(Student) - Paul Mahoney

C00227807

Abstract

Create an authentication mechanism for a web application using XAMPP, PHP & MySQL.

Authentication Mecahnism for a web application

Secure Apps Development

Contents

[Introduction 2](#_Toc67257499)

[Prerequisites 2](#_Toc67257500)

[Db.inc.php 2](#_Toc67257501)

[Generate.php 2](#_Toc67257502)

[Salt.php 3](#_Toc67257503)

[Sanitize.php 3](#_Toc67257504)

[Registration 3](#_Toc67257505)

[Allow a user to register with the system 3](#_Toc67257506)

[Complexity Rules 4](#_Toc67257507)

[Password should be salted and hashed. 4](#_Toc67257508)

[On an unsuccessful Attempt 4](#_Toc67257509)

[On successful attempt 5](#_Toc67257510)

[Greet the user 5](#_Toc67257511)

[Create an Authenticated Session 5](#_Toc67257512)

[View some pages. 5](#_Toc67257513)

[Allow for the user to logout securely. 6](#_Toc67257514)

[Max Session 1 hour 6](#_Toc67257515)

[Password Change 6](#_Toc67257516)

[Authenticated users may change password 6](#_Toc67257517)

[Complexity Rules enforced 6](#_Toc67257518)

[On password change session expire and reauthenticate 7](#_Toc67257519)

[Event Log & ADMIN User 7](#_Toc67257520)

[Log sign in attempts 7](#_Toc67257521)

[Accessible by only the admin user 7](#_Toc67257522)

[Account is created when the database is being created. 8](#_Toc67257523)

[Testing 9](#_Toc67257524)

# Introduction

In this document, I will be outlining my solution to each of the problems posed in the 30% Authentication mechanism project. I have broken this document down into the sections provided in the supplied brief. I will touch on the relevant information in each section and comment on my supplied code snippets. Before doing so I will briefly outline some notable information in the following section ‘Prerequisites’.

# Prerequisites

In this section, I will be outlining files I have made for reuse throughout the project. I stored all these in a resources folder to keep them in one place. As these files are reused throughout the website quite frequently, I felt the need to discuss them first to avoid repetition. There are a total of 4 files that will be discussed here the first being db.inc:

## Db.inc.php

Text

Description automatically generated

This is a skeleton file I still possess from 2nd year which connects to the database. I made one slight alteration to this for it to create the database if it does not exist. I did this to achieve the first problem in the brief “*create the underlying database on requesting the login page*”. I tried keeping this file separate to generate.php but the database must be created for any other file to access the database.

## Generate.php

Text

Description automatically generated

This file solely creates the user and logs tables, injecting the admin user in after their creation. Both these files are included on the login page while db.inc must be supplied to every page that wants to access the database. Although I could have wrapped the functionality of these files into one, I thought it was neater and easier to reuse if they were separate.

## Salt.php

Text

Description automatically generated

In addition to these files, I also created a salt.php file. This wraps both of the hashing functionalities into one file. The first function Salt() takes in the password and generated salt (which is done using uniqueid(math.rand)) and appends them together. This is then hashed using the built-in md5 PHP function and returned to the user. The second function is called saltDatabase() which takes in the username and password supplied. It then grabs the salt associated with that user from the database and assigns it to a variable. I then append these together again and md5 hash them to compare them to the hash stored in the database.

## Sanitize.php



The final of these functions is the sanatize.php file. This is used to filter out and potential malicious characters which may be used for XSS functions. It does so by simply replacing the special characters with their character entity counterparts. This essentially “takes the power” away from them so malicious JavaScript cannot be executed.

# Registration

## Allow a user to register with the system

Text

Description automatically generated

* The system first checks to see if anything has been posted. If it has not it builds the page with blank fields. Otherwise, it checks if the supplied username already exists. If not, we can continue.
* A simple check verifies if the user entered the password correctly by comparing it with the confirmation password. If any issues arise it will inform the user.
* Finally, if no issues arise with the SQL statement the username, password and salt are stored in the database and the user is redirected to the log-in screen.

## Complexity Rules

Text

Description automatically generated

* Password complexity is simple done by using the HTML pattern function. This only allows the form to be posted if certain criteria are met. In this case, the password must be 8 characters long containing at least one capital, lowercase, special char, and number. Otherwise, it will not post.

## Password should be salted and hashed.



* A random unique string is created. This is then appended onto the password and sent to the mentioned salt.php file where it gets md5 hashed. The salt is stored along with the hashed password in the database. This allows the salt to be retrieved for password verification.

## On an unsuccessful Attempt

In this section I will be discussing how I handled an unsuccessful login attempt:

Text

Description automatically generated

* After the entered username is sanitized and the password is salted and hashed as mentioned above, I check for this information with a SQL statement. If no rows were found this is considered an unsuccessful attempt.
* Attempts in incremented and the username has reflected the user in an error message. Additionally, the remaining number of attempts is here.
* If the number of attempts reaches 5 the user is locked out.
* Additionally, when an unsuccessful attempt is made various information is logged for the admin to view at their digression.

# On successful attempt

## Greet the user

Text

Description automatically generated

* When the user is logged in I pass both the username and remaining time down to the makehtml() function I mentioned earlier. I then display both the user on the page.

## Create an Authenticated Session

Text

Description automatically generated

* If the salted password and user exist within the database, we can now log the user in. First, I log the information of the successful user.
* I then set a session variable called logged in to true. This is responsible for preventing a malicious user from accessing authenticated pages.
* Finally, I redirect the user to the first page. At the start of this page, I create a session variable with the time which is used for the max session.

## View some pages.

Text

Description automatically generated

* A simple if statement checks on authenticated pages to logged in the variable is true. If it isn’t, I use the log-out file to destroy the session and any session variables. This also redirects you to the login page.
* This is on four pages in my system. Page1, page2, change password and logs. Each of these pages requires an authenticated user to view them with the addition of the authenticated user being an admin to view logs.

## Allow for the user to logout securely.

Graphical user interface, text, application

Description automatically generated

* This file is quite simple. It destroys all currently assigned session variables, destroys the session, and redirects the user back to the log-in screen.

## Max Session 1 hour

Text

Description automatically generated

* This is supplied on all authenticated pages. It takes the login time and subtracts it from the current time. If this equals 3600 (1 hour in seconds) it’ll trigger the user to be logged out.

## Inactive for 10 minutes

A screenshot of a computer

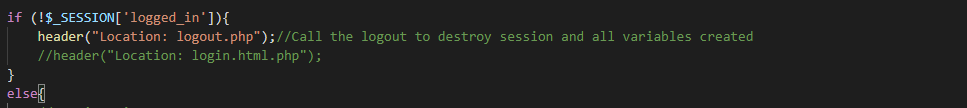
Description automatically generated

* A simple check that looks for the mouse to be inactive. If so it begins counting to 600 and logs the user out

# Password Change

In this section, I will discuss how the password change system works.

## Authenticated users may change password



* This works the same as any other page. An if statement checks, if the ‘logged in’ session variable, has been assigned. If it has not logout (destroy session) else, we can continue.

## On password change session expire and reauthenticate

Text

Description automatically generated

* The system performs various checks to see if the password combination is fine. A new salt is generated and appended onto the new password. This combination is then hashed.
* If there are no errors in the SQL statement, then it's executed and we can sign the user out.
* Calling logout.php destroys the session and all session variables assigned. The user is then redirected to the sign-in page where they may use their new credentials.

## CSRF

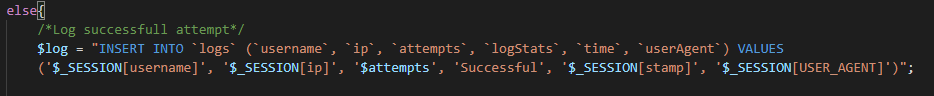
Text

Description automatically generated

* A token is created in log in which is assigned to a session variable. I pulled that session variable into change.php and passed it into a hidden field in the form. I retrieved this with the get method and compared them. If they don’t match, then CSRF has been attempted otherwise we can move on.

# Event Log & ADMIN User

## Log sign in attempts



* A simple SQL statement is used to log relevant information.
* I have positioned three of these in the login file. The first logs an unsuccessful attempt. The second logs a lockout after 5 attempts and the third logs a successful attempt. The only difference between them excluding location in the code is the value for the logStats field. This reflects what is being logged. As seen in the above image it says ‘successful’. The other two options are ‘unsuccessful’ and ‘lockout’.

## Accessible by only the admin user

Text

Description automatically generated

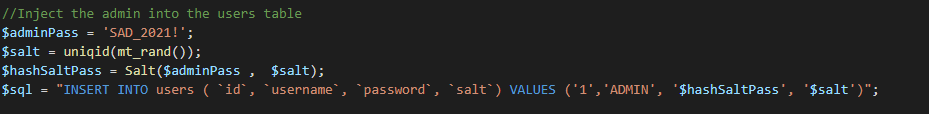
* For the logs page to be only accessible by an admin I edited my usual authentication check to also check if the logged user is admin. Now they have to be both an authenticated user (Logged in) and have the username ‘ADMIN’ to access the page.

Text

Description automatically generated

* Additionally, I took advantage of my makehtml() function and create an makeAdminHTML() function. The only difference between the two is the logs appear in the admin's navbar, but this functionality could be used further to provide an admin with an array of different information than a normal user. In the example of my page, the Lorem Ipsum could be hidden from an admin so they could focus solely on different tasks.

## The account is created when the database is being created.



* This snippet of code is from the bottom of generate.php. It simply generates the admin user and injects it into the user's table once it has been created.

# Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No | Test Name | Description | Expected Outcome | Outcome |
| 1 | persistent XSS on Login screen | <Script>alert(“Hacked!”)</script> | XSS Fails | XSS Fails |
| 2 | persistent XSS on create a screen | <Script>alert(“Hacked!”)</script> | XSS Fails | XSS Fails |
| 3 | persistent XSS on change password screen | <Script>alert(“Hacked!”)</script> | XSS Fails | XSS Fails |
| 4 | Create an account | Username:”test” Password: “notsecure” | Fail | Fail - password not secure |
| 5 | Create an account | Username: “test2” password: “SecurePass123!” | Succeed | Succeed |
| 6 | Log in with test2 | Username: “test2” password: “SecurePass123!” | Succeed | Succeed |
| 7 | Change Password | OldPass “SecurePass123” newPass: “SecurePass123”  conPass: “SecurePass123” | Fail | Fail – Passwords match |
| 8 | Change Password | OldPass “SecurePass123” newPass: “SecurePass1234”  conPass: “SecurePass1234” | Succeed | Succeed |
| 9 | Access authenticated pages as unauth | /page1 /page2 /change /logs | Fail | Fail |
| 10 | Access authenticated as normal user (Not admin) | /page1 /page2 /change | Succeed | Succeed |
| 11 | Access authenticated as normal user (Not admin) | /logs | fail | fail |
| 12 | Access authenticated as admin | /page1 /page2 /change /logs | succeed | Succeed |
| 13 | SQL injection | INSERT INTO `users` ( `username`, `password`, `salt`) VALUES ('1, '1, '1) | Fail | Fail |
| 14 | 1 hour timeout | Wait one hour | Succeed | Succeed |
| 15 | 10 minute timeout | Leave mouse unattended for 10 minutes | Succeed | Succeed |
| 16 | Reflective XSS | ?"><script>alert("hacked")</script> | Fail | Fail |
| 17 | Dom based XSS | #?"><"script>alert(“Hacked”)</script> | Fail | Fail |