# **CPS 475/575 - Secure Application Development, Spring 2020**

Dr. Phu Phung

Department of Computer Science,  
University of Dayton

Lab 6 - Secure Session Authentication and   
Database Modification

Lab introduction and instruction released: Lecture 19 (3/31/2020)

Deadline: TDB

Introduction

In Lab 5, we have learned how to develop a simple yet secure login system that connects to a database to check if the provided username and password are valid. We have implemented SQL Prepared Statements to prevent SQL Injection attacks. We have learned to sanitize HTML outputs to prevent possible cross-site scripting attacks that can be leveraged to steal sensitive information such as cookies. We have deployed the system on HTTPS so that all transaction data including passwords, cookies are encrypted and protected.

This lab is an extension of Lab 5 by adding two topics:

* Session authentication: This to ensure that the user will stay logged in after authenticated. You will perform the session hijacking again in this lab to understand how unprotected session can cause broken authentication, the top 2 vulnerability in OWASP Top Ten for the last decade. After that, you will implement methods to protect the session to ensure that session hijacking will not be possible.
* Database Modification: After authenticated, the user will be able to do some tasks that unauthenticated users cannot do, for example, to change the password. You will learn how to add the session authentication to ensure the access control, and how to modify the database from PHP.

**Note:** Lab 6 is based on Lab 5, therefore you MUST have your Lab 5 code works properly, i.e., it can connect the database to check the username/password.

**Source code organization:** You need to push all your code for this lab under a folder labs/lab6 within your private repository. You need to include the URL of this folder at the beginning of your report. For example, in my case, the URL is:  
<https://bitbucket.org/phu-udayton/secad-pphung1/src/master/labs/lab6/>

**You will lose 5% of this lab grade if this URL is missing in your report and 5% if the code is not in your repository.**

### Report and submission

You need to write your report to submit via Isidore. Your report must be submitted in PDF file (reports not in PDF format will be returned without grading for this lab). Your report MUST have the course number, course name, the instructor, your name and UD email. You MUST organize your report as in this instruction, for example:

Task 1. xxx

1. yyy

You need to demonstrate your experiments by screenshots, therefore, **it is important to capture screenshots when you perform the lab**. **Your screenshots must have a short description or a caption to explain the task.**

I recommend you to use Google Docs to write your report as it is convenient to include the screenshots for your report. *f you use Google Docs to write the report, you can use an add-on such as “Code Blocks” (*[*https://gsuite.google.com/marketplace/app/code\_blocks/100740430168*](https://gsuite.google.com/marketplace/app/code_blocks/100740430168)*) to format and indent your code.*

## Task 1 (2 points): Data Protection and HTTPS setup

This is a homework task to ensure that you have set up and copied the code for the next steps. Follow the steps in Lecture 18 to set up HTTPS for your webserver.

You also need to copy your Lab 5 code to this Lab 6 by the following steps (introduced in Lecture 19):

* Create labs/lab6 folder in your private repo. Copy the files index.php and form.php from lab5 to lab6

$ cd <your-repo>/labs

$ mkdir lab6

$ cp lab5/index.php lab5/form.php lab6

$ cd lab6

* Create lab6 folder on the webserver root directory and copy the PHP files there :

$ sudo mkdir /var/www/html/lab6

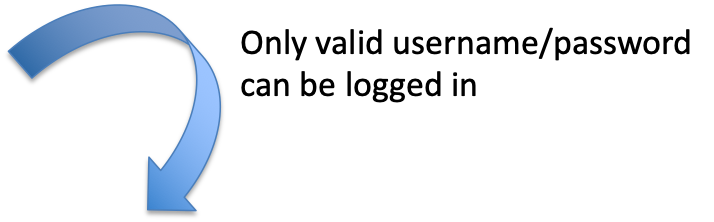
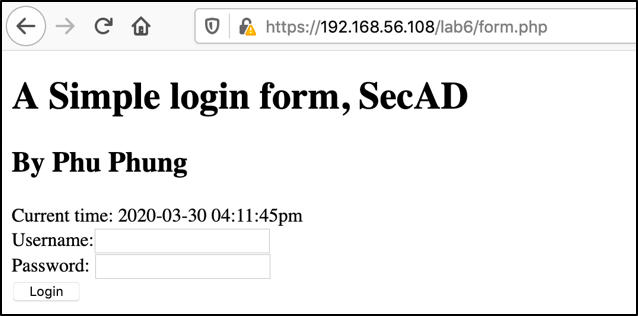
$ sudo cp \*.php /var/www/html/lab6

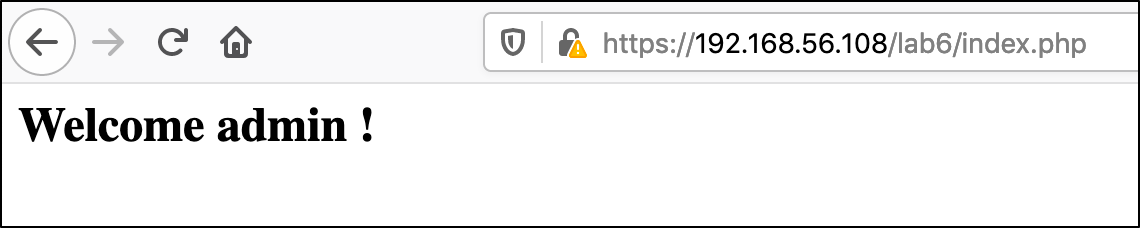
* Open the index.php file to edit:

$ subl index.php

Delete the checklogin and checklogin\_mysql functions are they are not needed

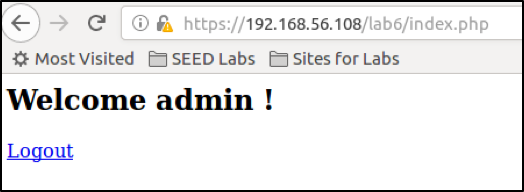
Capture the screenshot to demonstrate the successful HTTPS setup and Lab 6 preparation as below:

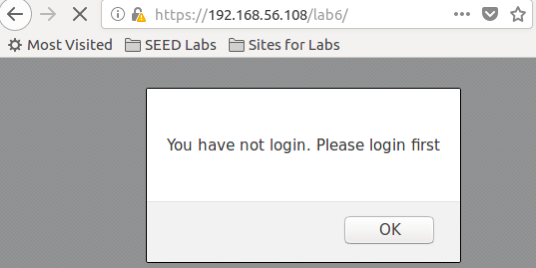


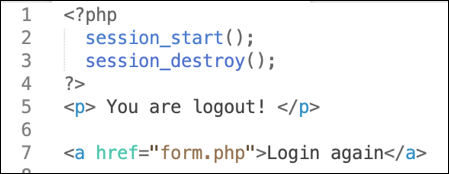


## Task 2 (14 points): Secure session authentication

1. Insecure session authentication
   1. (3p) Follow the steps in Lecture 19 (3/31/2020) to revise the index.php file to add the store the logged-in status into the session. Insert the new code (only the modified code) in your report and capture the screenshots of

* The direct access to the URL (without index.php) after authenticated, e.g:   
  
* The direct access to the URL (without index.php) when the user has not provided username/password, e.g:



* 1. (1 p) Implement the logout.php as introduced in the lecture. The code is provided again below:  
     

1. (5p) Session Hijacking - Broken authentication

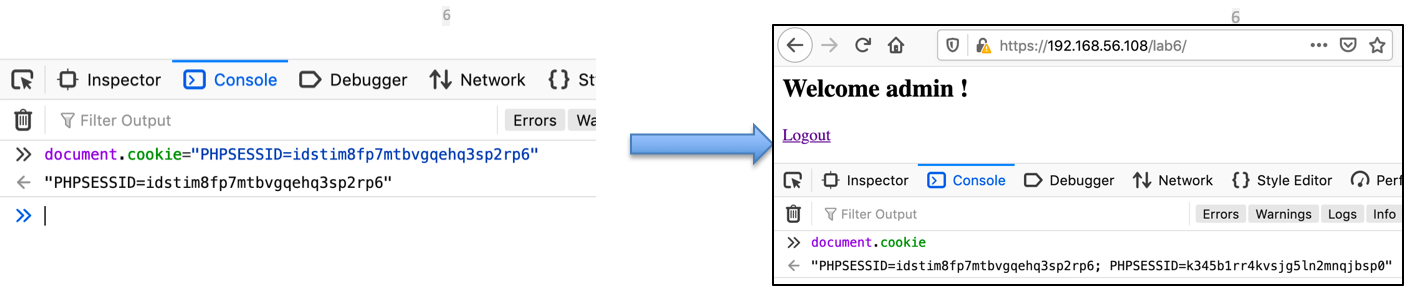
Session hijacking is a common attack in web applications where the attacker steals the session ID from the cookies of the victim and impersonates that victim so that the attacker logins to the system with the session ID without username/password. Therefore, a session hijacking attack is normally accompanied by another attack that steals the session ID such as XSS Attacks or network eavesdropping. Even though we have protected against XSS attacks and network eavesdropping using HTTPS in Lab 5, there are other possible channels that might leak the session ID such as unknown XSS Attack channel (Recall zero-day vulnerability) or any JavaScript code, e.g., third-party JavaScript. In this task, we will simulate the attacks to understand the consequences clearly so that we can secure the session to ensure a robust and secure session authentication.

In this sub-task, you will first act like a victim: Suppose that the victim has logged in to the system with a valid username/password. We simulate the session ID stealing by turn on the browser console (in Firefox), type: document.cookie to get and copy the cookie value. Capture the screenshot similar to the below to demonstrate this step (1p).



In another browser (e.g., in your laptop), you act as the attacker: Suppose the attack got the stolen session ID and set the cookie using the console, and then access the index.php page directly, the attacker will be logged in without any username/password.

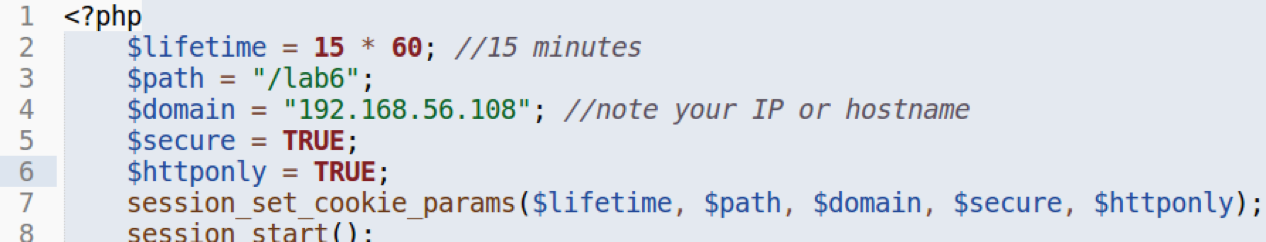
Capture the two screenshots as below to demonstrate this attack (2p):



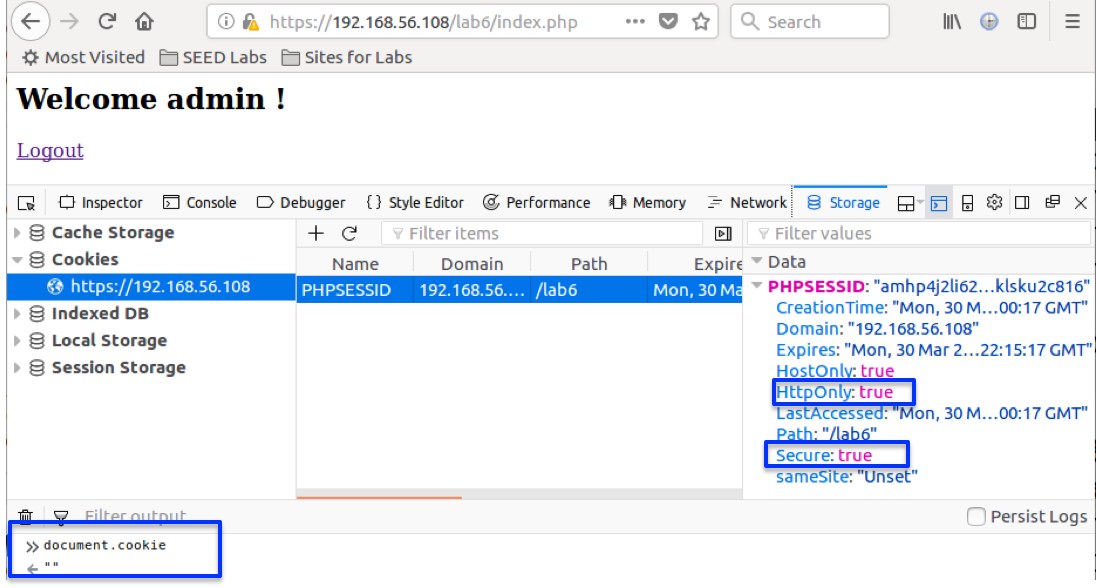
Explain in your report why the attack happened (2p).

1. (5p) Session Hijacking prevention
   1. (2p) Set the session cookie parameters

Follow the code introduced in the lecture (copy here again) to set the limit lifetime for the cookie, secure and httponly flag:



Deploy the new code to the server, clear the cookies and login again with a valid username/password. Try to access the cookie from the console again to observe the difference. Describe the difference in your report (1p) Capture the screenshot of this step together with cookie information from the browser console to include in your report (1p). The screenshot should be similar below:



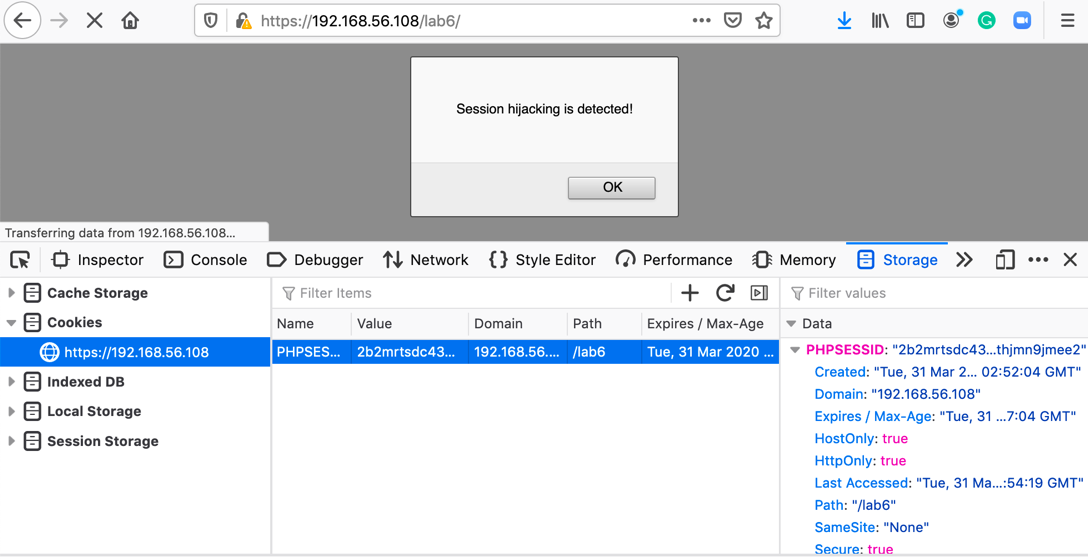
* 1. (3 p) Detecting the browser to prevent session hijacking attacks

A more robust method is to store the browser information in the session after successful authentication and check against it whenever we check the session for authentication. The reason is that session hijacking is an attack carried out by the attacker in a different browser.

The new code below (in the box) will store the browser information to the session:  
 and check it:



Implement this new code and deploy it. Follow the steps in the lecture to try the attack again and capture the screenshot demonstrating the attack failed as below.



## Task 3 (14 points): Secure Database Modification and Authorization

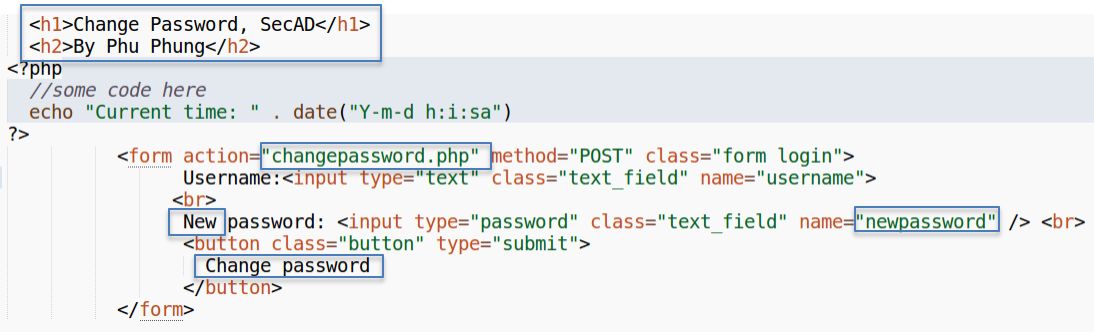
Note: this task is updated on April 5, 2020 for Lecture 21 and Lecture 22.

In this task, we will add a new use case into the login system: the logged-in user wants to change the password. We need to create a new PHP page with a form with username and the new password inputs, and then get the inputs and update the database in another PHP page

### Preparation (Homework):

To implement this scenario, we need to have two PHP pages:

* A PHP page with a form so that users can provide a new password, let’s name it **changepasswordform.php**. Follow the below steps to create this PHP page:
  + Copy the current **form.php** file and name it to **changepasswordform.php**:  
    $ **cp form.php changepasswordform.php**
  + Add the link to this page from the current index.php: ﻿  
    <a href="**changepasswordform.php**">Change password</a>
  + Change the HTML content and the form in **changepasswordform.php** as below (new/revised contents are highlighted in boxes - Replaced Phu Phung by your name):

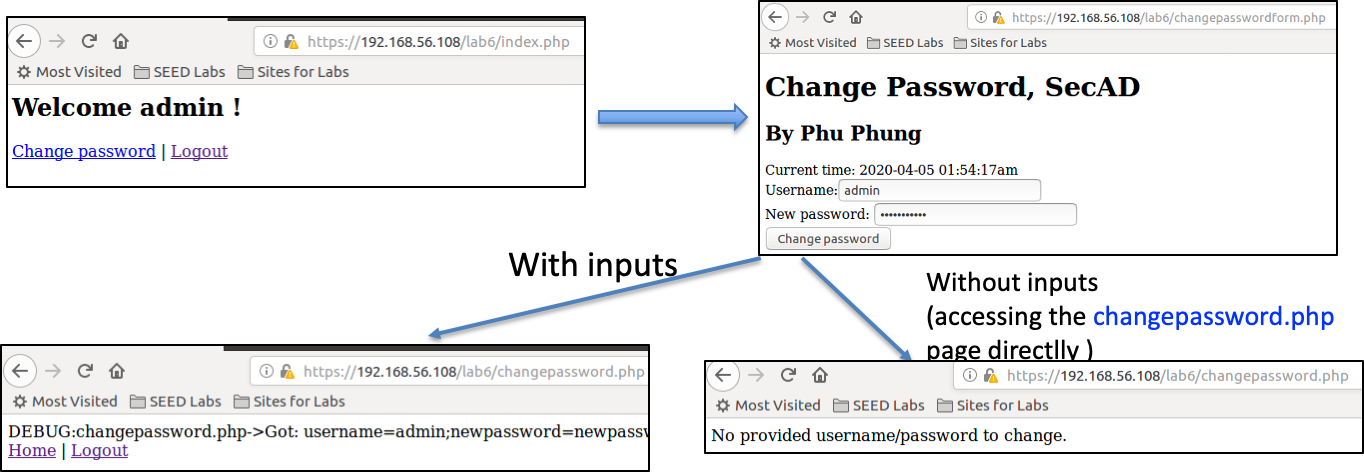


* A PHP page to handle the inputs and update the database , let’s name it **changepassword.php**. Let’s create a new changepassword.php file that just prints out the inputs as the code below:



You can now deploy all files to the webserver **(ensure that you are in your lab6 private repo folder)**:  
$ ﻿**sudo cp \*.php /var/www/html/lab6**

After deployment, you can test these new page by logging to the index.php page, then click on the link to **changepasswordform**.**php** to provide the inputs and submit. The input data will be submitted to the **changepassword**.**php** page. The scenario is illustrated as below.



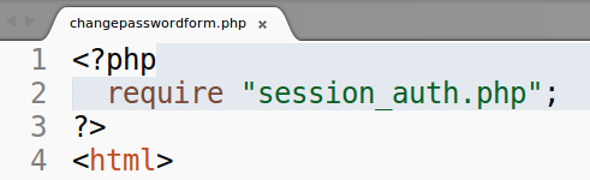
In addition, we need to create a new PHP page that only does the secure session authentication, e.g., **session\_auth**.**php**. We will require this file in any PHP pages that need authentication. To prepare for this task hands-on, you need to create such a file (**session\_auth**.**php**), and copy the code of session settings, validation and session protection from your current index.php file to this **session\_auth**.**php** file. The content of this file is below:

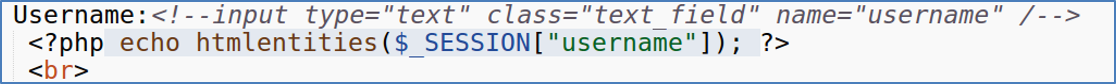
|  |
| --- |
| <?php  $lifetime = 15 \* 60;  $path = "/lab6";  $domain = "192.168.56.108";  $secure = TRUE;  $httponly = TRUE;  session\_set\_cookie\_params($lifetime,$path,$domain,$secure,$httponly);  session\_start();   //check the session  if( !isset($\_SESSION["logged"]) or $\_SESSION["logged"] != TRUE){  //the session is not authenticated  echo "<script>alert('You have to login first!');</script>";  session\_destroy();  header("Refresh:0; url=form.php");  die();  }   if( $\_SESSION["browser"] != $\_SERVER["HTTP\_USER\_AGENT"]){  //it is a session hijacking attack since it comes from a different browser  echo "<script>alert('Session hijacking attack is detected!');</script>";  session\_destroy();  header("Refresh:0; url=form.php");  die();  } ?> |

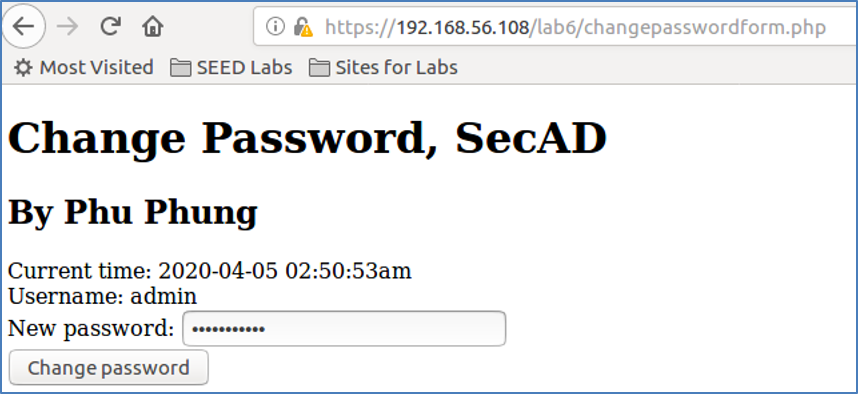
1. Secure Database Modification
   1. (1p) Database connection and handling: implement the **database.php** page with the below content (as introduced in Lecture 21):

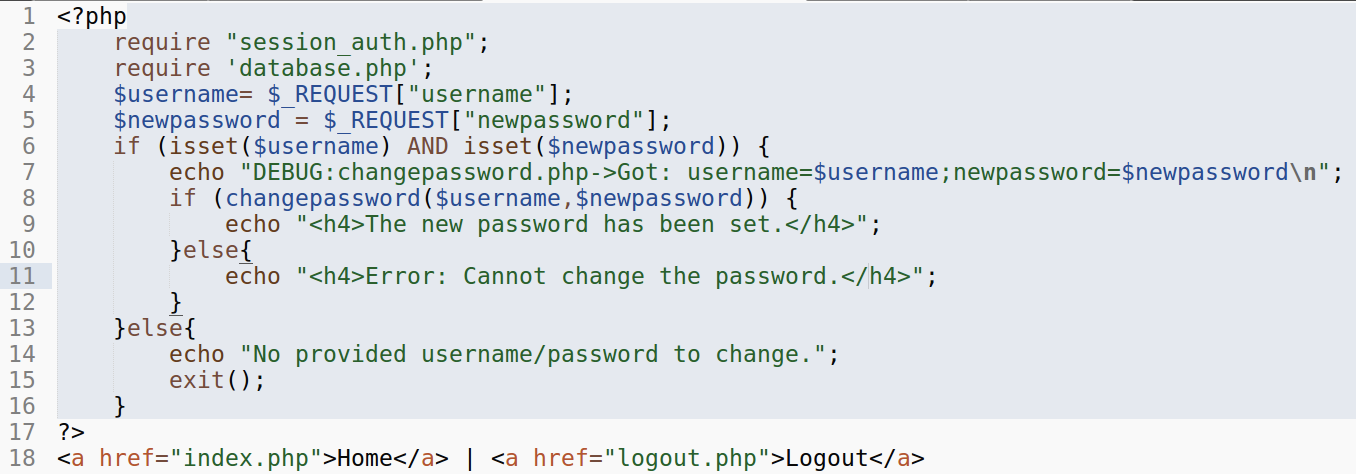
****

Include your code in your report.

* 1. (1p) Implement the **changepasswordform**.**php**, the version that requires **sesstion\_auth**.**php** and fixes the access control issue as below (introduced in Lecture 21):  
     

Deploy this revised page on the server and capture the screenshot of this page to include in your report. Your screenshot should be similar to the below:



* 1. (2p) Implement the **changepassword**.**php** page as below (introduced in Lecture 21):  
     

and revise it to get the username from session as introduced in the Lecture 21. Include the final code in your report.

* 1. (3p) Testing and evaluation  
     Capture the screenshots to demonstrate that a logged-in user can change the password and un-logged-in users cannot access the page. Does your system allow a user to change the password of other users? Why or why not?

1. Cross-site Request Forgery (CSRF) Attack and Prevention
   1. (2p) CSRF Attack: Perform an CSRF Attack on the **changepassword**.**php** page as introduced in Lecture 21/22. Capture the screenshot to include in your report and explain why this attack happened?
   2. (1p) Revise the **changepasswordform**.**php** page to generate a token, store it in the session and send it to the client as introduced in Lecture 22. Include the page in your report and highlight the revised code.
   3. (1p) Revise the and **changepassword**.**php** page to get the token from the client to compare with the information in the session to detect CSRF attacks as introduced in Lecture 22. Include the page in your report and highlight the revised code.
   4. (3p) Testing

Capture the screenshots to demonstrate that CSRF attacks are prevented and regular users can change the password normally. Explain why the attacks can be prevented with this method?