Assignment 2: Authentication and Sessions

Due Thursday, Oct 2, 2014, at 11:55 PM.

For this PA, you will continue working on the photo album website developed in PA1. However, do not touch the files in the pa1 sub-directory. Make another sub-directory called pa2, and copy the files from pa1 into the pa2 sub-directory and work on the files there. By the end of this programming assignment you will learn how to authenticate users and maintain sessions.

Part 1: Getting started

These sites contain useful tutorial and reference information for what you'll be implementing in this PA.

- HTTP Cookies (http://en.wikipedia.org/wiki/HTTP_cookie)
- Sessions: Python (http://flask.pocoo.org/docs/quickstart/#sessions) or PHP (http://www.php.net/manual/en/ref.session.php)
- Authentication: Python (http://flask.pocoo.org/snippets/8/) or PHP (http://php.net/manual/en/features.http-auth.php)

Part 2: Build the website

This PA is about personalization. The first step to doing any kind of personalization is to keep track of who is browsing your site. In class we discussed how HTTP is a stateless protocol, which cannot itself retain data from one request to the next. The way to maintain state from one page request to another is using ***sessions***. In this PA we will add a login page to the site. Users will only need to type their username and password once. Thereafter the system will use session variables to determine who the logged in user is.

Pages that are ***sensitive*** require users to login before they can view those pages. The rest of the pages will remain public and will not require a username or password to be viewed. Whenever a user tries to enter a sensitive page you should make sure that he/she has the privileges to view it. This is done by checking if the user has a valid session, and if so, whether the user is authorized to view the page.

Some pages do not require user authentication or sessions (e.g., a default home page or a create account page). Some other pages only require that the user be authenticated and are not dependent on who the user is (e.g., a logged in home page). Others may provide access depending on who the user is and whether he/she is permitted to access that page (e.g., someone's private album).

In short, when a user requests a URL, you should:

- Check if the page is public. If so, view the page.
- Check if the page is sensitive and the user has a valid session. If so, check if the user has permission to see the page, and if so, let them see the page.
- If the page is sensitive and the session has expired, say so and give the user a link to the Login page.
- If the page is sensitive and there is no session, then state that one must authenticate to view this page and give a link to the Login page.

In some of the cases above, the user is redirected to the Login page. This Login page needs to remember what the requested sensitive page was using a query parameter. For example: `/login?url=/the/prev/url` After the user types her name and password she should be returned right back to the sensitive page she previously tried to access via a redirect again. Note, some of the pages can be sensitive some of the time and public the rest of the time. For example, the View Album page is only sensitive if the album is private.

Your code should observe the following rules about access privileges:

- Public albums are ***accessible*** to both logged in users and unauthenticated visitors.
- Private albums are ***accessible*** only to those users that have explicit access to that album. Users will have access to user's private album if and only if there exists a tuple (a, u) in the *AlbumAccess* relation (see below).
- A picture that does not belong to any public albums is accessible to a user if and only if it belongs to at least one private album that ***u*** has permission to access to.

Add public/private feature to the albums

Update Album Table

You need to add `access` attribute to this table, so the new scheme for Album will be

- Album (***albumid***, title, created, lastupdated, username, access)

`access` specifies whether access to the album should be limited to a set of users indicated in the *AlbumAccess* table (described below). It only takes values of "public" or "private."

Create new Table AlbumAccess

- AlbumAccess (***albumid***, ***username***)

This relation indicates the users who have access to each specific album.

In `/album/edit`, the user should be able to edit the `access` permission for the ***Album*** table. The user should also be able to give/remove user's access by editing the AlbumAccess table. Thus an album could be public, private or private with someone accessible. The interface for `/albums/edit` should appear roughly as below:

Album	Access		

Summer 2011 in Iceland	Public	[Edit]	[Delete]
Spring break 2010 in Brooklyn	Public	[Edit]	[Delete]
Thanksgiving 2010	Public	[Edit]	[Delete]
New: ___	___	[Add]	

Session Management

Session variables are stored at the server. Clients that are part of a live session present a session ID to the server with each request, either via cookies or via a URL parameter. That session ID allows your server side code to figure out which set of session variables are relevant to the current client. Thus, the only value the client really has to provide is the session ID, though your application can explicitly transmit other cookie values if you like. Server side scripting languages have built-in support for managing session IDs and the server side session variables that go along with them.

You should enforce a session inactivity time limit of 5 minutes. If a user tries to continue a session after 5 minutes or more of inactivity, then log out the user, destroy the session, and force the user to log in again.

You may want to maintain two session variables: `username` and `lastactivity`. The `username` stores the user name of the authenticated user. The `lastactivity` stores the time of user's last activity to check inactivity time out.

Sessions in PHP:

Created with the `session_start()` command. This will test to see if there's a current session. If not, it will create one. If so, it will use the session ID to bind the `$_SESSION` variable. This array is rebound to a different value, depending on the session ID presented by the client. Sessions are destroyed with `session_destroy();`

Sessions in Python:

In Flask sessions are started automatically. The session variables can be accessed as such: `session['username']`. In order for sessions to work properly however a secret key must be set. Please refer to the Flask docs for more infomation about how to use sessions. Be sure that sessions are imported when attempting to use them. Sessions work similarly in other Python frameworks.

Implement Sessions and Authentication:

What follows is a list of the files that you should create in your application. You should have created some of these for PA1.

Default home page:/ [public]

Contains welcoming message and information about the website. Also has an invitation for new users to join as members. There should be a link to a New User page. There should be some way of getting from this home page to all the public albums of all users.

New User page:`/user` [public]

Contains a form for users to fill in their username, firstname, lastname, e-mail address and password. Make sure the password field does not display text and that there are two password fields for verification.

There are validation rules set forth in Part 4 below that describe the set of permissible passwords. You should use client-side HTML5 validation checking to ensure that the password is "good enough." You should ***not*** use JavaScript to perform this testing.

You should also check that the two password fields match. You should perform this test at both the client- and server-side. For this test ***only*** you can use a small JavaScript function.

If a session already exists redirect the user to `/user/edit`.

Logged in home page:`/` [sensitive]

This page is the home page for a user who has already logged in. Make sure for all pages in a logged in state, you clearly display the message "Logged in as ". This page and all subsequent logged-in pages should have a navigational interface with links to Home (this page), Edit Account, My Albums and Logout. The main body of the page should have a list of all the accessible albums categorized by their owners. Accessible albums include public albums as well as private albums which have been authorized for the current logged in user by the owner.

Edit Account page:`/user/edit` [sensitive]

The user should be able to change his/her firstname, lastname, password and e-mail address (but not username). Again, validate the input values both on the client-side as well as server-side. Also keep a link on this page to delete the user's account. This should remove all associated pictures , albums, as well as access to other users' albums (useful in case the username is recycled and allocated to a new user).

You can perform the delete however you like. However creating a `/user/delete` endpoint which accepts only POST requests may be the easiest, then you can redirect the user back to the public homepage.

My Albums page:`/albums/edit` [sensitive]

This is your `/albums/edit` page from PA1. It allows the user to add new albums, view existing albums, delete them or edit them. Remember that deleting an album should also involve deleting pictures in the album.

Edit Album page:`/album/edit` [sensitive]

At the top of this page the user should be able to change the album name and permissions. There should be some way the user can edit a list of other users to whom he/she would like to give explicit access to view this album, if it is private. You should also list the pictures in the album. Users should be able to delete pictures from the album as well as add new pictures. Users should also be able to click on individual images and be directed to `/pic` from your PA1.

View Album page:`/album` [sensitive/public]

This page displays an album just like the previous assignments. The album title should be at the top, along with the album's owner. The photos should be displayed in sequence order, each with its date, and a caption. The page is the same as in the previous assignment, except that if the album is private, only the logged-in user has permission to view the album. This means the `/album` can be reached either from the logged-in user's homepage or your main page for non-logged in users.

View picture page:`/pic` [sensitive/public]

This page displays a picture just like the previous assignment. It should have the caption, full-sized picture and links to previous and next picture. If the user does not have access to the album this picture is in, they should not be able to see the picture.

Logout page:`/logout` [sensitive]

This should destroy the session and redirect to the default home page.

Part 3: Authentication

Perhaps obviously, authentication issues are present throughout this assignment. In order to view a sensitive page, the user must be authenticated. You should use a form with a username/password that is shipped to a server-side script that tests the information against the database. Logging in should yield one of three situations, each of which should be handled differently.

- Username OR password are invalid: Complain that the user-password combination is invalid.
- Username-password combo is valid: Take the user to the logged in home page.

Part 4: Validation

The new HTML5 specification standardizes many features that are previously common across the web, but were previously implemented in a piecemeal and/or quirky fashion. One such example is client-side validation of forms. Previously implemented using JavaScript, you can now implement client-side form validation with HTML5's built-in support. (See http://www.html5rocks.com/en/tutorials/forms/html5forms/#toc-validation (http://www.html5rocks.com/en/tutorials/forms/html5forms/#toc-validation))

Again: you should *not* use JavaScript for form validation, except to test that multiple fields contain the same value.

You should enforce the following rules:

- The username must be unique (this can only be enforced server-side)

- The username must be at least three characters long
- The username can only have letters, digits and underscores
- The password should be at least 5 and at most 15 characters long
- The password must contain at least one digit and at least one letter
- The password can only have letters, digits and underscores

You can assume that the user is acting in good faith: your goal is to prevent users from adding bad usernames/passwords, not to guard against motivated attackers who want to sneak a strange entry (http://en.wikipedia.org/wiki/Code_injection) into your password database. (which means you do not need to check things beyond above rules)

Grading

We will check the pa2 directory for your new secure photo album website. Based on PA1, your website should contain at least the following users with their albums.

- Username: sportslover, Password: paulpass93 - "I love football" (public), "I love sports" (public)
- Username: traveler, Password: rebeccapass15 - "Around The World"(public)
- Username: spacejunkie, Password: bob1pass - "Cool Space Shots" (private, also accessible to traveler)

Your website may contain other users and albums, but please ensure that the above users and albums exist. Do not touch the files in pa2 sub-directory after the deadline.

As mentioned before, **Remember to commit your code into GitHub and the server, please do not modify your code after the due date - either on the repo or the server**, or else we will assume your submission is late. We then can assess late days or take off points.

Extra Credit

Each correctly implemented extra credit will increase your score by 2%, to a maximum of 6%. The main reason for the extra credit is to provide opportunities for you to challenge yourself. Please take on the extra credit after you have completed the rest of the assignment. Make sure to mention which extra credit features you implemented in your README.md.

- When form is submitted in the New User page, send an e-mail message to the new user confirming his/her membership and redirect them to the logged in home page. (HINT: Check out Mandrill - Ask Guan for help.)
- Ask the user if he/she has forgotten the password and if so, create a new password and e-mail it to them.
- An additional class of "root-user". Anyone who is a root-user can edit anyone's album. There should be an administrative interface for giving/removing the root-user privilege.
- Use CSS to align images in rows and columns (no HTML tables allowed!) and make the website look more appealing - GSI/IAs will award points based on effort and overall

design.