# Lab 10: Working with Passwords

In this lab, you will work with hashed and salted passwords.

## Objectives:

- To work with hashed passwords

### Part 1: Project Setup

## Steps:

_1. First you must start the database server. You will use the free, open-source HSQLDB database to hold the user information.

Follow these steps to start the database:

a. Open a command prompt window and change to the **{Lab Installation Directory}/lib** directory. If your computer has the standard lab setup:

```
cd \javaclassp3\lib
```

b. Start the HSQL server:

```
java -cp hsqldb.jar org.hsqldb.Server -database regdb
```

Leave the database server window open for the remainder of the lab.

_2. If necessary, switch to the JEE Perspective.

_3. As you have in earlier labs, import the **{Lab Installation Directory/starters/lab10/lab10.zip** existing project into your workspace.

_4. Examine the imported files. Note the following in the *lab10Web* project:

- The Java Resources/src/domain package contains three classes. **User** represents a registered user of the application. **UserManager** contains methods to work with Users and **HashedPassword** has some utility methods to work with passwords.
- The Java Resources/src/dao package contains a class named **UserDaoImpl** that uses JDBC to access a database.
- The Java Resources/src/servlets package contains two servlets. **Controller** acts as the main application controller and **Login** responds to user requests to login or logout.
- The WebContent folder contains a **login.jsp** with a form into which the user can enter an email address and password.
- The WebContent/WEB-INF folder contains **populate.sql**, which is a database script you can run to create the database table and add a user or two.
- The WebContent/WEB-INF/pages folder contains the only "real" page of the application, **home.html**.


Copyright © Descriptor Systems 2011. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

Please ensure that you understand all of the provided pieces before continuing.

_5. Next, you need to run the script to create and populate the database table:

a. Back in Eclipse, choose *Window - Preferences*, then expand the *Data Management - Connectivity* category.

b. Click on *Driver Definitions*, then press the Add button.

c. In the Database category, find the HSQLDB JDBC driver entry and click on it.

d. Click the *Jar List* tab at the top of the pane. Remove any existing entries, then press the *Add JAR/Zip* button and browse to the **{Lab Installation Directory}/lib** folder and select *hsqldb.jar* and press Open followed by OK twice.

e. In Eclipse, from the menu, choose *Window - Show View - Data Source Explorer* to surface that view.

f. In the Data Source Explorer, right-click on the *Database Connections* entry and choose New.

g. In the Connection Profile Types list, click on *HSQLDB*, then in the *Name* entry field, enter **users** and press Next.

h. On the next page, for the *Database*, enter **regdb**.

For the *Database location*, enter **hsql://localhost/**.

Press the *Test Connection* button and ensure that it succeeds, then press Next followed by Finish.

You should see an entry in the Data Source Explorer view for the new database.

i. In the Project Explorer, double-click on *popuate.sql* to open it into the SQL editor.

j. Note that it creates a user with email address "sam@sam.com", a clear-text password "1234" and no salt. In later steps, you will hash the password and salt it.

k. At the top of the SQL window, for the *Type* select **HSQLDB_1.8**.

For the *Name*, choose **users**.

For the *Database*, select **regdb**

Press Ctrl+S to save these selections.

l. In the Project Explorer, right-click on populate.sql and choose *Execute SQL Files*. The script should run with no errors. You can close the SQL editor window.

_6. Test the application as it stands:

a. In the Project Explorer, right-click on *lab10Web* and choose *Run As - Run on Server*.

b. You should be able to login using the email address **sam@sam.com** and password **1234** and go to the home page. This application has no other functionality.

c. Log out, close the browser window, then stop Tomcat.

## Part 2: Password Hash and Salt

As it stands, the application stores passwords in the database in cleartext with no salt. In this part, you will update the application to store hashed, salted passwords.

### Steps:

_1. First, update the database with a hashed, salted password for *sam@sam.com*:

a. Open the *domain.HashedPassword* class into the Java editor and examine it.


Copyright © Descriptor Systems 2011. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

Note that it contains a *main* method that creates a hash and salt for the value "1234".

b. In the Project Explorer, right-click on HashedPassword.java and choose *Run As - Java Application*.

Note that it displays a digest and salt in the Eclipse console.

c. Open the WebContent/WEB-INF/populate.sql file. At the bottom, enter:

```
UPDATE users SET password='xxxxx',
   salt='yyyy' WHERE
   email='sam@sam.com';
```

For the 'xxxxx', copy and paste the digest from the Eclipse console. For the 'yyyyy', copy and paste the salt from the console.

d. In the populate.sql file, highlight the entire UPDATE command, then right-click on it and choose *Execute Selected Text*.

The command should execute with no errors.

e. Enter a new command into the populate.sql file and execute it in the same fashion as in the last step:

```
SELECT * FROM users;
```

Click the *Result1* tab to see the results and verify that the user has a hashed and salted password.

You can close the populate.sql window.

_2. Next, modify the *domain.UserManager* class so it works with hashed and salted passwords:

a. Find the *login* method in the UserManager class. Immediately following the *User retVal = null;* line, define a few variables:

```
boolean userExists = true;
String digest, salt;
```

b. Immediately following the line you just typed, write code so that if an attacker doesn't provide an email or password, the login method takes the same amount of time it would for a legitimate request:

```
if (email == null || password == null)
{
    // TIME RESISTANT ATTACK
    // Computation time is equal to the time needed by a legitimate
    // user
    userExists = false;
    email = "";
    password = "";
}
```

c. Existing code creates a Dao object and calls *findByEmail* and then checks the return for null.

Modify the *if* statement, so that if the return is NOT null, you initialize the digest and salt variables you defined earlier from the database.

If the return IS null, provide dummy values so that the later calculation takes the same amount of


Copyright © Descriptor Systems 2011. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

time for a "not-found" request as it does for a "found" request. Your completed *if* statement should look like:

```
if (user != null)
{
    digest = user.getPassword();
    salt = user.getSalt();
}
else
{
    // TIME RESISTANT ATTACK (Even if the user does not exist the
    // Computation time is equal to the time needed for a legitimate
    // user
    digest = "00000000000000000000000000=";
    salt = "00000000000=";
    userExists = false;
}
```

d. Write code to calculate a salted hash from the password that the user entered, then compare the calculated hash with the hash from the database. If they match, set the return value for this method to the found user:

```
byte[] bDigest = HashedPassword.base64ToByte(digest);
byte[] bSalt = HashedPassword.base64ToByte(salt);

byte[] proposedDigest =
  HashedPassword.getHash(1000, password, bSalt);

if (Arrays.equals(proposedDigest, bDigest) && userExists)
    retVal = user;
```

Highlight all those lines and choose from the Eclipse menu *Source - Surround with Try/Catch Block*.

_3. Run and test the application in the same fashion as you did earlier. It should work as before, but is much more secure since it no longer stores passwords in the database in clear text.


Copyright © Descriptor Systems 2011. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.