# Lab 11: Mitigating SQL Injection

In this lab, you will investigate an application that's vulnerable to an SQL injection attack and then fix the application so it's less vulnerable.

## Objectives:

- To mitigate SQL injection

### Part 1: Investigating an SQL Injection Vulnerablity

## Steps:

_1. If necessary, switch to the JEE Perspective.

_2. As you have in earlier labs, import the **{Lab Installation Directory/starters/lab11/lab11.zip** existing project into your workspace.

_3. In the Project Explorer, right-click on *login.html* and choose *Run As - Run on server*.

Note that application displays a login form as well as a form to let a user recover a password. This is already a bad sign - if an application offers to email forgotten passwords, this implies that it's actually storing passwords rather than hashing them.

**Note:** This application does not actually send an email since it's just for demonstration purposes.

_4. Start by seeing what the application does if you try to request a password for an email address:

a. Enter a made-up email address, e.g. **abc@abc.com** into the email-address input box, then press the *Send password* button.

Note that the application displays a page indicating that the email address was not found - now we know the application's normal behavior.

Press the "back" button to return to the login page.

_5. Next, let's see if we can tell if the application sanitizes input or if it simply takes user input and constructs SQL:

a. Enter **abc@abc.com'** (note the single-quote), then press the send-email button.

How does the application respond? This is a particularly vulnerable application - note how it shows the actual SQL it used in the error message! We can clearly see that it took the user input and simply concatenated it to build an SQL query.

We also now know the name of the database table **members** and one of the column names, **email**. If we were seriously attacking this application, this is valuable information.

Press the back button.

_6. Now let's see if we can trick the application into revealing a registered email address:

a. Enter **anything' OR 'x' = 'x** into the email input box. Be sure to get the quotes correct.

This SQL will always evaluate to "true", which will perhaps cause the query to return all rows from a table.

b. The application responds by telling us that it sent an email to bill@bill.com - we can hypothesize


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

that the application used the first row in the result set to get that email address.

We now know at least one legal email address for the application.

Again, if we were seriously attacking, we now have a lot of information to go on.

Press the back button.

_7. Now let's be malicious, and delete bill@bill.com from the application:

   a. Enter **bill@bill.com';DELETE FROM members WHERE email='bill@bill.com';--** into the email-address input box and submit the form.

      If the attempt is successful, it removed the row with that email address from the table. Press the back button.

   b. Enter **bill@bill.com** and submit the form - before we started, that was a valid email address - what happens now?

_8. As a final test, see if you can find another legal email address.

## Part 2: Mitigating SQL Injection

In this part, you will modify the application so that it's less vulnerable to SQL injection.

### Steps:

_1. The vulnerability you exploited in the last step was in the *Java Resources/src/servlets/EmailPasswordServlet* class. Open this class into the editor and make the following changes:

   a. Change the JDBC code so that it uses a PreparedStatement instead of a Statement.

   b. Modify the exception-handling code so that it reveals less information to an attacker.

_2. Run login.html on the server again and repeat the attack. Ensure that this servlet is no longer vulnerable.

_3. The *LoginServlet* class has similar SQL-injection vulnerabilities. Edit that class and correct them.


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