Computer Science 50
Fall 2008
Scribe Notes

Week 8 Wednesday: November 12, 2008
Andrew Sellergren, Anjuli Kannan

Contents

Computer Science 50
Fall 2008
Scribe Notes

Week 8 Wednesday: November 12, 2008
Andrew Sellergren, Anjuli Kannan

1 Announcements (0:00–6:00)

- Apologies about `teh cloud`! As it turns out, a few students' programs were hogging resources by gulping up more and more RAM via `valgrind`. We've since put a cap on the resources which a machine is allowed to use in the hopes that this problem will be prevented in the future. If you're having problems, please e-mail `sysadmins@cs50.net` instead of griping during OHs!
- Final Projects! Send your TF an e-mail by Monday containing your pre-proposal. Don't fret too much about this, just take it as encouragement to get the creative juices flowing. Your TF will be able to refocus your idea and help you narrow it down to a manageable programming task. Note the typo—the relevant e-mail address is `projects@cs50.net`.
- Google's I'm Feeling Lucky button? It costs them $110 million per year, according to this article. This is because the 1% of users who click it actually skip the ad pages.
- Use the Bulletin Board! Your question might already be answered, or, if not, it *will* be answered by a member of the staff or perhaps even a fellow student.
- Only three announcements this week![1]

2 PHP (6:00–47:00)

- Dynamic web pages! Using PHP and SQL in addition to HTML enables us to respond to user's input instead of simply blinking and flashing.
- Check out our `google.html`:

```
<!DOCTYPE html PUBLIC
     "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Fake Google</title>
  </head>
  <body>
    <div align="center">

      <img src="fake.gif" />
```

[1] Those of you who thought to yourself "Wow, I can't believe he didn't change this from last week's notes," congratulations! Click here to claim your prize!

```
        <br /><br />
        <form action="http://www.google.com/search" name="f"
               method="post">
          <input name="hl" type="hidden" value="en">
          <input autocomplete="off" maxlength="2048"
           name="q" size="55" title="Google Search" value="" />
          <br />
          <input name="btnG" type="submit" value="Fake
              Google Search" />
          <input name="btnI" type="submit" value="I'm Feeling
              Lucky">
        </form>

      </div>
    </body>
  </html>
```

What is contained at the URL specified by the action attribute? As you might've guessed, it's not simply an HTML file, but rather a program, perhaps written in Python or PHP. Facebook, the website we all love to hate, is written in large part in PHP.

- How is user input communicated to the server? If we open an extension called Live HTTP Headers in Firefox, we can see how this is done when we execute a Google search.
- Notice that part of the header is the string "`GET /search?hl=en&q=cs50...`" The values after the question mark are known as parameters. One of them, as you can see, is `cs50` which is our search string.
- What happens if we change the URL by adding a `+harvard` after the `cs50`? It's as if we searched for "cs50 harvard." By doing this, we've entirely circumvented the web form.
- Know that there's a reasonable limit on the length of a URL, usually a hundred characters or so, lest a browser not have a buffer large enough to store it.
- There's another type of search submission, however, called `POST` as opposed to `GET`. This enables us to submit a great deal more data to a web page. We specify that we want to use `POST` rather than `GET` by writing `method="post"` in the form tag.
- Also check out the Firefox extension Firebug which enables you to view source code of web pages in a well-organized way.
- Let's look at some more code!

```
<html>

  <head>
    <title>Frosh IMs</title>
  </head>
  <body>
   <form action="register2.php" method="get">
     Name: <input name="name" size="40" />

     <br />
     Dorm: <input name="dorm" size="40" />
     <br />
     Email: <input name="email" size="40" />
     <br />

     <input type="submit" value="Register!" />

   </form>

  </body>

</html>
```

This highly sophisticated web form for Freshman IMs was originally whipped up by David when he was a sophomore intent on teaching himself Perl and saving himself a trip to Wigglesworth to drop off a registration form.

- What happens when we try to access `froshims2.html`? We get a HTTP 403 error, meaning forbidden. We need to run the command `chmod 644` to make it world-readable.
- PHP is an interpreted language, as opposed to a compiled language like C. This means that we don't run our source code through a compiler to get object code, as with C, but rather that we run it through an interpreter in order to execute it. This has performance implications: in general, programs in C and other compiled languages are much faster than programs in PHP and other interpreted languages.
- Every PHP program begins and ends with these tags:

  ```
  <?php

  ?>
  ```

  This alerts the browser that there is code to be executed, not merely displayed.

- PHP files don't need to be world-readable in order to be executed. This is so that your source code don't get ganked![2]
- How do we get at the data the user has submitted? We're going to access the superglobal variables, which are the following:
  - `$_COOKIE`
  - `$_ENV`
  - `$_FILES`
  - `$_GET`
  - `$_POST`
  - `$_REQUEST`
  - `$_SERVER`
  - `$_SESSION`

  Two of these are especially important to us at this stage, namely `$_GET` and `$_POST`. These are arrays filled with the parameters the user has provided via an HTML form.
- If we write `print_r($_GET);` in our `register2.php` file, we get out the user's input. Adding the `<pre>` tag beforehand will display it in a little more readable format.
- What kind of data structure is `$_GET`? We've actually seen it before—a hash table! It simply maps keys to values.
- One useful feature of PHP is that it can be integrated inline with HTML. For example, we can write:

```
<html>
    <body>
        Hello, <?php print($_GET["name"]); ?>.  You live in
        DORM.  Your e-mail is EMAIL.
    </body>
</html>
```

  When we reload the web page, notice that "Hello, David." is printed. If we view the page's source, it is indeed written directly in the HTML. That's because our PHP was interpreted before the page finished loading. Its output was inserted directly into our HTML!
- Now we can write:

[2] i.e. stolen

```
<html>
    <body>
        Hello, <?php print($_GET["name"]); ?>.  You live in
        <?php print($_GET["dorm"]); ?>.  Your e-mail is
        <?php print($_GET["name"]); ?>
    </body>
</html>
```

If we want to make the e-mail address appear as a URL, we can enclose it in the `<a>` tag set the `href` attribute equal to it like so:

```
<html>
    <body>
        Hello, <?php print($_GET["name"]); ?>.  You live in
        <?php print($_GET["dorm"]); ?>.  Your e-mail is <a href="
        mailto:<?php print($_GET["name"]); ?>">e-mail me!!!</a>.
    </body>
</html>
```

- PHP is one of the best languages to learn after C because it's so freakin' easy!!!111 Check out the online manual. One of our hopes is that you'll be able to teach *yourself* new programming languages when you're finished with this course.
- PHP variables are a cinch because they don't have explicit types. Whether you want a string, an int, or a bool, you simply name it with a $ before it. Under the hood, of course, PHP knows the difference between data types.
- PHP also has an entire suite of functions specifically designed to handle arrays!
- If we look at `index.html`, we notice that the following line displays an image from file:

  ```
  <img alt="" src="malanrouge.jpg" />
  ```

- If we want to be extremely annoying, we can display it multiple times using a familiar for loop:

  ```
  <?php for($i = 0; $i < 5; $i++) {

      print '<img alt="" src="malanrouge.jpg" />';
      print "\n";

  }
  ```

- Another feature of PHP is regular expressions, which allow you to check and match patterns. This is particularly useful in validating user input.

- In the `login` directory of this week's source code, you'll find multiple PHP files which implement a website's login functionality in various ways.

3 Two Minute Break (47:00–47:00)

- Hey guys. Big Gulps, huh?[3]

4 More PHP (47:00–60:00

- Check out the source code of `froshims.html` to see how to implement the other types of HTML forms. Notice that it uses (invisible) tables to standardize the page layout.
- How does a web page remember a user? If we take a look at Live HTTP Headers again, we can see a very long string in the Cookie field. PHP will plant on the user's computer a variable called PHPSESSID. This is a pseudo-randomly generated cookie value which uniquely identifies the user. When the user accesses the server again, his browser will send back this cookie to tell the server who it is. This cookie can also be used to save personal information such as usernames, passwords, preferences.
- Take a look at `login1.php` and notice the following line:

  ```
  <form action="<?php echo $_SERVER["PHP_SELF"]; ?>"
        method="post">
  ```

  This is an explicit way of telling the browser to submit the form to itself. Thus, when we submit our login information, the page we're on we'll be reloaded.
- What else does the `$_SERVER` superglobal contain? If we print it out, we can see the user's IP address, browser type, and a whole lot more.
- How do we enable sessions in PHP? A simple call to `session_start()` will do!
- As we keep walking through `login1.php` one line a time, we find a hard-coded username and password followed by two calls to the function `isset()`. As you might've guessed, `isset()` simply checks that a variable has a value. Once we've verified that the "user" and "pass" fields weren't empty when submitted, we compare them to the hard-coded username and password to validate them.
- If we validate our user, we can add a simple boolean to our `$_SESSION` global like so:

  ```
  $_SESSION["Authenticated"] = TRUE;
  ```

[3] Welp, see ya later!

That's all it takes to know whether a user is logged in or not.

- Once we have validated a user's login, we can redirect him to our home page using these lines of code:

```
// redirect user to home page, using absolute path
$host  = $_SERVER['HTTP_HOST'];
$path  = rtrim(dirname($_SERVER['PHP_SELF']), '/\\');
header("Location: http://$host$uri/home.php");
exit;
```

- When we sniff the web traffic using Live HTTP Headers, we notice that the server returns HTTP 302 Moved Temporarily. This is because we issued it the redirect request, as detailed above.

- Notice that we have no `else` condition on our login validation, so if the user does not login successfully, the rest of the code of `login1.php` executes. As it's written, the user will get kicked back to the login page and the message "INVALID LOGIN" will display. What condition do we check before displaying this message? In fact, we check `count($_POST) > 0`. In other words, if the user got to this point in our code after clicking the Submit button, then he must have provided an invalid login (because a successful login will redirect and then exit). Therefore, if there is anything at all in the `$_POST` global array, then we can be sure that the login was invalid and we can display an appropriate message.

- Obviously, each time a user registers at your website, you're not going to hard-code their username and password into your PHP files. More practically, you'll want to do this by interacting with a database. The language of databases is SQL.

5 SQL (60:00–77:00)

- In order to interact with your MySQL database for the next problem set, we've installed DirectAdmin (thanks to a donation) which provides a convenient web panel. All you'll have to do to administer your database, then, is open up your browser and login to the web-based interface.

- If we click Create New Database, we can name it whatever we want and also add a new user and password entirely through the web-based interface, which generates and executes the SQL automatically for us. Another such tool for accomplishing this is phpMyAdmin, which is freely available.

- Don't worry, Problem Set 7's specification will walk you through this!

- Oracle, SQL Server, and MySQL are all *relational* database engines. The word *relational* merely implies that they use tables which are linked to one another via one or more fields.

- Our goal is to improve on the hard-coding of usernames and passwords. Not too hard! Let's create a table called `users` which contains two columns: `user` and `pass`. For each of the columns, we have to specify a data type. Let's keep it simple: we'll declare them as `VARCHAR` and specify their(maximum) Length/Values. Let's also set them to "not null," meaning exactly what you think it means.

- In relational databases, there exists the concept of *primary keys*, fields which by definition are unique. Having primary keys on the tables in our database will greatly improve lookup time. In this case, it makes sense to specify the username as our primary key because we don't want the same username to map to multiple different passwords. It wouldn't make sense to specify the `pass` field as unique because it would prevent two users from having the same password.

- For performance reasons, you can also specify a field as an *index* or as *unique*, which like primary keys, are optimized for lookup.

- If we click Save, a SQL query will be displayed. This is the code that phpMyAdmin has automatically generated and executed for us. Back in the day,[4] these SQL commands were executed from the command line.

- Now that we have a database running on the server, let's connect to it using PHP. Take a look at `login5.php` and notice that it contains code almost identical to `login1.php` but with these lines, among others, inserted:

```
// connect to database
if (($connection = mysql_connect("localhost",
                                 "malan_real",
                                 "12345")) === FALSE)
    die("Could not connect to database");

// select database
if (mysql_select_db("malan_real", $connection) === FALSE)
    die("Could not select database");
```

  The server name is `localhost` because the database is on the same server as the web server. Only by coincidence is the username the same as the database name, `malan_real`. The login validation is almost identical to `login1.php`, with the exception, at least, of the following line:

```
$sql = sprintf("SELECT * FROM users WHERE user='%s'",
               mysql_real_escape_string($_POST["user"]));
```

[4] When the only computer in town was as big as a barn and you had to walk 3 miles in the snow, uphill both ways, just to wait for 5 hours in line to use it. And it used punchcards. This is the era known as David's "youth" or "heyday."

What we're doing here is querying the database for a row in the `users` table which contains the submitted username. The call to `mysql_real_escape_string` prevents SQL injection attacks.[5] With SQL, queries of the following types, among others, are possible:

- `SELECT`
- `INSERT`
- `UPDATE`
- `DELETE`

In this case, we want to use `SELECT` because our goal is to look up the submitted username. The `*` indicates that we want to select all columns for the specified row. After we add a user `jharvard` with password `12345` to the `users` table using phpMyAdmin's web interface, the query below will return 1 row:

```
SELECT * FROM users WHERE username='jharvard'
```

Note that the capitalization is merely for clarity's sake. SQL is case-insensitive. Now, back to the original PHP code:

```
$sql = sprintf("SELECT * FROM users WHERE user='%s'",
               mysql_real_escape_string($_POST["user"]));
```

As you did in pset5, we're using `sprintf` to insert the submitted username into this string which looks like a SQL query. Finally, we're ready to execute the query:

```
// execute query
$result = mysql_query($sql);
if ($result === FALSE)
    die("Could not query database");

// check whether we found a row
if(mysql_num_rows($result) == 1) {

  // fetch row
    $row = mysql_fetch_assoc($result);

    // check password
    if($row["pass"] == $_POST["pass"]) {

        // remember that user's logged in
        $_SESSION["authenticated"] = TRUE);
```

[5] Like this.

```
        ...
    }
}
```

First, we check if any rows were returned by our SQL query. If none were, then the user is not in our `users` table. If a single row was returned, then we're going to fetch it and check that the associated password matches the password provided by the user. If it does, we remember that the user is logged in by setting a `bool` in `$_SESSION` redirect to the homepage as before.

- Know that MySQL comes with its own datatypes and commands which are well-documented online.
- How might you go about implementing an e-trade website? You'll need a `users` table, for starters. Then you'll also need to remember a stock ticker symbol, a quantity, and a time for every stock that a user has bought. If you want to merge data from two different tables, you'll need to get familiar with the SQL command `JOIN`.
- We might maintain two different tables for employees and orders because we don't want to repeat certain information like an employee's name for every single order. For efficiency's sake, we split up this information, but when we want to access that information, we need to join it back up. The query below will accomplish just this:

```
SELECT Employees.Name, Orders.Product
FROM Employees, Orders
WHERE Employees.Employee_ID=Orders.Employee_ID
```

- For Problem Set 7, you'll be tasked with all of this and more in implementing a stock-buying website. Anything that wasn't covered in lecture is sure to be covered in the specification, so read up!