MySQL and PHP

CS4753: E-Commerce
Fall 2012

Assigned: 18 October 2012
Due: Noon 3 November 2012

The three programming assignments are worth 60% of the course grade. The first programming assignment was worth 20% of the 60%; this assignment is worth 50% of the 60%. Assignment 3 is worth 30% of the 60%.

Overview

This assignment is devoted to using PHP scripts and MySQL for implementing standard user/client functionality in your websites. The website's information (e.g., user logins and client accounts) is stored in a relational database that is accessed using PHP. The type of functionality that is expected out of your website could be, for example, if for your previous assignment you created:
(1) Shopping website - your clients need to be able to sign-up, login/logout, buy merchandise, send requests for order cancellation, create a wishlist or add items to their shopping cart (or both), send in inquiries, etc;

(2) Blogging website - your users need to be able to create an account, create a blog, write/delete/edit posts, create post categories, add/edit/delete tag words, receive comments which the author can then accept/reject, etc.;

Please adjust the functionalities according to the characteristics of your websites. **Use a minimum of 6 distinct functionalities!** **Distinct functionalities are those which do not belong to the same logical group of functionalities (e.g. write/delete/edit posts is **one** functionality).**

Objectives

The foremost objective of this assignment is to gain experience writing PHP code. Although PHP may be standalone, it is most frequently paired with a data source (e.g., a relational database) to create dynamic, data-driven websites. Thus, your code will integrate these two technologies (relational databases and server-side scripting) to complete the authentication and registration pages, as well as the user functionalities provided by your website. Also introduced is common security vulnerability, SQL injection, and how to protect against it. And an additional security requirement will be password hashing and salting (more information in section **PHP**).

What to turn in

You may complete this assignment individually or in teams of two (preferably same teams as in Assignment #1). The submission must include the source code (as detailed below), and a short document (1-3 pages) giving an overview of your work, choice reasoning, and issues you ran into while completing this assignment. If you elect to partner with someone, both partners must make a submission on Collab that clearly identifies your partner. Only one partner's submission must include the source code and document; it is not necessary for the other partner to make a duplicate submission.

Types of Users

All websites must have at least two different types of users - authorized users (registered users who can log in and have access to their own functionalities and data, but do not have access to other users' data), unauthorized users (non-registered users with view-only rights). An additional option is an admin user (user with full access to all functionalities and data).

MySQL and phpMyAdmin

MySQL is a popular open-source relational database management system with more than 100 million copies downloaded or distributed throughout its history. Many of the most popular Internet sites—including Google, Wikipedia, and Facebook—use MySQL. MySQL plays an important role in the Linux, Apache, MySQL, and Perl / PHP / Python (LAMP) software stack, due to its ease of acquisition and operability on many different software platforms.

In this assignment, you will be using phpMyAdmin, which is a popular open-source application for administering MySQL databases. phpMyAdmin provides a graphical user interface for common tasks, such as executing SQL statements, working with fields (add, edit, delete), working with tables (create, alter, drop), creating additional databases, and allowing the execution of arbitrary SQL statements.

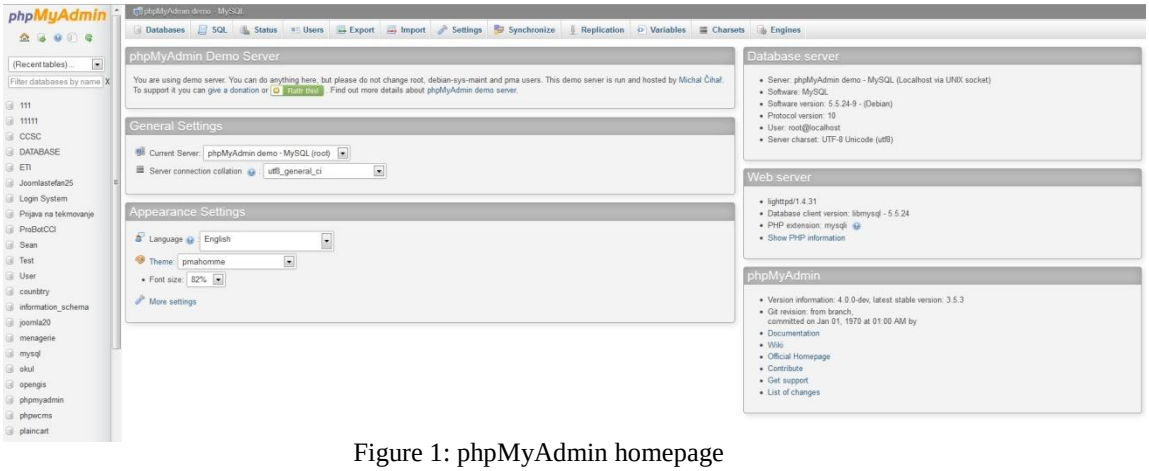

Figure 1: phpMyAdmin homepage

If you haven't done so previously, create a MySQL account on **http://dbm2.itc.virginia.edu** (or setup your own MySQL server if you can☺). Next (using the same website), create a database for this series of assignments. After creating the database, log into phpMyAdmin (**http://dbm2.itc.virginia.edu/phpMyAdmin)** with the username and password that you specified when creating the MySQL account. Once authenticated, you will be directed to the phpMyAdmin home page.

PHP

After setting up your MySQL database, write a PHP script for connecting to it.

Hint

```
$dbh=mysql_connect ("localhost", "username", "password")//THIS CONNECTS TO THE SERVER WITH YOUR DETAILS
    or die (mysql_error());//IF IT CANNOT CONNECT, THIS ERROR MESSAGE WILL SHOW
mysql_select_database ("db_name");//THIS SELECTS THE CORRECT DATABASE FROM YOUR SERVER
```

- **Authentication**

Either in a separate script, or in your connection script, implement a user login. Make sure to include a check for whether the user name is already taken (give an error when it is), password matching, and password encryption. After that implement a check of whether the login form has been submitted, and if it has, whether all the data in it complies with your requirements (e.g. password length, allowed characters in username etc.). If everything is valid, send it to the database.

Write a script which at user login checks whether the user exists and the user credentials are proper. If they are, and it is the first time that user logs in, create a cookie or session, and give the user access to users-only functionalities (e.g. buying merchandise, cancelling orders, etc.).

- **SQL Injection**

SQL injection is one of the most common website vulnerabilities. Attackers provide specially crafted input to web forms so that when the form is submitted, it circumvents the site's normal security, corrupts the backend database, or even provides the attacker with a means to access the backend database directly. Fortunately this form of attack is one of the easiest to defend against, requiring only "best practice" techniques to eliminate.

The most common form of SQL injection stems from improperly handled escape characters. Imagine constructing a query to add the artist "Eugen d'Albert" to your database (if you sell art work for instance). As a first attempt, the SQL query might look like:

```
INSERT INTO User ( name , email , username , password )
VALUES ( 'Eugen d'Albert' , 'eugen.dalbert@example.com' , 'dalbert' ,
'5f4dcc3b5aa765d61d8327deb882cf99');
```

Unfortunately, this query contains a syntax error: string constants in SQL are delimited by single quotes so only "Eugen d" is considered to be the name and the query concludes with an un-terminated string constant. SQL provides a way to escape single quotes that appear in string constants: replace each single quote with two single quotes (not a double quote). Thus, the correct query for inserting the new artist is:

```
INSERT INTO User ( name , email , username , password )
VALUES ( 'Eugen d''Albert' , 'eugen.dalbert@example.com' , 'dalbert' ,
'5f4dcc3b5aa765d61d8327deb882cf99');
```

Here the apostrophe (single quote) in d'Albert has been escaped. When executed, this query adds the appropriate information to the user table.

While this initial example is trivial, it illustrates the importance of handling escape characters correctly. Now consider several ways to malform the username. Including a single quote in the username will cause the code to break (because the database backend will reject the invalid SQL query), but there are more interesting attacks that can be exploited. Suppose you know that John Doe uses the site and his username is "doe". Is there any way you can exploit this knowledge and your knowledge of SQL (specifically, how strings are escaped) to log in as John Doe? In fact, there is a way to defeat the login mechanism entirely. Consider the strange looking username "doe' OR 'abc' = '123" and what happens if this username is entered. The resulting SQL query2 becomes

```
SELECT id FROM User WHERE username = ' doe ' OR ' abc ' = ' 123 '
AND password = ' d 4 1 d 8 c d 9 8 f 0 0 b 2 0 4 e 9 8 0 0 9 9 8 e c f 8 4 2 7 e ' ;
```

Because the AND binds more tightly than the OR, the password check is circumvented entirely. The SQL query logically evaluates like

```
username == "doe" OR (false AND . . . )
```

Even though the password check will fail, the OR condition gives the attacker access with only a valid username. The intentionally malformed username is an example of SQL injection. SQL injection allows users to alter the commands executed by the database backend. In this case, the additional OR condition eliminated the password check intended to verify the identity of the user. Additional knowledge of SQL allows attackers to issue more destructive queries although such examples will not be discussed here.

- **Cookies vs. Sessions**

Cookies are text files stored on the user's computers, while session variables are kept in memory on the server. Please, select either one of those two options for information storing, justify your choice in your report, and implement it. If you choose cookies, implement some simple mechanism for protection against at least one type of attacks against them.

- **Hashing and Salting Passwords**

Most e-commerce websites contain some private user information which should remain disclosed (passwords, payment details, etc.). If user passwords are stored in your database in plain text form they are easy to steal, hence we should use a protection mechanism and a popular one is hashing. A hash can be thought of as the digital fingerprint of a piece of data. You can easily generate a fixed length hash for any text string using a one-way mathematical process. Afterwards, it is nearly impossible to recover the original text from a hash alone. It is also vastly unlikely that any different text string will give you an identical hash - a 'hash collision'. These properties make hashes ideally suited for storing your application's passwords. While hashing one's password makes hacking that password more difficult, it does not make it impossible. One commonly known way of hacking hashed password is 'brute force' when the attacker randomly generated hashes until finding a match. Modern computers are capable of generating such random combinations at an immense speed, so that makes password hacking fairly easy. An additional security step that can be taken in this case is salting our passwords by making a small addition to our hashing algorithm. Before generating the hash we create a random string of characters of a predetermined length, and prepend this string to our plain text password. Provided the string (i.e. salt) is of sufficient length - and of course sufficiently random - the resulting hash will almost certainly be different each time we execute the function. Afterward, we must also store the salt we have used in the database together with the hash but that only makes the width of the field a few characters longer. Implement a password hashing and salting scheme, and explain how you did it in your report.

Grading Scheme

You will be given a grade (maximum 100) and this grade will be a part of your final programming assignment score. While your short report is not graded on its own, if you DO NOT submit it 10 points will be deducted from your final score! Here is how your grade will be made up:

- **Functionalities (30 points)**

A minimum of 6 distinct functionalities has to be created. Each functionality has to be fully working and must fit the concept of your website to receive the full 5 points. Points will be deducted if the functionality is buggy, or seems out of place with the rest of the concept.

- **MySQL Database (10 points)**

Setting up your database properly and feeding information to it is essential for this assignment. Please use the phpMyAdmin tool (as mentioned in section MySQL).

- **PHP Scripts (25 points)**

This assignment requires writing a number of PHP scripts for handling a lot of the functionalities, so having properly behaving scripts is crucial to your performance. Make sure your scripts are well formatted. Points will be deducted for buggy performance.

- **SQL injection attacks (10 points)**

Malformed usernames is a classic example of SQL injections. Handling them is the minimum requirement for SQL injection attacks for receiving the full 10 points.

- **Hashing & Salting Passwords (10 points)**

Hashing and salting passwords with PHP is a standard way of password encryption which helps level up the security. The minimum requirement is to use SHA-1 hashing algorithm. For extra credit use the SHA-256 or SHA-512 hashing algorithms.

- **Cookies and Session Variables (10 points)**

Either one of those two information storing option has to be implemented. Justification for your choice has to be given in your report!

- **User Types (5 points)**

At least 2 types of users have to be implemented (see section Types of Users). Additional work will receive extra points if necessary (score cannot be over 100 points).

Due Date – Noon 3 November 2012

When you submit your assignment send an email to **besaleva@virginia.edu** including your NAME, STUDENT ID and your ASSIGNMENT URL. In the subject of your e-mail add CS4753! If you have any database entries (stored users, passwords, etc.) DO NOT submit those!
All students must do a demonstration to the TA no later than 1 week after the deadline, so follow up with Lia about that in your submission e-mail!

Resources

1. phpMyAdmin Wiki
2. phpMyAdmin Basic Tutorial (video)
3. PHP login, encryption, cookies and redirection tutorial (very useful!)
4. SQL Injection Prevention Cheat Sheet

Those are just basic helpful resources but there is a plethora of suitable examples on the web. Search away! If that does not help, please join Lia during her office hours (MoTue 4-6pm, Rice 430).