CSE 135

Access Control
Authentication & Authorization

Access Control Mechanisms

• Declarative Authorization using Realms
  – The expression of app security is separate from your JSP and Java code
  – Access control to resources based on roles
    – Role: group of users that have access to particular resources
    – Resources: pages, action URLs in Struts, etc

• Programmatic
  – Your code is responsible
  – Choose when you need to create intricate access control strategies
Declarative Authorization Using Realms

• Really simple!
• Many mechanisms for specifying role/user pairs are “ready out of the box”
  – Memory, JDBC, DataSource and JNDI Realms
• Code eventually has access to who is the logged in user and what is his role
• Memory Realm: access control in <1hr
  – Users’ info can be provided in
    <TOMCAT_HOME>/conf/tomcat-user.xml
  – Unfortunately static and clear text passwords
• DataSource Realm
  – Users’ info is stored in DB (preferred – your project)

Authentication

• How does a user prove her identity?
  – login pages, passwords, etc

• Methods:
  BASIC
  DIGEST
  FORM (to be used in your projects)
**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**

**Motivation:**
- BASIC sends clear text password over http
  - Can manually employ HTTPS but will switch back to clear text once

**How DIGEST solves the problem**
- Browser encrypts (digests) password using the MD5 algorithm (or SHA, MD2)
- Poor support by browsers has killed method
Encryption and Security Basics – Part I

- **Private key (password)**
  - Server stores public key, i.e., encrypted version of private key
    - $\text{publicK} = f(\text{privateK}, \text{randomKey})$
  - During logging in, function valid() decides if private key matches public key
    - valid(\text{privateK}, \text{publicK})
  - Public key is useless to attacker!

- **Passwords and possibly other data (credit cards) sent by the browser must be encrypted**
  - Tricky protocol!

- **Client must verify that server is who it says it is**
  - Certificates

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
- “mary” 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
<?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="mary" password="mary" roles="admin"/>
</tomcat-users>
```

**Login Configuration**

**web.xml**

```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>/login-error.jsp</form-error-page>
    </form-login-config>
</login-config>
```
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!

Example pages

**Login page**

**Protected page**
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() %>
Authentication Method: `<%= request.getAuthType() %>`

```jsp
<% if(request.isUserInRole("admin")) { %>
   You are in <i>admin</i> role<br/>
<% } %>
```
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
  - Requires postgresql-8.4-701.jdbc3.jar in <TOMCAT_HOME>/lib
- **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/context.xml

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

<table>
<thead>
<tr>
<th>users</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>password</td>
<td></td>
</tr>
<tr>
<td>john</td>
<td>john</td>
<td></td>
</tr>
<tr>
<td>ted</td>
<td>ted</td>
<td></td>
</tr>
<tr>
<td>mary</td>
<td>mary</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>user_roles</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>role</td>
</tr>
<tr>
<td>john</td>
<td>student</td>
</tr>
<tr>
<td>ted</td>
<td>student</td>
</tr>
<tr>
<td>mary</td>
<td>admin</td>
</tr>
</tbody>
</table>
**JDBC resource declared to app server**

```xml
<Context path="" debug="5" override="true" reloadable="true">
  <Resource name="jdbc/ClassesDbPool" description="Classes DB Pool"
    driverClassName="org.postgresql.Driver"
    type="javax.sql.DataSource"
    auth="Container"
    url="jdbc:postgresql://localhost/access_control?autoReconnectForPools=true"
    username="postgres"
    password="postgres"
    defaultAutoCommit="false"
    maxActive="10"
    minIdle="0"
    maxIdle="5"
    maxWait="3000"
    removeAbandoned="true"
    removeAbandonedTimeout="60"
    logAbandoned="true"
    validationQuery="SELECT 1" />
  <Realm className="org.apache.catalina.realm.DataSourceRealm" ...
  </Context>
```

**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 Postgresql DB of DataSourceRealm used for security

// use Postgresql's MD5 function

PreparedStatement makeNewUser = con.prepareStatement(
    "INSERT INTO users(username, password) VALUES (?, md5(?))"
);
makeNewUser.setString(1, user);
makeNewUser.setString(2, pwd);
makeNewUser.execute();

Plenty of stronger encrypting functions on the web
• Jasypt
• jBCrypt
Enabling Secure Sockets Layers (SSL)

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

SSL Protocol

- Client & server need to know key for encrypting, decrypting messages.
- Server sends PuK
- Secret key encrypted by PuK
- rn2 encrypted by Secret Key
- Consequently material of rn2 used as encrypting, decrypting key
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`

```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**

```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`