CSE 510
Web Data Engineering

Access Control

Authentication & Authorization

cse@buffalo

Access Control Mechanisms

- Declarative Authorization using Realms
  - The expression of app security external to the app
  - Separate from your JSP and Java code
  - Based on specifying centralized policy
  - Based on **static roles** who are groups of users that have access to particular resources (typically pages)
  - Configured in web.xml
- Programmatic
  - Your code is responsible
  - Choose when you need to create intricate access control strategies

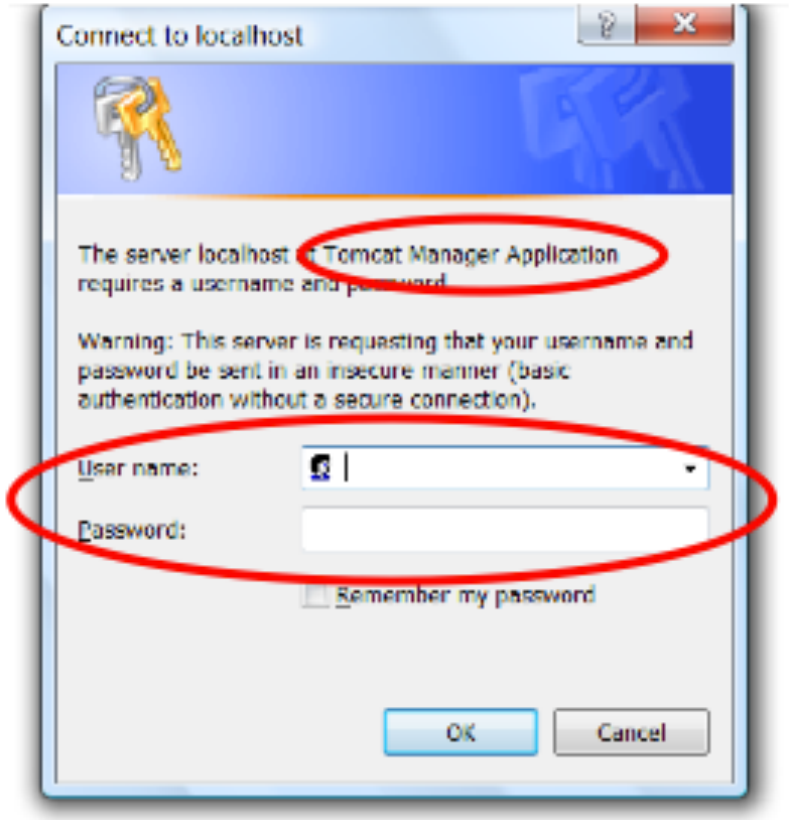
Declarative Authorization Using Realms

- PLUS: Really simple!
- MINUS: Static policy (very rarely a problem)
- Memory, JDBC, DataSource and JNDI Realms are "ready out of the box"
- Memory Realm
  - Users' info is static
  - Clear text passwords
  - Define in <TOMCAT_HOME>/conf/tomcat-user.xml
- JDBC Realm
  - Users' info is stored in DB (preferred)
- Authentication Method
  - BASIC, DIGEST, FORM

Authentication Method – 1: BASIC

Usage:

- Pop up a dialog box
- Browser-based authentication
- User & Password are sent in every HTTP request
- **Must exit the browser to logout**



Authentication Method – 2: DIGEST

Usage:

- Same as BASIC
- Username and password are encrypted into a message digest value

Authentication Method – 3: FORM

Usage:

- Define your own login and error page
- Authentication is defined in servlet session
- **Logout by session.invalidate()**

Authentication Method – 4: Client

Usage

- Implemented with **SSL (Secure Sockets Layer)**
- Requires the client to possess a public key certificate
- Most secure, but costly

Memory Realm Example

- Using **tomcat-users.xml** file
- Two classes of users: student, admin
- All **`http://host/app/admins/*`** pages will be accessed only by administrators
- All **`http://host/app/students/*`** pages will be accessed by students and administrators
- "john" is a student
- "ted" is a student
- "yvette" is an administrator

Security Constraints

web.xml

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Students Area</web-resource-name>
    <!-- Define the context-relative URL(s) to protect -->
    <url-pattern>/students/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>student</role-name>
    <role-name>admin</role-name>
  </auth-constraint>
</security-constraint>
```

Security Constraints (cont'd)

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Admin Area</web-resource-name>
    <!-- Define the context-relative URL(s) to protect -->
    <url-pattern>/admins/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>admin</role-name>
  </auth-constraint>
</security-constraint>
```

tomcat-users.xml

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="student"/>
  <role rolename="admin"/>
  <user username="john" password="john" roles="student"/>
  <user username="ted" password="ted" roles="student"/>
  <user username="yvette" password="yvette" roles="admin"/>
</tomcat-users>
```

Login Configuration

web.xml

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

login.jsp

```
<form method="POST" action="j_security_check">
  Username:
  <input size="12" name="j_username" type="text"/><br />
  Password:
  <input size="12" name="j_password" type="password"/><br />
  <input type="submit" value="Login"/>
</form>
```

Access Authentication Info

- getRemoteUser()
- getAuthType()
- isUserInRole()
- getUserPrincipal()
  - Principal is an object to identify user

```
User Principal: <%= request.getUserPrincipal().getName() %>
Username: <%= request.getRemoteUser() %>
Authenticatin Method: <%= request.getAuthType() %>
<% if(request.isUserInRole("admin")) { %>
   You are in <i>admin</i> role<br/>
<% } %>
```

Declarative Authorization

- Accessing protected pages is the **only** way to invoke the login page
- If you try to access protected page A:
  - Login page will pop up
  - After you login successfully, you will be directed to page A
- However, if you go to login page directly, after you login, which page you are directed to?
  - Tomcat doesn't know and there is no way to specify!

Dynamic DB-Driven Access Control

- **`tomcat-users.xml`** is a kind of **Security Realm**, that is, a provider of user credentials
- **`JDBCRealm`**: User credentials are stored in a relational database, accessed via JDBC
- **`DataSourceRealm`**: User credentials are stored in a JNDI named JDBC DataSource
  - no need to specify connection details again
- **`JNDIRealm`**: User credentials are stored in a directory server, accessed via JNDI

DataSourceRealm

META-INF/config.xml

```
<Realm className="org.apache.catalina.realm.DataSourceRealm"
       debug="99"
       dataSourceName="jdbc/ClassesDBPool"
       localDataSource="true"
       userTable="users"
       userNameCol="username"
       userCredCol="password"
       userRoleTable="userroles"
       roleNameCol="role"
       digest="MD5"/>
```

users

username	password
john	john
ted	ted
yvette	yvette

userroles

username	role
john	student
ted	student
yvette	admin

Scope of Realm

- If you place declaration in context.xml, that is, at **Context Level**, then realm applies only to the enclosing app
- If you place declaration in server.xml, at **Engine Level**, then realm applies to all apps

Hiding Passwords

// Assume `pwd` has password, `user` has user name and
// `con` is connection to database of DataSourceRealm used for security

```
String encMD5Pwd =
      org.apache.catalina.realm.RealmBase.Digest(pwd, "MD5");
```

// returns MD5 encoding, which you insert in DB

```
PreparedStatement makeNewUser = con.prepareStatement(
 "INSERT INTO users(username, password) VALUES(?, ?)" );
makeNewUser.setString(1, user);
makeNewUser.setString(2, encMD5Pwd);
makeNewUser.execute();
```

Hiding Passwords - Alternative

// Assume `pwd` has password, `user` has user name and `con` is a
// connection to a **MySQL** DB of DataSourceRealm used for security

// use MySQL's **MD5 function**

```
PreparedStatement makeNewUser = con.prepareStatement(
"INSERT INTO users(username, password) VALUES (?, MD5(?))" );
makeNewUser.setString(1, user);
makeNewUser.setString(2, pwd);
makeNewUser.execute();
```

Enabling Secure Sockets Layers (SSL)

1. Generate Certificate
   - Web server's assurance to the web client
2. Configure Tomcat
3. Configure Web Application

Generate Certificate

- Create a certificate **keystore** by executing the following command:

- Windows:

  %JAVA_HOME%\bin\keytool -genkey -alias tomcat -keyalg RSA

- Unix:

  $JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA

- This command will create a new file, in the home directory of the user under which you run it, named **.keystore**

Configure Tomcat

- Uncomment the SSL HTTP/1.1 Connector entry in <TOMCAT_HOME>/conf/**server.xml**

```
<Connector port="8443" protocol="HTTP/1.1"
           SSLEnabled="true" maxThreads="150”
           scheme="https" secure="true”
           keystoreFile="${user.home}/.keystore”
           keystorePass="changeit"
           clientAuth="false" sslProtocol="TLS" />
```

Configure Web Application

web.xml

```
<!-- Force SSL on all application pages -->
<security-constraint>

  <web-resource-collection>
    <web-resource-name>Entire Application</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>

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

</security-constraint>
```

Enabling SSL

- Try accessing:

  `https://localhost:8443/`

- Since your certificate is not *verified*, you should get a message similar to:

  The certificate is not trusted because it is self-signed

- For more information, see:

  `http://localhost:8080/docs/ssl-howto.html`

SSL Negotiation

Client has secret key (random)

- **Step 1**: It sends a random number rn2 encrypted by the secret key to server

Server has signed certificate and a private key

- **Step 2**: Server sends certificate to client
- **Step 3**: Client encrypts the secret key with certificate and sends to server
- **Step 4**: Server sends back to client rn2 encrypted by the secret key