

Logins, Sessions and Cookies

INFO/CS 2300:
Intermediate Web Design and
Programming

1 sheet on half wall

Project 3 M2 due date

If you submitted M1 before Thurs 3/16 at midnight
and did not receive a grade within 48 hours
please see Piazza @894 for due date options

Otherwise, the M2 due date remains
Tues 3/21 5PM

Project 3 Update

Schema – if you chose **album_title** as your primary key, think about how to create a url to view a single album. A **title can have spaces, a URL can't**. Additionally, if the title changes, you'll need to **cascade the updated value** to related tables

Feedback

Read the TA comments

If you are not going to implement the suggestions you need to have a good reason why and explain it with your file submission.

HW2

There are at least 3 questions that the automatic grader did not handle well. If the question had an ORDER BY component but we didn't ask you to order all the fields in the output, the remaining fields could be in random order and the auto-grader registered that as failure. See Piazza for details.

Simple logins: *A warm-up*

Site logins

With what we've covered so far in the course, we could build a login for a single page.



```
<?php
if ( ! isset( $_POST['username'] ) && ! isset( $_POST['password'] ) ) {
?>
    <h2>Log in</h2>
    <form action="login.php" method="post">

        Username: <input type="text" /> <br>
        Password: <input type="password" /> <br>

        <input type="submit" value="Submit">
    </form>
```

```
<?php
} elseif ( <input type="text" /> ) {
?>
    <p>You have accessed the secret content of this page.</p>
<?php
} else {
```

```
<?php
if ( ! isset( $_POST['username'] ) && ! isset( $_POST['password'] ) ) {
?>
    <h2>Log in</h2>
    <form action="login.php" method="post">

        Username: <input type="text" name="username"> <br>
        Password: <input type="password" name="password"> <br>

        <input type="submit" value="Submit">
    </form>
```

```
<?php
} elseif ( $_POST['username'] == "smohlke"
           && $_POST['password'] == "mypassword" ) {
?>
    <p>You have accessed the secret content of this page.</p>
<?php
} else {
```


Click In!

Click In!

Which of the following is not a problem with this type of login?

- A. Only works for one page
- B. Password stored as plain text
- C. Only allows one username/password
- D. Have to update the file for password changes
- E. These are all problems with this login

Click In!

Which of the following is not a problem with this type of login?

- A. Only works for one page
- B. Password stored as plain text
- C. Only allows one username/password
- D. Have to update the file for password changes
- E. These are all problems with this login

Multiple pages via sessions



\$_SESSION

Session variables:

- Can be **set and read** by PHP
- **Persist** from page to page.
- **Temporary**: Usually expire when the user closes the browser
- **Stored on the server**
- Are **associated with a browser** session
- Only good for **small quantities of data**

Sessions persist page to page

When a user successfully logs in, we'll **set a session variable**. On the other pages, we'll check if this is set.

Starting sessions

We start a session on any page in which we want to set or read a `$_SESSION` variable

This must start *before* the HTML.

```
<?php session_start(); ?>
```

```
<!DOCTYPE html>
```

```
<html>
```

```
...
```

The login in a session variable

Then we store information in the associative array `$_SESSION`.

```
if ( // username and password are correct ) {  
    echo "<p>Congratulations, $username ...<p>";  
    $_SESSION[ 'logged_user' ] = $username;  
} else {  
    echo '<p>You did not login successfully.</p>'  
    echo '<p>Please <a href="login.php">login</a></p>'  
}
```


The login code

```
$username = filter_input( INPUT_POST, 'username', FILTER_SANITIZE_STRING );
$password = filter_input( INPUT_POST, 'password', FILTER_SANITIZE_STRING );

if ( empty( $username ) || empty( $password ) ) {
    //Show login form
} else {
    if ( $username === 'smohlke' && $password === 'mine' ) {
        echo "<p>Congratulations, $username ...<p>";
        $_SESSION[ 'logged_user' ] = $username;
    } else {
        echo '<p>You did not login successfully.</p>';
        echo '<p>Please <a href="login.php">login</a></p>';
    }
}
```

Other pages

```
<?php session_start(); ?>
```

```
<!DOCTYPE html>
```

```
<html>
```

```
...
```

```
<?php
```

```
if ( isset( $_SESSION[ 'logged_user' ] ) ) {
```

```
    //Protected content here
```

```
    $logged_user = $_SESSION[ 'logged_user' ];
```

```
    print "<p>Welcome, $logged_user !</p>";
```

```
} else {
```

```
    print "<p>Please <a href='login.php'>login</a></p>";
```

```
}
```

```
?>
```



filter_input would
be better

Ending a session

```
<?php session_start(); ?>
```

Need to start sessions on the page in which you wish to end them

```
<!DOCTYPE html>
```

```
<html>
```

```
...
```

```
<?php
```

```
unset($_SESSION[ "logged_user" ] );
```

Clear one session variable

```
unset( $_SESSION );  
$_SESSION = array();
```

Clear ALL session variables. Not reversible

```
session_destroy();  
?>
```

Not as permanent as it sounds. One can start a session and access it again

Sessions aren't just for logins

Good for things like shopping carts

Click In!

Click In!

Which is not true about Session variables

- A. They always remain valid as long as the browser is open
- B. They are stored on the server
- C. They are good for small amounts of data
- D. They persist through a page reload
- E. They persist on other pages on the same server

Click In!

Which is not true about Session variables

- A. They always remain valid as long as the browser is open
- B. They are stored on the server
- C. They are good for small amounts of data
- D. They persist through a page reload
- E. They persist on other pages on the same server

Back to logins

Sessions take care of the first problem

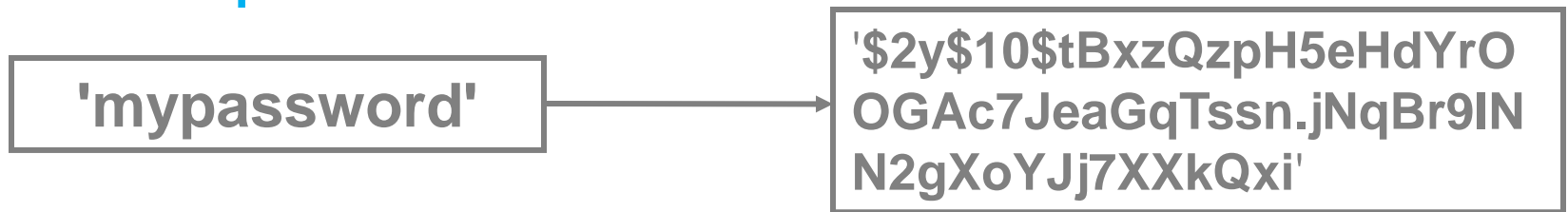
- ~~• Only works for one page~~
- Password stored as plain text
- Only one user
- Have to update the file for password changes

Securing passwords with hashing

We don't want to have the user's password visible in our PHP file. What can we do?

Hashing

A cryptographic hash function obscures the password.



Hashes should be
“one-way”

Hashes should be
“collision-free”

A hash function

```
password_hash( 'mypassword', PASSWORD_DEFAULT)
```

creates a new password hash

The above produces this

```
$2y$10$tBxzQzpH5eHdYrOOGAc7JeaGqTssn.jNqBr9IN  
N2gXoYJj7XXkQxi
```

Hash the password

```
$post_username = filter_input( ..., 'username', ...);  
$post_password = filter_input( ..., 'password', ...);  
$valid_password = password_verify( $post_password,  
    '$2y$10$tBxzQz pH5eHdYrOOGAc7JeaGqTssn.jNqBr9INN2  
gXoYJj7XXkQxi' );  
  
if ($post_username == "smohlke" && $valid_password ) {  
    $_SESSION['logged_user'] = $post_username;  
}
```

2 down, 2 to go

Sessions take care of the first problem

- ~~Only works for one page~~
- ~~Password stored as plain text~~
- Only one user
- Have to update the file for password changes

Multiple users via MySQL



Multiple users

We can **create a table** in which we store valid users and their (hashed) passwords; check this upon login.

Users(username: string, name: string,
hashpassword: string)



As of March 2017 this needs to be 60 characters but make it 255 to future-proof it

<http://php.net/manual/en/function.password-hash.php>


```
CREATE TABLE IF NOT EXISTS `users` (  
    `userID` int(11) NOT NULL AUTO_INCREMENT,  
    `hashpassword` varchar(255) NOT NULL,  
    `username` varchar(50) NOT NULL,  
    `name` varchar(50),  
    PRIMARY KEY (`userID`),  
    UNIQUE KEY `idx_unique_username` (`username`)  
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8  
    COLLATE=utf8_unicode_ci AUTO_INCREMENT=1;
```

```
$mysqli = new mysqli( ... );
```

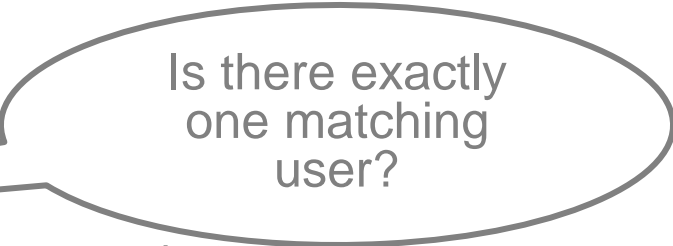
```
$query = "SELECT *
```

```
FROM users
```

```
WHERE
```

```
    username = '$post_username'; "
```

```
$result = $mysqli->query($query);
```



Is there exactly
one matching
user?

```
if ( $result && $result->num_rows == 1 ) {
```

```
    $row = $result->fetch_assoc();
```

```
    $db_hash_password = $row[ 'hashpassword' ];
```

```
    if( password_verify( $user_passwd, $db_passwd ) ) {
```

```
        $_SESSION['logged_user'] = $post_username;
```

```
    }
```

```
}
```

Logouts

What do we need to do to log out a user?

```
<?php
    session_start();
    if (isset($_SESSION['logged_user'])) {
        $olduser = $_SESSION['logged_user'];
        unset( $_SESSION[ 'logged_user' ] );
    } else {
        $olduser = false;
    }
}
```

```
?>
```

```
<!DOCTYPE html>
```

```
<html>
```

```
...
```

```
<?php
```

```
    if ( $olduser ) {
```

```
        print( "<p>Thanks for using our page, $olduser!</p>");
```

```
        print( "<p>Return to our <a href='login.php'>login page</a></p>");
```

```
    } else {
```

```
        print( "<p>You haven't logged in.</p>");
```

```
        print( "<p>Go to our <a href='login.php'>login page</a></p>" );
```

```
    }
```

```
?>
```

Some security issues

Password hash issues

There's sometimes an easy way to figure out what a password is given its hash.
What is it?

Bad guy method

1. Take lots of common passwords
2. Compute all of their hashes
3. Get access to the hashed passwords of lots of users
4. See if any of their hashed passwords match yours

A fix

Check passwords for common words and variants before accepting them.

```
<?php
    if ( isset( $_POST[ 'new_password' ] ) ) {
        if ( ! crack_check( $_POST[ 'new_password' ] ) ) {
            print( "Choose another password: " );
            print( crack_getlastmessage() );
            // Go back and get another password
        }
    }
?>
```

NOTE: These functions not a part of standard PHP installs.
Not expected for this course
www.php.net/manual/en/function.crack-check.php

Add salt

Add extra information (e.g. a random string)
before hashing the password

```
$salt = 'yadayadayada';
```

```
...
```

```
if (...hash('sha256', $post_password . $salt) ==  
    $row[ 'hashpassword' ] ) {
```

Hashed credentials in db

Major problems solved

- ~~Only works for one page~~
- ~~Password stored as plain text~~
- ~~Only one user~~
- ~~Have to update the file for password changes~~

Cookies



Cookies

Cookies are one way to save limited amounts of information between visits of a user.

Cookies are stored in the browser

Saving a cookie

`setcookie($name, $value)`

`setcookie($name, $value, $expiration)`

Without `$expiration` argument, cookie goes away when the browser closes. `$expiration` is number of seconds after January 1, 1970. Set as follows:

```
setcookie( "name", "value", time() + 60 * 60 );
```

Must be at the top of the .php file (*before* the DOCTYPE or any other HTML).

Getting a cookie

If a cookie was set using `setcookie("name", "value")`, and the expiration date has not passed, then when the user returns, the variable `$_COOKIE["name"]` will contain "value".

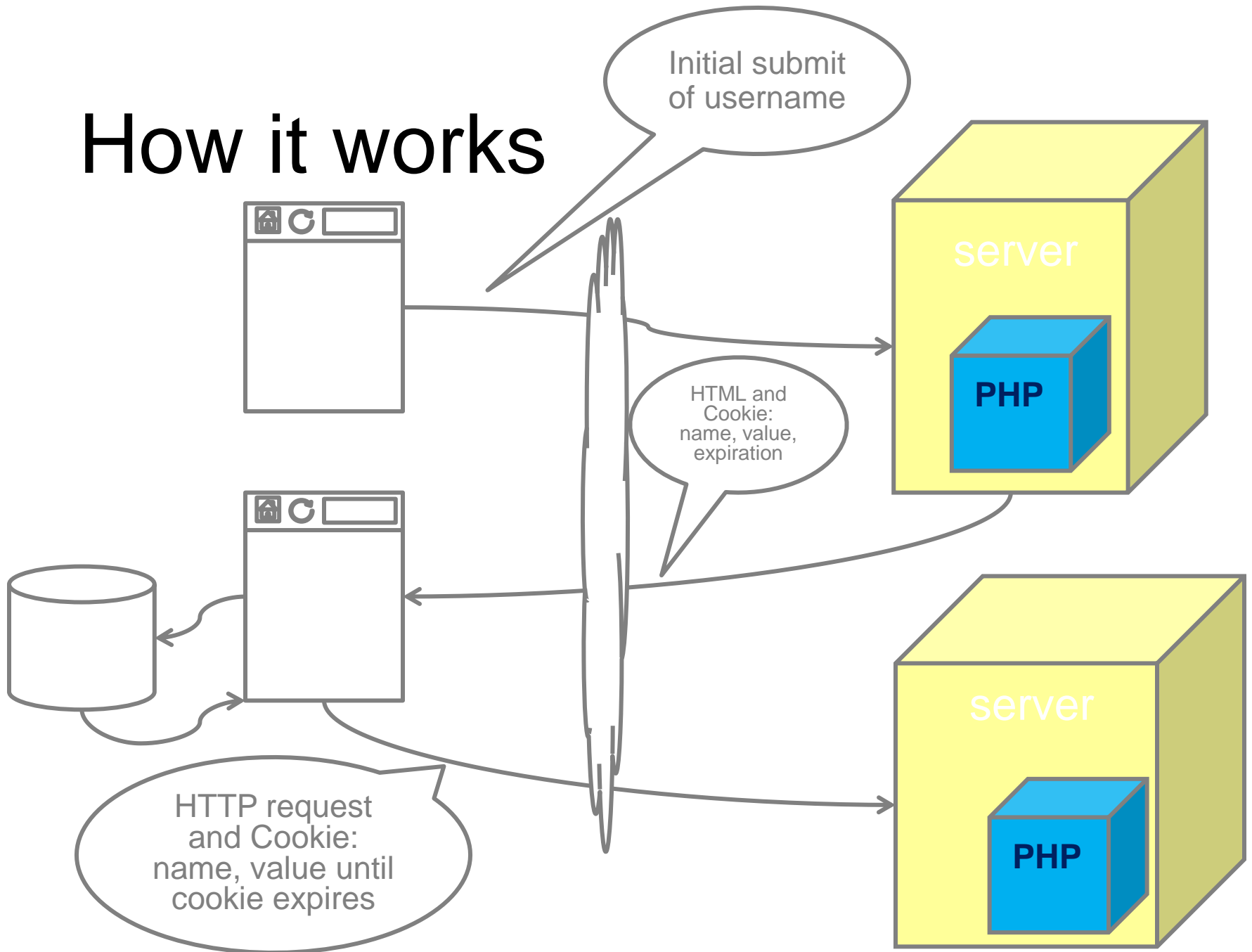
```
<?php
$username = filter_input( INPUT_COOKIE, 'username', ... );

if( empty( $username ) ) {
    $username = filter_input( INPUT_POST, 'username', ...);
}

if( empty( $username ) ) {
    ?>
    <form method="post" action="cookie-form.php">
        <p>What is your name?
        <input type="text" name="username">
        </p>
        <input type="submit" value="Click to submit">
    </form>
    <?php
} else {
    print("<p>Welcome, $username!</p>");
}
?>
```



How it works



Deleting a cookie

Set its expiration date to be in the past.

```
setcookie('name', 'value', time() - 3600);
```

How long? / How big?

The largest time that can be represented by a 32-bit integer is January 19, 2038.

By then, all browsers will be able to handle later dates.

Max cookie size is browser dependent but staying under 4k is safe

Cookie Security

Think of cookies the same way as JavaScript. Because they are stored / processed in the browser, they are inherently insecure. Don't store anything sensitive in a cookie.

Sessions use a cookie

Creating a PHP Session on the server creates a cookie named PHPSESSID.

Security: On [http](#), cookies are visible to anyone monitoring network traffic. If someone intercepts your PHPSESSID, they can pretend to be you.

With [https](#), all headers and content are encrypted so much more difficult to intercept.

Sessions and Cookies

Improve session security with a cookie

Instead of `$_SESSION['logged_user'] ...`

`$loggedin=`

`($_SESSION['cookie']==$_COOKIE['session'])`

<http://stackoverflow.com/questions/17413480/session-spoofing-php>

Cookies for logins

Since cookies can be faked, how could this be done safely?

- Can't store 'logged_user' = 'steve'
- Insecure to store even a hashed password
- Store a token that is deleted when logging out and expires in a reasonable time.
- Just one token per user or can the user log in from multiple browsers simultaneously?