is still a lot of work

Programmatic HTTP Security (2)

1. **Check whether there is an Authorization request header.**
   - If there is no such header, go to Step 5.
2. **Get the encoded username/password string.**
   - If there is an Authorization header, it should have the following form:
     - Authorization: Basic encodedData
   - Skip over the word Basic and the space—the remaining part is the username and password represented in base64 encoding.
3. **Reverse the base64 encoding of the username/password string.**
   - Use the decodeBuffer method of the BASE64Decoder class. This method call results in a string of the form username:password. The BASE64Decoder class is bundled with the JDK; in JDK 1.3+ it can be found in the sun.misc package in *jdk_install_dir/jre/lib/rt.jar*.

import java.util.Base64;
Base64.getDecoder().decode(…)

Programmatic HTTP Security (3)

4. **Check the username and password.**
   - The most common approach is to use a database or a file to obtain the real usernames and passwords. For simple cases, it is also possible to place the password information directly in the servlet. If the incoming username and password match one of the reference username/password pairs, return the page. If not, go to Step 5. With this approach you can provide your own definition of "match." With container-managed security, you cannot.
5. **When authentication fails, send the appropriate response to the client.**
   - Return a 401 (Unauthorized) response code and a header of the following form:
     WWW-Authenticate: BASIC realm="some-name"
   - This response instructs the browser to pop up a dialog box telling the user to enter a name and password for some-name, then to reconnect with that username and password embedded in a single base64 string inside the Authorization header.

SecretServlet (Registered Name of ProtectedPage Servlet)

...

```
import java.util.Base64;

@WebServlet(name = "/ProtectPage", urlPatterns = { "/protect" }, initParams = {
        @WebInitParam(name = "passwordFile", value = "passwords.properties") })
public class ProtectPage extends HttpServlet {

    private Properties passwords;
    private String passwordFile;

    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        try {
            passwordFile = config.getInitParameter("passwordFile");

            generatePasswords();

            passwords = new Properties();
            passwords.load(new FileInputStream(passwordFile));
        } catch (IOException ioe) {
        }
    }
```

SecretServlet (2)

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String authorization = request.getHeader("Authorization");
    if (authorization == null) {
        askForPassword(response);
    } else {
        String userInfo = authorization.substring(6).trim();
        BASE64Decoder decoder = new BASE64Decoder();
        String nameAndPassword = new String(decoder.decodeBuffer(userInfo));
        int index = nameAndPassword.indexOf(":");
        String user = nameAndPassword.substring(0, index);
        String password = nameAndPassword.substring(index + 1);
        String realPassword = passwords.getProperty(user);
        if ((realPassword != null) && (realPassword.equals(password))) {
            out.println("<html><body>"
            + "<h1>Welcome to the Protected Page</h1>"
            + "Congratulations. You have accessed a protected document.\n"
            + "</body></html>");
        } else {
            askForPassword(response);
        }
    }
}
```

Base64.getDecoder().decode(userInfo)

Si può accedere anche al DB

SecretServlet (3)

Status Code 401

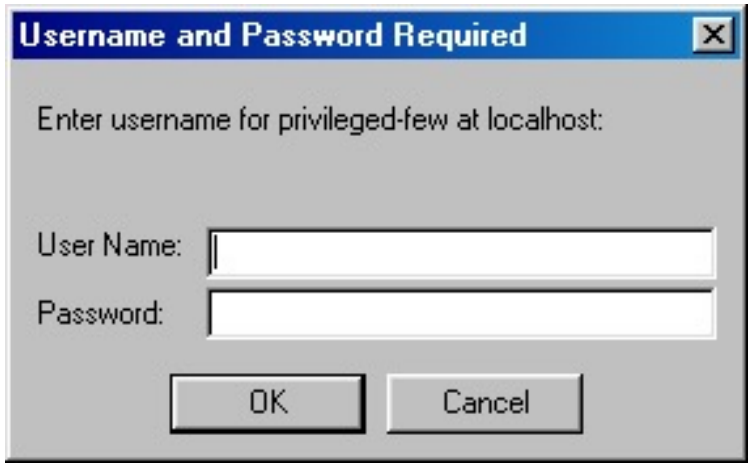
```
// If no Authorization header was supplied in the request.
private void askForPassword(HttpServletResponse response) {
    response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
    response.setHeader("WWW-Authenticate", "BASIC realm=\"privileged-few\"");
}

private void generatePasswords() {
    Properties passwords = new Properties();
    passwords.put("root", "admin");
    passwords.put("tiger", "tiger");
    try {
        FileOutputStream out = new FileOutputStream(passwordFile);
        passwords.store(out, "Passwords");
    } catch (FileNotFoundException e) {
    } catch (IOException ie) {
    }
}
```

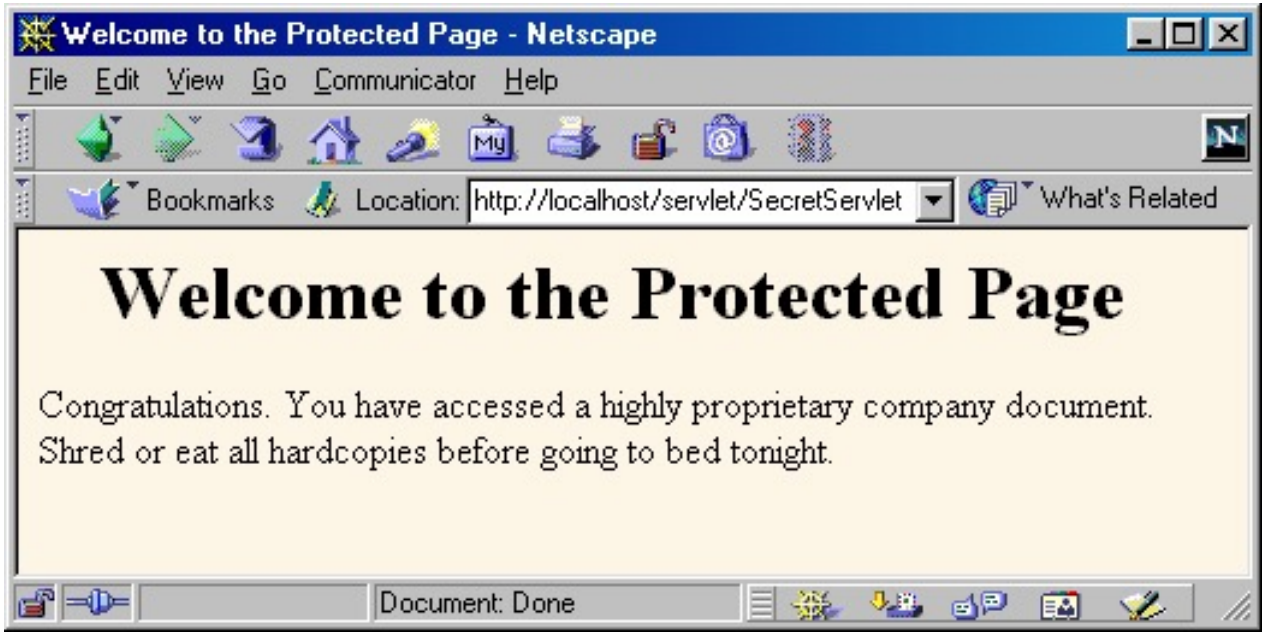
How does it work? (Protection.zip)

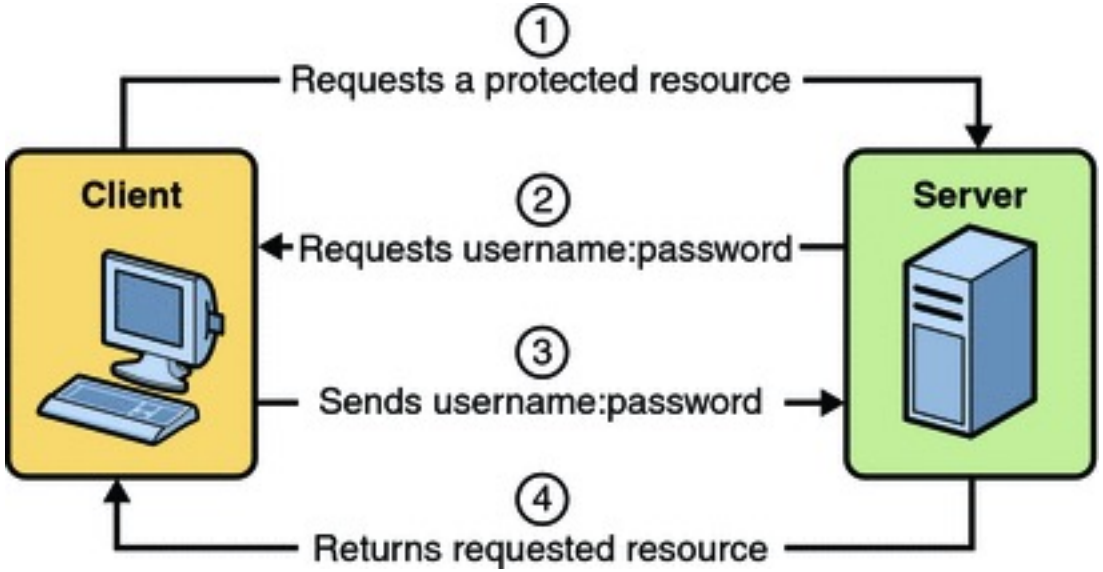
- Run: http://localhost:8080/Protection/protect

- Login: **root** pwd: **admin** (see generatePasswords() in ProtectPage.java)
- or Login: **tiger** pwd: **tiger**

ProtectPage Servlet In Action







BASIC/(custom) FORM-BASED DECLARATIVE AUTHENTICATION

Form-based authentication

- **When a not-yet-authenticated user tries to access a protected resource:**
  - Server automatically redirects user to Web page with an HTML form that asks for username and password
  - Username and password checked against database of usernames, passwords, and roles (user categories)
  - If login successful and role matches, page shown
  - If login unsuccesful, error page shown
  - If login successful but role does not match, 403 error given (but you can use error-page and error-code)
- **When an already authenticated user tries to access a protected resource:**
  - If role matches, page shown
  - If role does not match, 403 error given
  - Session tracking used to tell if user already authenticated

403: Forbidden
(Accesso negato)

Form-based authentication (2)

- **1) Set up usernames, passwords, and roles.**
  - Designate a list of users and associated passwords and abstract role(s) such as normal user or administrator.
  - This is a completely server-specific process.
  - Simplest Tomcat approach: use *install_dir/conf/tomcat-users.xml*:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tomcat-users>
  <user username="john" password="nhoj"
        roles="registered-user" />
  <user username="jane" password="enaj"
        roles="registered-user" />
  <user username="juan" password="nauj"
        roles="administrator" />
  <user username="juana" password="anauj"
        roles="administrator,registered-user" />
</tomcat-users>
```

Form-based authentication (3)

- **2) Tell server that you are using form-based authentication. Designate locations of login and login-failure page.**
  - Use the *web.xml* `login-config` element with `auth-method` of `FORM` and `form-login-config` with locations of pages.

```
<web-app> …
  <login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
      <form-login-page>/login.jsp</form-login-page>
      <form-error-page>/login-error.html</form-error-page>
    </form-login-config>
  </login-config>
  …</web-app>
```

Form-based authentication (4)

- **3) Create a login page (HTML or JSP)**
  - HTML form with `ACTION` of `j_security_check`, `METHOD` of `POST`, textfield named `j_username`, and password field named `j_password`.

```
<FORM ACTION="j_security_check" METHOD="POST">
  …
  <INPUT TYPE="TEXT" NAME="j_username">
  …
  <INPUT TYPE="PASSWORD" NAME="j_password">
  …
</FORM>
```

Do not call the page containing this form directly!

  - For the username, you can use a list box, combo box, or set of radio buttons instead of a textfield.

Form-based authentication (5)

4) Create page for failed login attempts.

- No specific content is mandated.
- Perhaps just "username and password not found" and give a link back to the login page.
- This can be either an HTML or a JSP document.

Invalid user name or password

Please enter a user name or password that is authorized to access this application. For this application, this means a user that has been created in the `file` realm and has been assigned to the *group* of `manager`.

Form-based authentication (6)

5) Specify URLs to be password protected.

- Use `security-constraint` element of *web.xml*. Two subelements: the first (`web-resource-collection`) designates URLs to which access should be restricted; the second (`auth-constraint`) specifies abstract roles that should have access to the given URLs. Using `auth-constraint` with no `role-name` means no *direct* access is allowed.

```
<web-app ...>…
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Sensitive</web-resource-name>
      <url-pattern>/sensitive/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>administrator</role-name>
      <role-name>executive</role-name>
    </auth-constraint>
  </security-constraint>
  <login-config>...</login-config>…
</web-app>
```

Form-based authentication (7)

- **6) List all possible abstract roles (categories of users) that will be granted access to *any* resource**
  - Many servers do not enforce this, but technically required

```
<web-app ...>
  ...
  <security-role>
    <role-name>administrator</role-name>
  </security-role>
  <security-role>
    <role-name>executive</role-name>
  </security-role>
</web-app>
```

Form-based authentication (8)

- **7) Specify which URLs require SSL.**
  - If server supports SSL, you can stipulate that certain resources are available only through encrypted HTTPS (SSL) connections. Use the `user-data-constraint` subelement of `security-constraint`. Only full J2EE servers are *required* to support SSL.

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

Tomcat Invoker Servlet

- The Invoker Servlet allows unconfigured servlets to be called by simply using a /**servlet** URL
- ***It is disabled by default in new Tomcat installations***
- To enable the Invoker Servlet, add the following to your **/WEB-INF/web.xml** file, between the <web-app> tags:

```
<servlet>
        <servlet-name>MyInvoker</servlet-name>
        <servlet-class>
                org.apache.catalina.servlets.InvokerServlet
        </servlet-class>
</servlet>

<servlet-mapping>
        <servlet-name>MyInvoker</servlet-name>
        <url-pattern>/servlet/*</url-pattern>
</servlet-mapping>
```

DefaultServlet

Form-based authentication (9)

- **8) Turn off the invoker servlet.**
  - You protect certain URLs that are associated with registered servlet or JSP names. The *http://host/prefix/servlet/Name* format of default servlet URLs will probably not match the pattern. Thus, the security restrictions are bypassed when the default URLs are used.
  - Disabling it
    - In each Web application, redirect requests to other servlet by normal *web.xml* method

      ```
      <url-pattern>/servlet/*</url-pattern>
      ```

      with a Servlet
    - Globally
      - Server-specific mechanism (e.g. *install_dir/conf/server.xml* for Tomcat).

Form-based authentication (web.xml)

```
<display-name>Protection</display-name>
  <welcome-file-list>
      <welcome-file>loginform.html</welcome-file>
  </welcome-file-list>

  <security-constraint>
      <web-resource-collection>
          <web-resource-name>SecretProtection</web-resource-name>
          <url-pattern>/admin/*</url-pattern>
          <http-method>GET</http-method>
          <http-method>POST</http-method>
      </web-resource-collection>
      <auth-constraint>
          <role-name>manager</role-name>
      </auth-constraint>
  </security-constraint>
  <login-config>
      <auth-method>FORM</auth-method>
      <form-login-config>
          <form-login-page>/loginform.html</form-login-page>
          <form-error-page>/loginerror.html</form-error-page>
      </form-login-config>
  </login-config>
  <security-role>
      <role-name>manager</role-name>
  </security-role>
```

U1

Do not call the page containing a security form directly!

If credentials are valid, it generates

HTTP Status 404 – Not Found

If credentials are valid for a different user, it generates the 403 error

```
<error-page>
        <error-code>403</error-code>
        <location>/loginerror.html</location>
</error-page>
```

Form-based authentication (loginform.html)

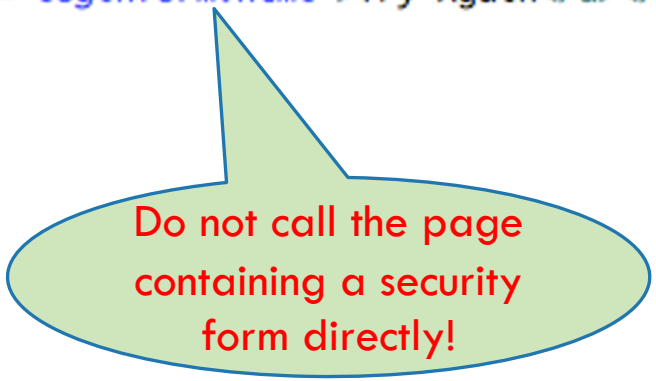
```
...
<form action="j_security_check" method="post">
<fieldset>
      <legend>Login Form</legend>
      <label for="username">Login</label>
      <input id="username" type="text" name="j_username" placeholder="enter login">
      <br>
      <label for="password">Password</label>
      <input id="password" type="password" name="j_password" placeholder="enter password">
      <br>
      <input type="submit" value="Login"/>
      <input type="reset" value="Reset"/>
</fieldset>
</form>
```

- The credentials of rolename "*manager*" are stored in the **tomcat-users.xml** file

```
<?xml version="1.0" encoding="UTF-8"?>
<tomcat-users version="1.0" xmlns="http://tomcat.apache.org/xml"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd">

  <role rolename="tomcat"/>
  <role rolename="manager-gui"/>
  <role rolename="manager"/>
U1 <user username="tomcat" password="tomcat" roles="tomcat, manager-gui, manager"/>

U2 <user username="gui" password="gui" roles="manager-gui"/>
</tomcat-users>
```

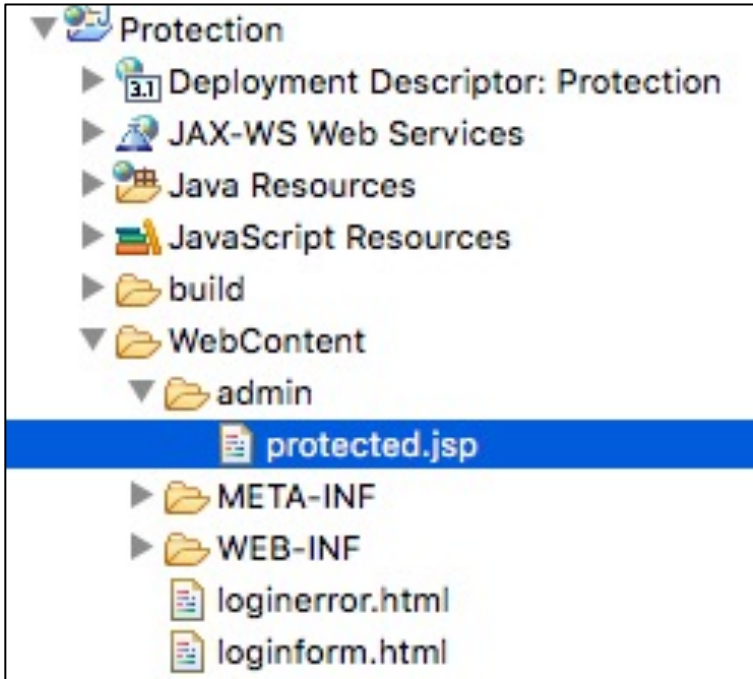
Form-based authentication (loginerror.html)

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Login Error</title>
</head>
<body>
    <h2>Invalid user name or password</h2>

    <p>Please enter a user name or password that is authorized to access this
    application. For this application, this means a user that has been created in the
    <code>file</code> realm and has been assigned to the <em>group</em> of
    <code>manager</code>.  Click here to <a href="loginform.html">Try Again</a></p>
</body>
</html>
```

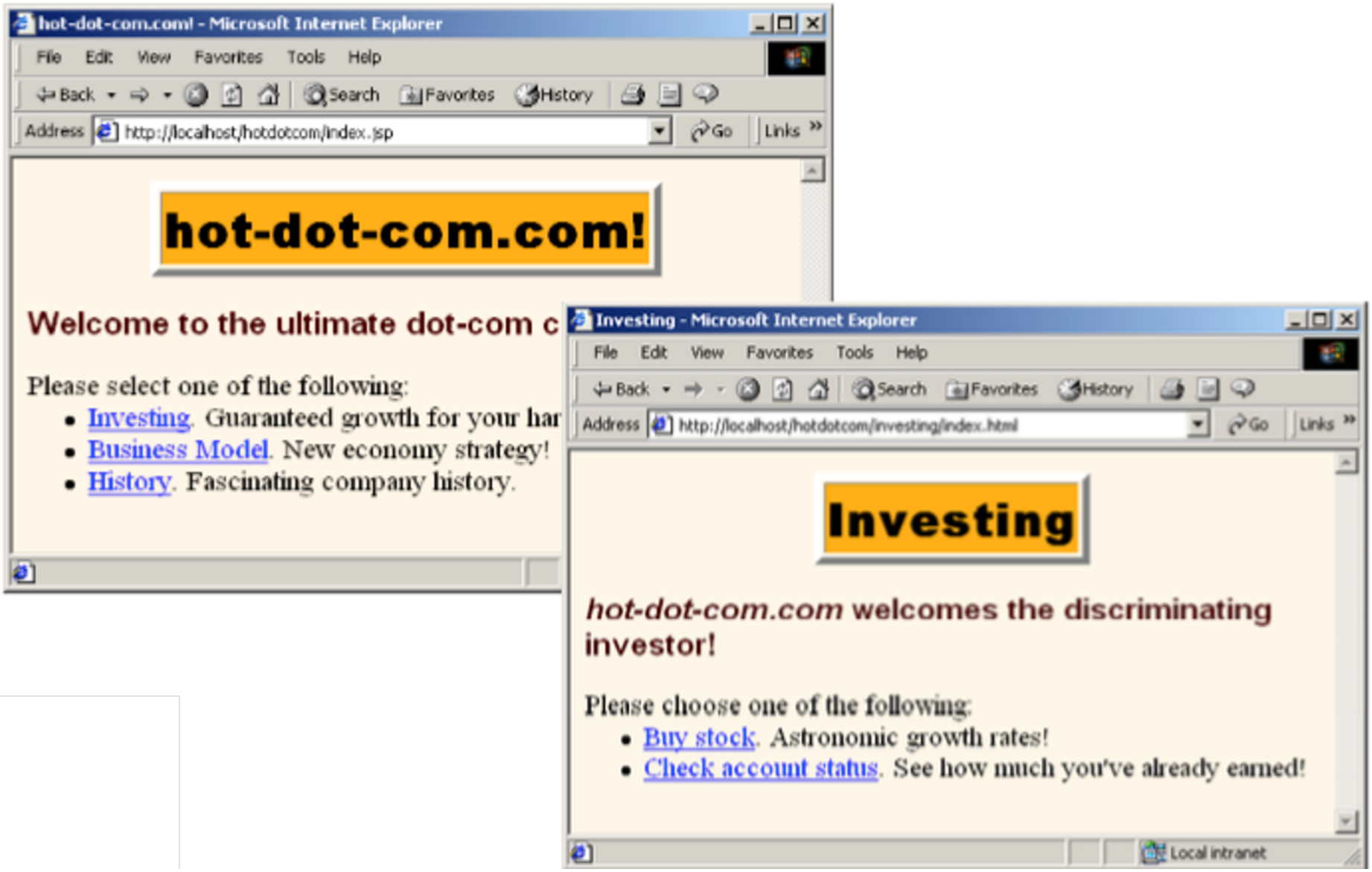


Form-based authentication (/admin/protected.jsp)

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Protected Page</title>
</head>
<body>
<h1>Welcome to the Protected Page</h1>
Congratulations. You have accessed a protected document.
<br>
</body>
</html>
```



Example: Form-based security



Example: Step 1

- **Set up usernames, passwords, and roles.**
  - *install_dir/conf/tomcat-users.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tomcat-users>
  <user username="john" password="nhoj"
        roles="registered-user" />
  <user username="jane" password="enaj"
        roles="registered-user" />
  <user username="juan" password="nauj"
        roles="administrator" />
  <user username="juana" password="anauj"
        roles="administrator,registered-user" />
  <!-- … -->
</tomcat-users>
```

Example: Step 2

- **Tell server that you are using form-based authentication. Designate locations of login and login-failure page.**

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

Example: Step 3

- **Create a login page**

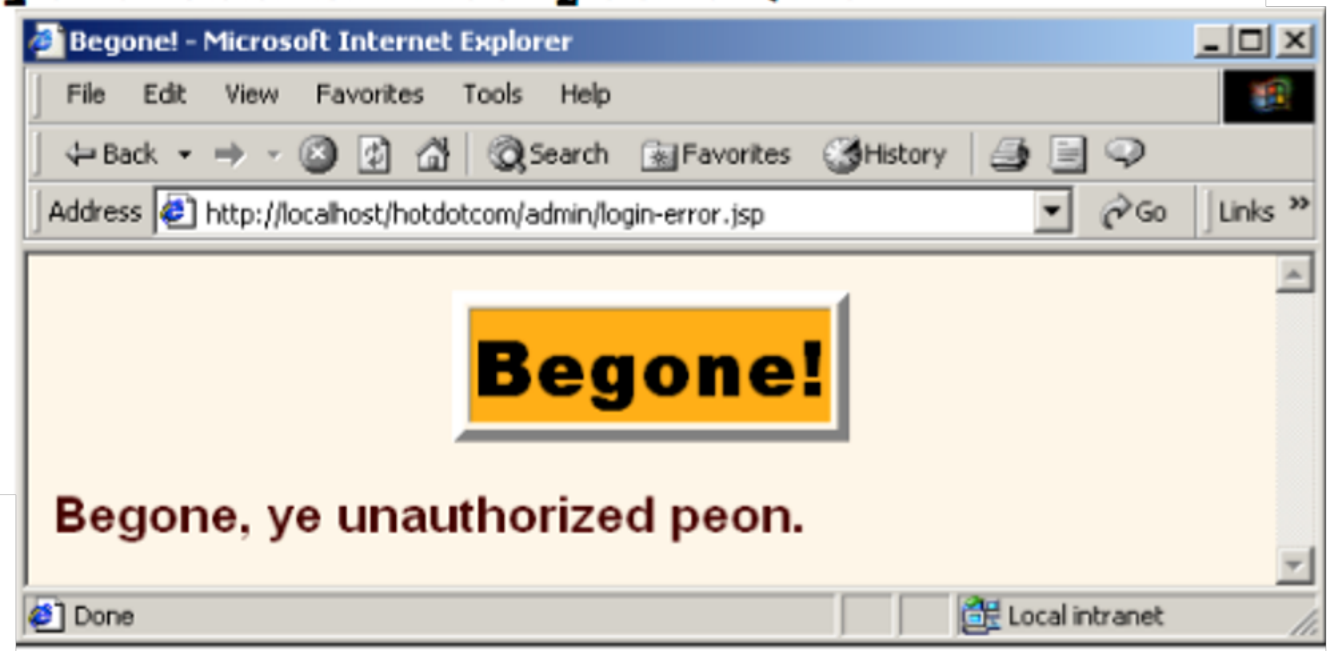
```
…
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">Log In</TABLE>
<P>
<H3>Sorry, you must log in before
accessing this resource.</H3>
<FORM ACTION="j_security_check" METHOD="POST">
<TABLE>
<TR><TD>User name:
        <INPUT TYPE="TEXT" NAME="j_username">
<TR><TD>Password:
        <INPUT TYPE="PASSWORD" NAME="j_password">
<TR><TH><INPUT TYPE="SUBMIT" VALUE="Log In">
</TABLE>
</FORM></BODY></HTML>
```

Example: Step 3 (result)

Example: Step 4

- **Create page for failed login attempts.**

```
…
<BODY>
<TABLE BORDER=5 ALIGN="CENTER">
  <TR><TH CLASS="TITLE">Begone!</TABLE>

<H3>Begone, ye unauthorized peon.</H3>

</BODY>
</HTML>
```



Example: access rules

- **Home page**
  - Anyone
- **Investing page**
  - Registered users
  - Administrators
- **Stock purchase page**
  - Registered users
  - Via SSL only
- **Delete account page**
  - Administrators

Example: Step 5

- **Specify URLs to be password protected.**

```
<!-- Protect everything within
     the "investing" directory. -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Investing
    </web-resource-name>
    <url-pattern>/investing/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>registered-user</role-name>
    <role-name>administrator</role-name>
  </auth-constraint>
</security-constraint>
```

Example: Step 5 (continued)

```
<!-- Only users in the administrator role
     can access the delete-account.jsp page
     within the admin directory. -->

<security-constraint>
  <web-resource-collection>
    <web-resource-name>Account Deletion
    </web-resource-name>
    <url-pattern>/admin/delete-account.jsp
    </url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>administrator</role-name>
  </auth-constraint>
</security-constraint>
```

Example: Step 5 (results)

- **First attempt to access account status page**

- **Result of successful login and later attempts to access account status page**

Example: Step 6

- **6) List all possible abstract roles (types of users) that will be granted access to *any* resource**

```
<web-app ...>
  ...
  <security-role>
    <role-name>registered-user</role-name>
  </security-role>
  <security-role>
    <role-name>administrator</role-name>
  </security-role>
</web-app>
```

Example: Step 7

- **Specify which URLs require SSL.**

```
<!-- URLs of the form
       http://host/webAppPrefix/ssl/blah
       require SSL and are thus redirected to
       https://host/webAppPrefix/ssl/blah. -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Purchase
    </web-resource-name>
    <url-pattern>/ssl/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>registered-user</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL
    </transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Example: Step 7 (results)

- **http://host/prefix/ssl/buy-stock.jsp *or* https://host/prefix/ssl/buy-stock.jsp**

Example: Step 8

- **Turn off the invoker servlet**

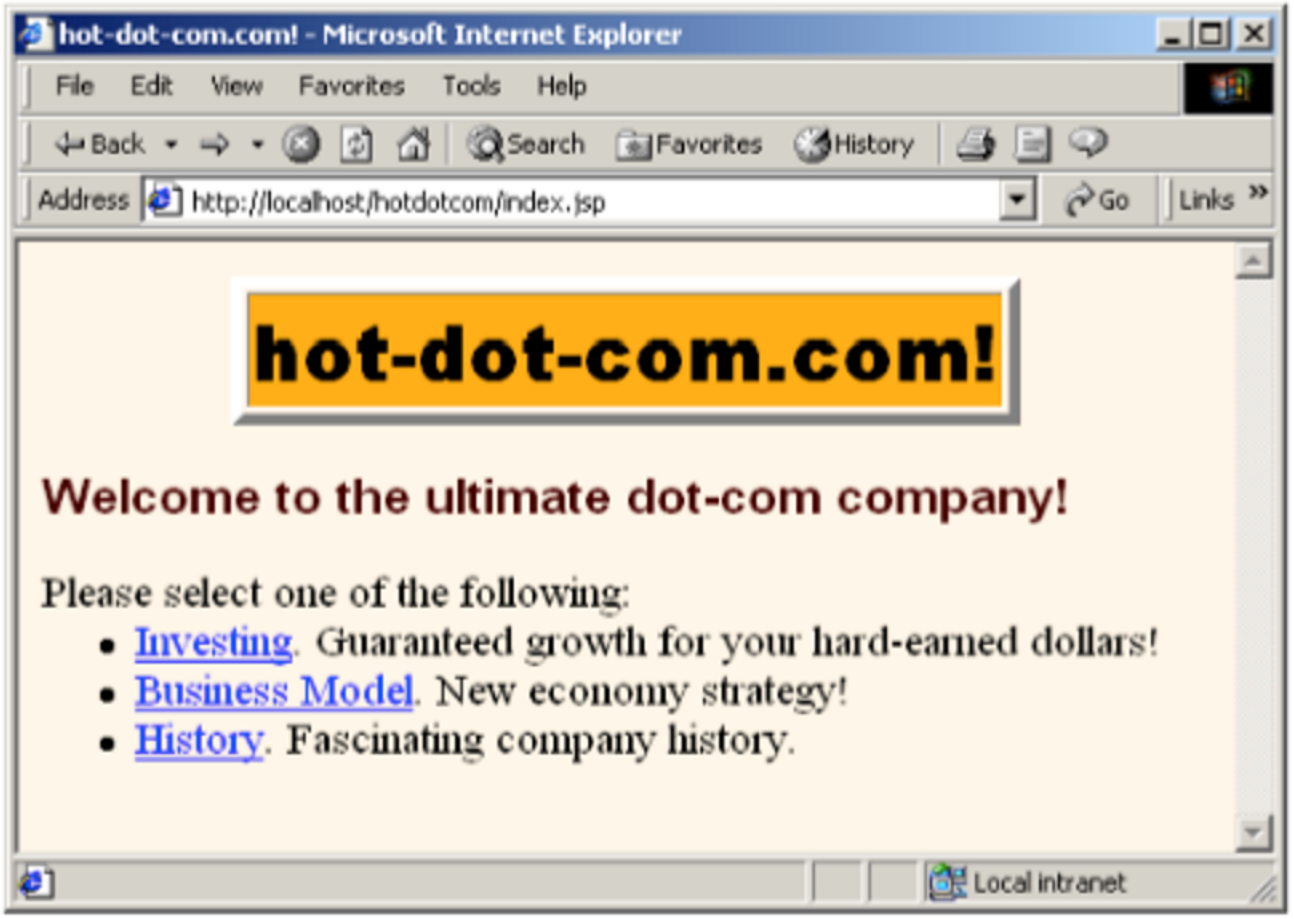
```
<!-- Turn off invoker. Send requests
     to index.jsp. -->

<servlet-mapping>
  <servlet-name>Redirector</servlet-name>
  <url-pattern>/servlet/*</url-pattern>
</servlet-mapping>
…
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
  <welcome-file>index.html</welcome-file>
</welcome-file-list>
```

Example: Step 8 (continued)

```
/** Servlet that simply redirects users to the
 *  Web application home page.
 */

public class RedirectorServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.sendRedirect(request.getContextPath());
  }

  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
    doGet(request, response);
  }
}
```

Example: Step 8 (results)

- **Attempt to access http://host/hotdotcom/servlet/Anything**



Basic authentication

- **When a not-yet-authenticated user tries to access a protected resource:**
  - Server sends a 401 status code to browser
  - Browser pops up dialog box asking for username and password, and they are sent with request in Authorization request header
  - Username and password checked against database of usernames, passwords, and roles (user categories)
  - If login successful and role matches, page shown
  - If login unsuccesful or role does not match, 401 again
- **When an already authenticated user tries to access a protected resource:**
  - If role matches, page shown
  - If role does not match, 403 error given
  - Request header used to tell if user already authenticated

401: Authorization Required

403: Forbidden (Accesso negato)

Basic authentication

1. **Set up usernames, passwords, and roles.**
   - Same as for form-based authentication. Server-specific.
2. **Tell the server that you are using BASIC authentication. Designate the realm name.**
   - Use the *web.xml* `login-config` element with an `auth-method` subelement of `BASIC` and a `realm-name` subelement (generally used as part of the title of the dialog box that the browser opens).

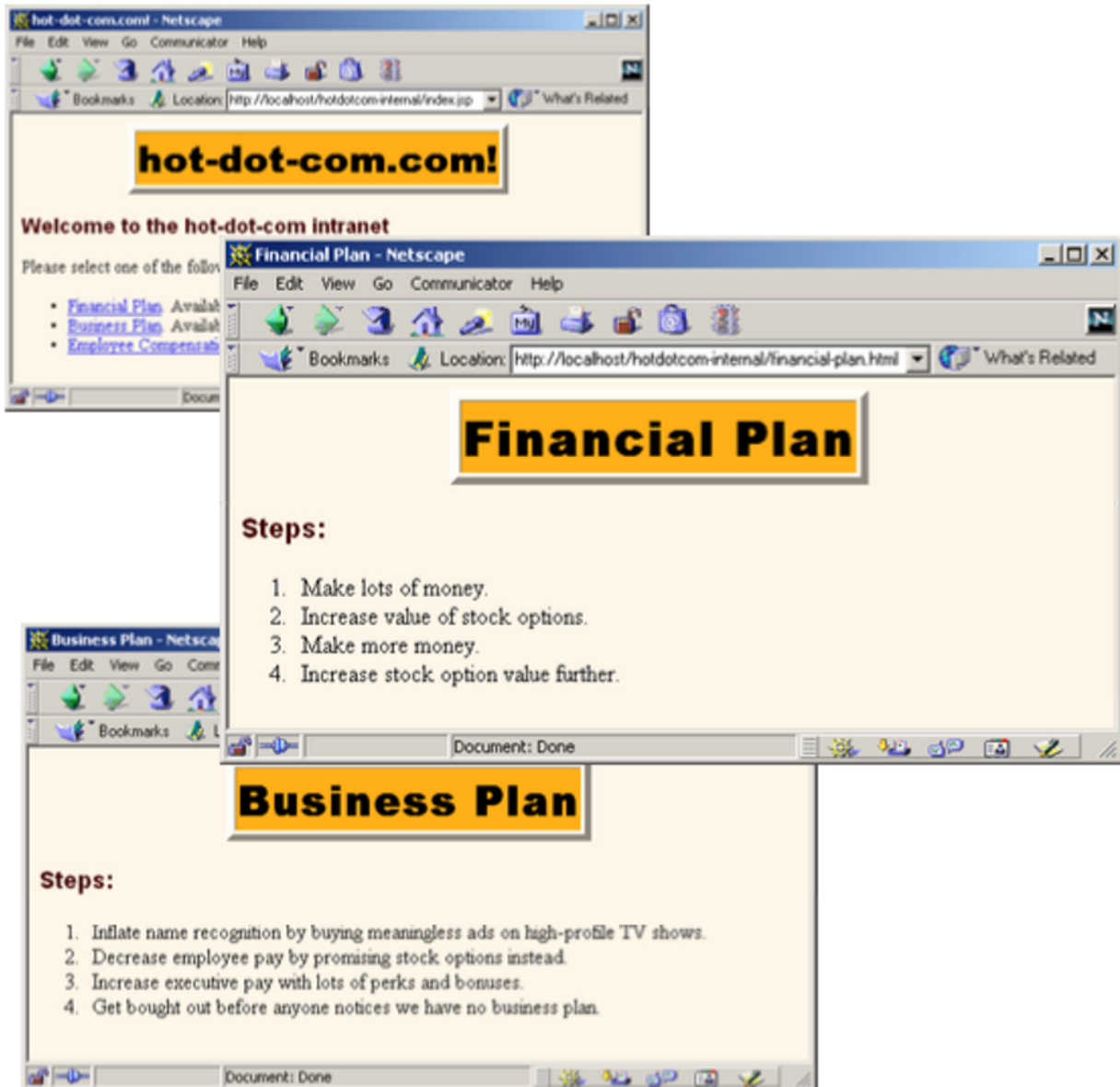
```
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Some Name</realm-name>
</login-config>
```

Basic authentication (continued)

3. **Specify which URLs should be password protected.**
   - Same as with form-based authentication.
4. **List all possible roles (categories of users) that will access any protected resource**
   - Same as with form-based authentication
5. **Specify which URLs should be available only with SSL.**
   - Same as with form-based authentication.
6. **Turn off the invoker servlet.**
   - Same as with form-based authentication.

Example: Basic authentication

- **Home page**
  - Anyone
- **Financial plan**
  - Employees or executives
- **Business plan**
  - Executives only



Example: Step 1

- **Set up usernames, passwords, and roles.**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tomcat-users>
  …
  <user username="gates" password="llib"
        roles="employee" />
  <user username="ellison" password="yrral"
        roles="employee" />
  <user username="mcnealy" password="ttocs"
        roles="executive" />
</tomcat-users>
```

Example: Step 2

- **Tell the server that you are using BASIC authentication. Designate the realm name.**

```
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Intranet</realm-name>
</login-config>
```

Example: Step 3

- **Specify which URLs should be password protected.**

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>
      Financial Plan
    </web-resource-name>
    <url-pattern>
      /financial-plan.html
    </url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>employee</role-name>
    <role-name>executive</role-name>
  </auth-constraint>
</security-constraint>
```

Example: Step 3 (continued)

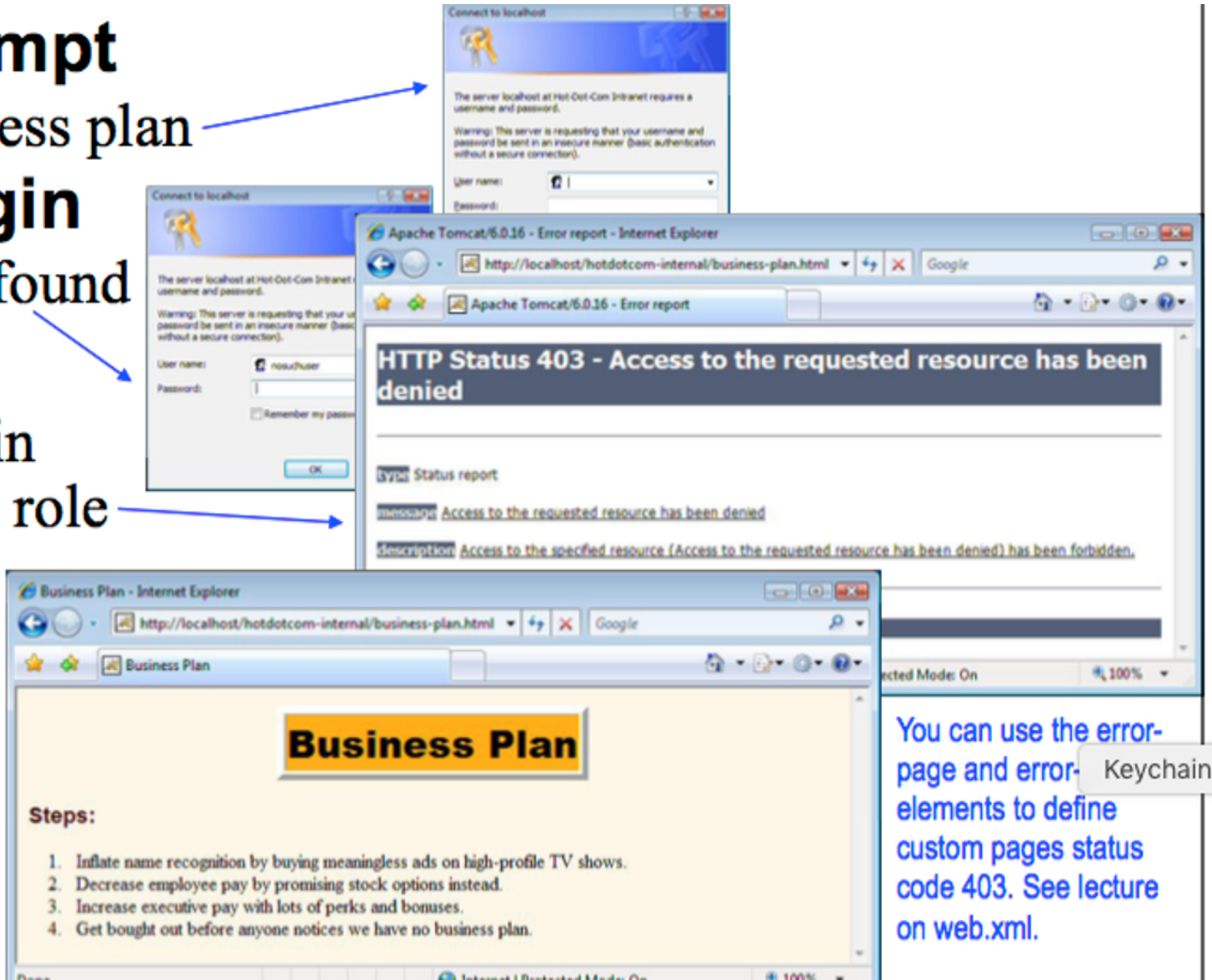
```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>
      Business Plan
    </web-resource-name>
    <url-pattern>
      /business-plan.html
    </url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>executive</role-name>
  </auth-constraint>
</security-constraint>
```

Example: Step 4

```
<web-app ...>
   ...
   <security-role>
      <role-name>employee</role-name>
   </security-role>
   <security-role>
      <role-name>executive</role-name>
   </security-role>
</web-app>
```

Example: Basic Authentication (results)

- **First attempt**
  - For business plan
- **Failed login**
  - User not found
- **Denied**
  - User not in executive role
- **Success**
  - User in executive role

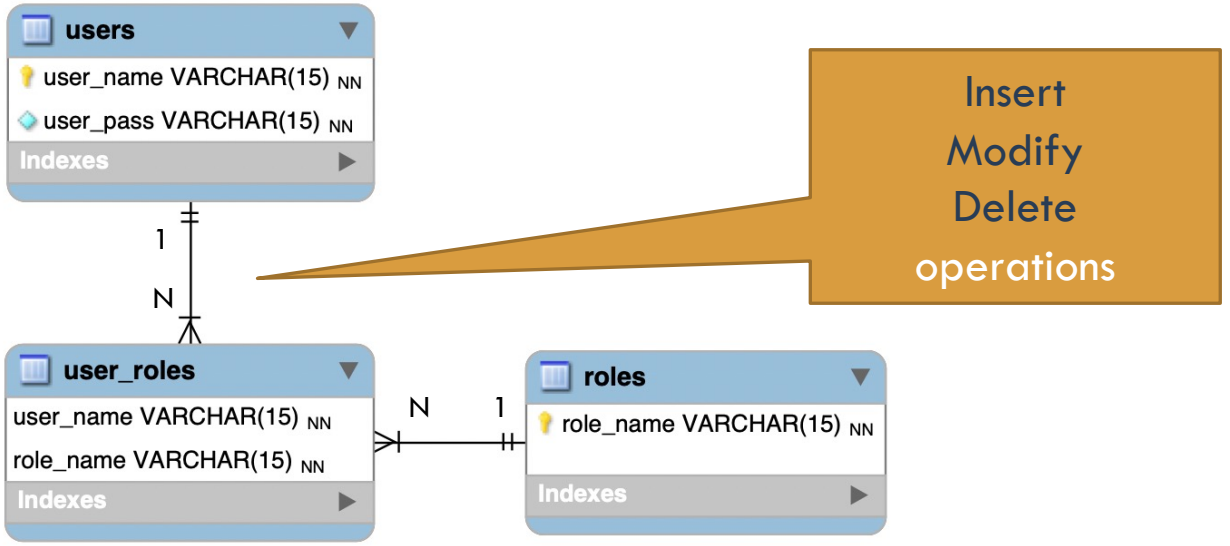


Form-Based vs. Basic authentication

- **Advantages of form-based**
  - Consistent look and feel
  - Fits model users expect from ecommerce sites
- **Disadvantage of form-based**
  - Can fail if server is using URL rewriting for session tracking. Can fail if browser has cookies disabled.
- **Advantages of BASIC**
  - Doesn't rely on session tracking
  - Easier when you are doing it yourself (programmatic)
- **Disadvantage of BASIC**
  - Small popup dialog box seems less familiar to most users

JDBC Realm (UserRealm.zip)

- Is an implementation of a Tomcat Realm that use a set of configurable tables inside a MySQL to store user's data
  - this tables are accessed by means of standard JDBC drivers (MySQL JDBC driver)
- It avoids the use of credentials stored in the **tomcat-users.xml** file
- All the parameters, drivers, tables, and columns are user configurable



The database ‘authority’

```
DROP DATABASE IF EXISTS authority;
CREATE DATABASE authority;

USE authority;

DROP USER IF EXISTS 'authority'@'localhost';
CREATE USER 'authority'@'localhost' IDENTIFIED BY 'authority';
GRANT ALL ON authority.* TO 'authority'@'localhost';

DROP TABLE IF EXISTS user_roles;
DROP TABLE IF EXISTS users;
DROP TABLE IF EXISTS roles;

CREATE TABLE roles (
  role_name VARCHAR(15) NOT NULL PRIMARY KEY
);

INSERT INTO roles(role_name) VALUES ('administrator');
INSERT INTO roles(role_name) VALUES ('registered-user');
INSERT INTO roles(role_name) VALUES ('manager');

CREATE TABLE users (
  user_name VARCHAR(15) NOT NULL PRIMARY KEY,
  user_pass VARCHAR(15) NOT NULL
);

CREATE TABLE  user_roles (
  user_name VARCHAR(15) NOT NULL,
  role_name VARCHAR(15) NOT NULL,
  PRIMARY KEY( user_name, role_name ),
  FOREIGN KEY (user_name) REFERENCES users(user_name),
  FOREIGN KEY (role_name) REFERENCES roles(role_name)
);

INSERT INTO users(user_name, user_pass) VALUES ('man','man');
INSERT INTO user_roles(user_name, role_name) VALUES ('man', 'manager');
```

Setup Tomcat

Server.xml in Tomcat

```
<Realm className="org.apache.catalina.realm.JDBCRealm"
    debug="99"
    driverName="com.mysql.cj.jdbc.Driver"
    connectionURL="jdbc:mysql://localhost:3306/authority?
            useUnicode=true&amp;characterEncoding=UTF-8&amp;
            serverTimezone=UTC&amp;verifyServerCertificate=false&amp;
            useSSL=true&amp;requireSSL=true&amp;autoReconnect=true"
    connectionName="authority"
    connectionPassword="authority"
    userTable="users"
    userNameCol="user_name"
    userCredCol="user_pass"
    userRoleTable="user_roles"
    roleNameCol="role_name">
</Realm>
```

Attribute	Meaning
driverName	The name of the driver needed to connect to the database
connectionURL	The connection URL used to connect to the database (with parameters)
userTable	The user's tables
userNameCol	The column in the user's table that contains the name
userCredCol	The column in the user's table that contains the password
userRoleTable	The user's roles table
roleNameCol	The column in the user's table that contains a role given to a user
connectionName	The name to use when connecting to the database
connectionPassword	The password to use when connecting to the database

- Place a copy of the JDBC driver inside the **$CATALINA_HOME/lib** directory

Additional notes

- Administering the information in the *users* and *user roles* table is the responsibility of your own applications
- When a user attempts to access a protected resource for the first time, Tomcat will call the authenticate() method of this Realm
  - Any changes you have made to the database directly (new users, changed passwords or roles, etc.) will be immediately reflected
- Once a user has been authenticated, the user (and his or her associated roles) are cached within Tomcat for the duration of the user's login
  - For FORM-based authentication until the session times out or is invalidated

```
session.invalidate();
```

  - For BASIC authentication until the user closes their browser
- The cached user is **not** saved and restored across sessions serialisations
  - Any changes to the database information for an already authenticated user will **not** be reflected until the next time that user logs on again

Problems

- **Problems with declarative security**
  - The advantages of declarative security usually outweigh the disadvantages. But not always.
- **Combination security: mixing server-managed and servlet-managed (programmatic) security**
  - Solve one of the drawbacks of declarative security with only a little bit of extra work.
- **Pure programmatic security**
  - Solve the other drawbacks, but with a very lot of extra work.

SESSION AND TOKEN

MANUAL APPROACH

Example 1: Token in session (page login-form.jsp)

...

```
<form action="Login" method="post">
<fieldset>
     <legend>Login</legend>
     <label for="username">Login</label>
     <input id="username" type="text" name="username" placeholder="enter login">
     <br>
     <label for="password">Password</label>
     <input id="password" type="password" name="password" placeholder="enter password">
     <br>
     <input type="submit" value="Login"/>
     <input type="reset" value="Reset"/>
</fieldset>
</form>
```

Token in session (page protected.jsp)

```
<%
// Check user credentials
Boolean adminRoles = (Boolean) session.getAttribute("adminRoles");
if ((adminRoles == null) || (!adminRoles.booleanValue()))
{
    response.sendRedirect("./login-form.jsp");
    return;
}
%>

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Protected Page</title>
</head>
<body>
<h1>Welcome to the Protected Page</h1>
Congratulations. You have accessed a protected document.
<br><br>
<form action="Logout" method="get" >
     <input type="submit" value="Logout"/>
</form>
</body>
</html>
```

Token in session (servlet Login)

```
@WebServlet("/Login")
public class Login extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        {
            String username = request.getParameter("username");
            String password = request.getParameter("password");
            String redirectedPage;
            try {
                checkLogin(username, password);
                request.getSession().setAttribute("adminRoles", new Boolean(true));
                redirectedPage = "/protected.jsp";
            } catch (Exception e) {
                request.getSession().setAttribute("adminRoles", new Boolean(false));
                redirectedPage = "/login-form.jsp";
            }
            response.sendRedirect(request.getContextPath() + redirectedPage);
        }
    }

    private void checkLogin(String username, String password) throws Exception {
        if ("root".equals(username) && "admin".equals(password)) {
            //
        } else
            throw new Exception("Invalid login and password");
    }
```

Si può accedere anche al DB

Token in session (servlet Logout)

```
@WebServlet("/Logout")
public class Logout extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        request.getSession().removeAttribute("adminRoles");
        request.getSession().invalidate();

        String redirectedPage = "/login-form.jsp";
        response.sendRedirect(request.getContextPath() + redirectedPage);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doGet(request, response);
    }
```

Si può fare anche il redirect alla Home page

- Run: http://localhost:8080/Protection/protected.jsp (login and logout)
- ***Filter can be used to control the access to resources***
  - Run: http://localhost:8080/Protection/baseFilter.html

Penetration Testing

https://pentest-tools.com/home

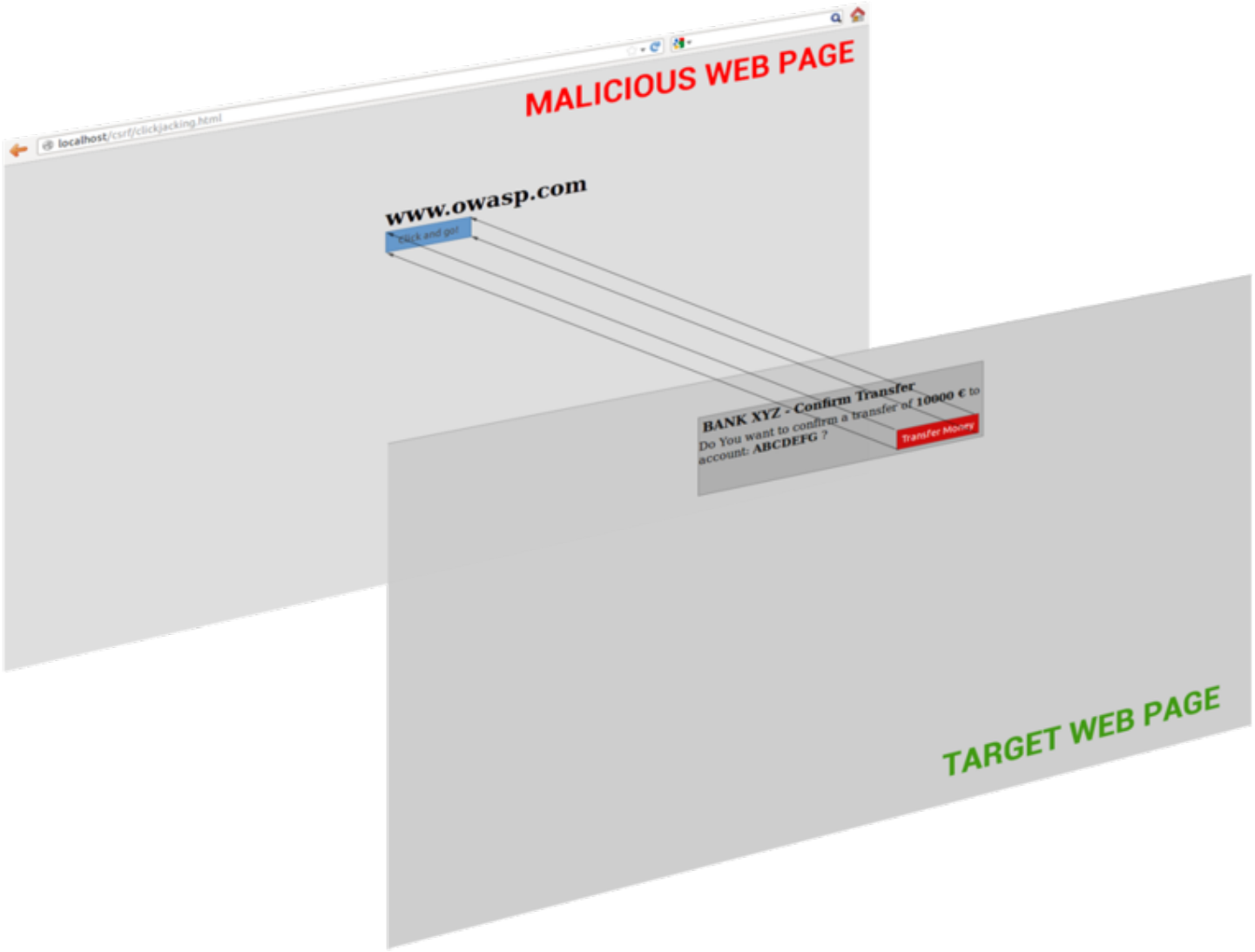
- A **penetration test** is an authorized simulated cyber attack on a computer system, performed to evaluate the security of the system
- The test is performed to identify both weaknesses (also referred to as vulnerabilities), including the potential for unauthorized parties to gain access to the system's features and data, as well as strengths, enabling a full risk assessment to be completed
- The process typically identifies the target systems and a particular goal, then reviews available information and undertakes various means to attain that goal. A penetration test target may be a **white box** (which provides background and system information) or **black box** (which provides only basic or no information except the company name)
- A penetration test can help determine whether a system is vulnerable to attack if the defenses were sufficient, and which defenses (if any) the test defeated
- Security issues that the penetration test uncovers should be reported to the system owner
- Penetration test reports may also assess potential impacts to the organization and suggest countermeasures to reduce risk

Penetration Testing

- Major type of computer security vulnerability typically found in web applications:
  - **Cross-site scripting (XSS) + Clickjacking + Missing X-Frame-Options**

- **XSS:** enables attackers to inject client-side scripts into web pages viewed by other users
- **Clickjacking:** is a malicious technique of tricking a user into clicking on something different from what the user perceives, thus potentially revealing confidential information or allowing others to take control of their computer while clicking on seemingly innocuous objects, including web pages
- **Missing X-Frame-Options:** The X-Frame-Options HTTP header field indicates a policy that specifies whether the browser should render the transmitted resource within a **frame** or an **iframe** (missing these options enables the risk of a clickjacking attack)

Clickjacking



Handling XSS + X-frame + clickjacking

Web.xml

```
<filter>
  <filter-name>httpHeaderSecurity</filter-name>
  <filter-class>org.apache.catalina.filters.HttpHeaderSecurityFilter</filter-class>
  <init-param><param-name>xssProtectionEnabled</param-name><param-value>true</param-value>
   </init-param>
  <init-param><param-name>antiClickJackingEnabled</param-name><param-value>true</param-value>
  </init-param>
   <init-param><param-name>antiClickJackingOption</param-name><param-value>DENY</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>httpHeaderSecurity</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

XSS

X-Frame-Options

Clickjacking

SSL/TLS (HTTPS) in Tomcat

Server.xml in Tomcat

<JAVA_HOME>/bin/keytool

- In a command shell, create the keystore:

  keytool -genkey -alias tomcat -keypass ***myPassword*** -keystore keystore.jks -storepass ***myPassword*** -keyalg RSA -validity 360 -keysize 2048

  - Follow through the prompts and fill in the information
  - It should save a keystore.jks file to your current directory

- To get it to work in Eclipse:

  or in the main directory of Tomcat

  1. Move the keystore.jks into the directory:

     <workspace>/.metadata/.plugins/org.eclipse.wst.server.core/tmp0

  2. Add (or uncomment) the **Connector** to the **server.xml** configuration file:

```
<Connector SSLEnabled="true" acceptCount="100" clientAuth="false"
debug="0" disableUploadTimeout="true" enableLookups="true"
keystoreFile="keystore.jks" keystorePass="myPassword"
maxSpareThreads="75" maxThreads="150" minSpareThreads="25"
port="8443" scheme="https" secure="true" sslProtocol="TLSv1" />
```

  443

  TLS

  3. Start Tomcat

- In the browser: **https://localhost:8443/...**

Cookie: Secure + HTTPS

Web.xml

- Using HttpOnly in Set-Cookie helps in mitigating the most common risk of **XSS attack**

```
<session-config>
        <cookie-config>
                <http-only>true</http-only>
                <secure>true</secure>
        </cookie-config>
</session-config>
```

Server Name, Shutdown port and Error reports

Server.xml in Tomcat

<Connector port="8080" ... server="Apache" />

<Server port="8005" shutdown="ReallyComplexWord" >

- In the Host section:

<Valve className="org.apache.catalina.valves.ErrorReportValve"
showReport="false" showServerInfo="false" />