Computer Science 75
Fall 2009
Scribe Notes

Lecture 6: October 26, 2009
Andrew Sellergren

Contents

Computer Science 75
Fall 2009
Scribe Notes

Lecture 6: October 26, 2009
Andrew Sellergren

1 Announcements (0:00–2:00)

- Hung Tran is in the lead on The Big Board! His success may be due to some savvy trading or perhaps just after-hours trading, but in either case, good work! See if you can catch him!

2 SQL (2:00–60:00)

2.1 Interacting with Databases (cont'd)

2.1.1 Via PHP (cont'd)

- We left off last time with `login6.php`, in which we used the `SELECT 1` trick to verify a user's escaped login credentials against our database. As with `login5.php`, we then checked if the result set contained exactly one row, which would mean that the login credentials were found in our database.
- We can test our query from within the phpMyAdmin interface by clicking the SQL tab and hardcoding in a username and password like so:

  ```
  SELECT 1 FROM users WHERE user = 'jharvard' AND pass = '12345';
  ```

  Once we click Go, we get an empty result set. Oops, wrong password. Let's try it again with the correct password:

  ```
  SELECT 1 FROM users WHERE user = 'jharvard' AND pass = 'crimson';
  ```

  Now we get a temporary table returned which has a single row containing the single value 1. Notice that in the previous case, when the credentials weren't found, we got no results. This confirms our logic in `login6.php`.
- So in the case above when we forgot the password for `jharvard`, we were able to quickly look it up in our database. In the real world, in most cases, this won't be possible. Even if you provide all the correct identifying information to an IT help desk, the best they'll be able to do is to reset your password. That's because passwords aren't generally stored as plaintext, but rather as encrypted text. This encryption is one-way for security reasons. If a malicious user were ever to crack the database containing these passwords, he would be unable to use any of them because he wouldn't be able to undo this one-way encryption.
- Question: don't you need to decrypt the password when verifying a user's credentials? No, in fact, because you can encrypt the user's input using the same one-way algorithm. If the results of that encryption matches the encrypted password stored in the database, then the user can be logged in. There is, however, some insignificant probability (say, 1 in 4 billion or so) that two plaintext inputs will produce the same encrypted output.

- Let's take a look at `login7.php` to see how we implement this password encryption:

```
<?
    /**
     * login7.php
     *
     * A simple login module that checks a username and password
     * against a MySQL table with weak encryption (well, a weak hash).
     *
     * David J. Malan
     * Computer Science E-75
     * Harvard Extension School
     */

    // enable sessions
    session_start();

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

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

    // if username and password were submitted, check them
    if (isset($_POST["user"]) && isset($_POST["pass"]))
    {
        // prepare SQL
        $sql = sprintf("SELECT 1 FROM users
                        WHERE user='%s' AND pass=PASSWORD('%s')",
                        mysql_real_escape_string($_POST["user"]),
                        mysql_real_escape_string($_POST["pass"]));

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

```
            // redirect user to home page, using absolute path, per
            // http://us2.php.net/manual/en/function.header.php
            $host = $_SERVER["HTTP_HOST"];
            $path = rtrim(dirname($_SERVER["PHP_SELF"]), "/\\");
            header("Location: http://$host$path/home.php");
            exit;
        }
    }
?>

<!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>Log In</title>
  </head>
  <body>
    <form action="<? echo $_SERVER["PHP_SELF"]; ?>" method="post">
      <table>
        <tr>
          <td>Username:</td>
          <td>
            <input name="user" type="text" /></td>
        </tr>
        <tr>
          <td>Password:</td>
          <td><input name="pass" type="password" /></td>
        </tr>
        <tr>
          <td></td>
          <td><input type="submit" value="Log In" /></td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

The only difference between `login7.php` and previous versions is the invocation of the `PASSWORD` function in SQL. Let's examine in phpMyAdmin what the output of this function might look like. If we type `SELECT PASSWORD('crimson')`, we'll get the following output:

```
*02A501BC718BBD7927FC5805D858811E3C3F1825
```

This output must match what's stored in our database in order for us to authenticate the user. But this value shouldn't be stored in our database

as a `VARCHAR`, but rather as a `BLOB`, or binary large object. Once we've changed the data type of our `pass` field, we can execute the following query:

```
INSERT INTO users (user, pass)
            VALUES('jharvard', PASSWORD('crimson'));
```

- Although it's a useful starting point in learning encryption, `PASSWORD`, is not a very strong algorithm and can be pretty easily cracked. The AES (advanced encryption standard) algorithm is much stronger. Thankfully, MySQL provides support for this algorithm as well, as we'll see in `login8.php`:

```
<?
    /**
     * login8.php
     *
     * A simple login module that checks a username and password
     * against a MySQL table with strong encryption (but insecure secret).
     *
     * David J. Malan
     * Computer Science E-75
     * Harvard Extension School
     */

    // enable sessions
    session_start();

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

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

    // if username and password were submitted, check them
    if (isset($_POST["user"]) && isset($_POST["pass"]))
    {
        // prepare SQL
        $sql = sprintf("SELECT 1 FROM users
                        WHERE user='%s' AND pass=AES_ENCRYPT('%s', 'secret')",
                        mysql_real_escape_string($_POST["user"]),
                        mysql_real_escape_string($_POST["pass"]));

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

                // redirect user to home page, using absolute path, per
                // http://us2.php.net/manual/en/function.header.php
                $host = $_SERVER["HTTP_HOST"];
                $path = rtrim(dirname($_SERVER["PHP_SELF"]), "/\\");
                header("Location: http://$host$path/home.php");
                exit;
            }
        }
    ?>

    <!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>Log In</title>
      </head>
      <body>
        <form action="<? echo $_SERVER["PHP_SELF"]; ?>" method="post">
          <table>
            <tr>
              <td>Username:</td>
              <td>
                <input name="user" type="text" /></td>
            </tr>
            <tr>
              <td>Password:</td>
              <td><input name="pass" type="password" /></td>
            </tr>
            <tr>
              <td></td>
              <td><input type="submit" value="Log In" /></td>
            </tr>
          </table>
        </form>
      </body>
```

```
</html>
```

In addition to `AES_ENCRYPT`, there is a MySQL function named `AES_DECRYPT`, which implies that AES is actually a two-way encryption algorithm. Unlike `PASSWORD`, the results of `AES_ENCRYPT` can actually be undone.[1] Two arguments are supplied to `AES_ENCRYPT`: the string to be encrypted and the secret key by which it will be encrypted. In this example, we hard-code in the value `secret` as our secret key. Well, now we have a stronger cryptographic algorithm, but if a malicious user steals our database and our PHP source code, he can decrypt all the passwords in the database.

- One solution to this quandary is to use the user's password as both the string to encrypt and the secret key by which it will be encrypted—in other words, pass it as both arguments to `AES_ENCRYPT`. What's the catch? Well, now we can't decrypt passwords because there's a Catch-22: we need the user to provide the correct password so that we can decrypt his password. In the real world, as we've seen, we don't really need this ability to decrypt passwords since we can simply reset them. In the world of theory, encrypting a password with itself prevents us from making strong claims about the uniqueness of the encrypted output, but for the most part, we can wave our hands at this.
- Another concern that we need to consider is that if a malicious user gains access to the database and scans the list of passwords, he will be able to notice if two users have the same encrypted password. Knowing this, he'll have more information toward guessing the plaintext password. To prevent this, we might perturb the encryption algorithm, so to speak, by adding an extra piece to the secret key—perhaps concatenate the username with the password to create the secret key.
- What we've been talking so far tonight is one-factor authentication, meaning you only have to know a single password to login. One step above this is two-factor authentication, which involves not only a password but a pseudorandom number generator which you carry on your person. This number generator creates a pseudorandom number several times per minute in sync with the server. So when you go to login, you provide your password as well as the current number on the generator.

2.2 Data Types

- MySQL has numerous different data types. For the purposes of this course, you probably won't notice a difference in performance even if you declare all your fields as strings. However, in general, you should aim to constrain your data types as much as possible, both to save space and to prevent invalid data.

[1] Although encryption methods generally tend to implement randomness to increase security, we can't be truly random because we need to be able to recreate that randomness in order to authenticate users.

- For most text input, `VARCHAR` is probably the best option. It has a maximum length of 65,536 characters and it will only use as many as necessary. Space-wise, this is advantageous. Performance-wise, however, there is an advantage to fixed length, as `CHAR` provides, for example, because the engine can optimize how it accesses the data if it knows with certainty how the data is laid out in memory. Longer text fields such as `TEXT` or `LONGTEXT` are useful when you want to store something like user comments or news articles. HarvardNews, for example, stores articles in `TEXT` columns because it supports fulltext search.

- MySQL offers numerous date data types, including `DATE`, `DATETIME`, `YEAR`, `TIME`, and `TIMESTAMP`. The first four might all be used to record when an order was placed or an article was posted, for example. The last is especially useful for recording when an update was made to the database. If you use the data type `TIMESTAMP`, you can specify `ON UPDATE CURRENT_TIMESTAMP` as an attribute, which will record the time when the row was updated.

- In terms of numbers, all the different variations of integers are available, as well as `FLOAT` and `DOUBLE`. One extra number data type worth mentioning is `DECIMAL`, which allows you to specify how many places to the left and to the right of the decimal point you'd like to store. This is especially useful when you're dealing with money (think Project 2) when tiny imprecisions might lead to glaring errors (think Superman 3 and Office Space).

- The binary data types, e.g. variations of `BLOB`, allow you to store binary data from password encryption algorithms or even the raw stuff of graphics. Although there are arguments that go both ways, generally it's not recommended to store full-fledged graphics in a database. You're better off storing a filename or path in the database and the actual graphics file on the file system.

- One other data type worth mentioning: `ENUM` allows you to specify exactly what values a field might take. For example, if you know a field will *only* take values 'foo', 'bar', or 'baz', then you can enumerate them and their storage will be optimized. This also helps with data validation.

- Question: is it possible to have a field take on a value only when it is first inserted and not when it is updated? Yes, using the `DEFAULT` keyword.

- To be on the safe side, you should probably use PHP's `sprintf` to prepare numbers or strings for insertion into a `DECIMAL` field so that you can be sure the right number of decimal places are filled.

2.3 Real World Applications: Faculty Lunch

- Last year, the computer science faculty for the engineering school would gather every week for lunch catered by Rebecca's Cafe. Previously, the process for ordering lunch consisted of e-mailing a staff member with a

particular request. David felt that twenty-some tech-savvy faculty ordering sandwiches in this tedious, non-automated fashion was a little too ironic. So he decided to implement an online ordering system which would streamline the process. Interestingly, he found that copying and pasting the menu into XML format was probably the fastest way of converting the menu to machine-readable format. Check out the production version of the site.

- To break it down into more manageable chunks, we can examine the development version of the site:

```
<?
    $xml = new SimpleXMLElement(file_get_contents("menu.xml"));
?>

<!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>Lunch</title>
  </head>
  <body>
    <form action="sqlite.php" method="post">
      <b>Name:</b> <input name="name" type="text" />
          <input type="submit" value="Submit Order" />
      <br /><br />
      <table border="0">
        <? foreach ($xml->xpath("/menu/category[@name='Specialty Sandwiches']/item")
            as $item): ?>
          <tr>
            <td valign="top"><input id="<?= $item["name"] ?>"
                name="item" type="radio" value="<?= $item["name"] ?>" /></td>
            <td>
              <label for="<?= $item["name"] ?>">
                <b><?= $item["name"] ?></b>
                <br />
                <?= $item ?>
              </label>
            </td>
          </tr>
        <? endforeach ?>
      </table>
    </form>
  </body>
```

```
</html>
```

Here, we see that we're using a little bit of XPath to iterate over the Specialty Sandwiches category of the menu and to print out radio buttons for each. Now let's take a look at `sqlite.php`, which is the page that this form submits to:

```
<?

    // ensure complete form was submitted
    if (!isset($_POST["name"]) || !isset($_POST["item"]))
    {
        header("Location: http://www.cs75.net/lectures/5/src/lunch/lunch.php");
        exit;
    }

    try
    {
        // open database
        $dbh = new PDO("sqlite:orders.db");
        $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        // prepare fields
        $name = $dbh->quote($_POST["name"]);
        $item = $dbh->quote($_POST["item"]);

        // insert order
        $dbh->exec("INSERT INTO orders (name, item) VALUES($name, $item)");
    }
    catch (PDOException $e)
    {
        die($e->getMessage());
    }

?>

<!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>Lunch</title>
  </head>
  <body>
    One <?= $_POST["item"] ?> for <?= $_POST["name"] ?>, coming right up!
```

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

SQLite is useful for lightweight applications like this. To use it, we'll begin by instantiating an object of type `PDO` (portable database object). Then we call its `quote` method in order to escape our input and eventually we call its `exec` method in order to execute a SQL query. What's nice about the PDO API is that if you ever decide to change database engines, you need only change the initial instantiation of the PDO object and you're good to go (as long as you don't use engine-specific syntax).

- As an aside, if we wanted to write the orders out to a CSV file instead of a SQLite database, we might submit to a file like `csv.php`:

```
<?

    // ensure complete form was submitted
    if (!isset($_POST["name"]) || !isset($_POST["item"]))
    {
        header("Location: http://www.cs75.net/lectures/5/src/lunch/lunch.php");
        exit;
    }

    // open CSV file for appending
    $handle = fopen("orders.csv", "a");

    // acquire exclusive lock
    flock($handle, LOCK_EX);

    // add order to CSV file
    $order = array($_POST["name"], $_POST["item"]);
    fputcsv($handle, $order);
    fclose($handle);

?>

<!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>Lunch</title>
  </head>
  <body>
    One <?= $_POST["item"] ?> for <?= $_POST["name"] ?>, coming right up!
```

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

In this file, we open a CSV file (in append mode) with an exclusive lock (meaning only one user can write to it at a time) and then call the `fputcsv()` function to convert a PHP array into comma-separated values.

- Similarly, if we wanted to write orders to an XML file, we might submit to a file like `xml.php`:

```
<?

    // ensure complete form was submitted
    if (!isset($_POST["name"]) || !isset($_POST["item"]))
    {
        header("Location: http://www.cs75.net/lectures/5/src/lunch/lunch.php");
        exit;
    }

    // open XML file for reading + writing
    $handle = fopen("orders.xml", "r+");

    // acquire exclusive lock
    flock($handle, LOCK_EX);

    // read contents of XML file
    $contents = fread($handle, filesize("orders.xml"));

    // build DOM out of contents
    $xml = new SimpleXMLElement($contents);

    // add order
    $order = $xml->addChild("order");
    $order->addChild("name", $_POST["name"]);
    $order->addChild("item", $_POST["item"]);

    // overwrite original XML file
    rewind($handle);
    fwrite($handle, $xml->asXML());
    fclose($handle);
?>

<!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>Lunch</title>
  </head>
  <body>
    One <?= $_POST["item"] ?> for <?= $_POST["name"] ?>, coming right up!
  </body>
</html>
```

Here, we again acquire a file lock using the `flock()` function and we use the `fread()` function rather than the `file_get_contents()` function in order to retain that lock. Then we go through the rather tedious process of adding a child to the XML file, where appropriate.

- One other quick aside: the `DATE_FORMAT` and `TIME_FORMAT` functions in MySQL will format your date and time strings in a much more human-readable fashion.

2.4 `JOIN`

- We mentioned earlier that redundancy in databases is generally discouraged because it wastes space. At the same time, if data needs to be looked up in a separate table before it can be meaningful to a human, then our database might suffer in terms of performance. Thankfully, MySQL supports the `JOIN` operation, which allows us to combine multiple tables.
- Imagine that we have a number of employees, each of whom might order some office supplies. If we want to keep track of who orders what office supplies, we're better off assigning an ID number to each employee so that we can keep track of their orders by ID number (represented as an `INT`) rather than their full name. But ultimately, when we want to view this data, we want to see the employee's full name since the ID number means nothing to a human. So we need the `JOIN` operation. In some cases, however, this `JOIN` might be implicit:

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

  This `JOIN` is implicit because we never actually invoke the keyword. Note that we specify the table by naming it before the period when we name a certain column.

- More readable is an explicit `JOIN`:

  ```
  SELECT Employees.Name, Orders.Product
  FROM Employees
  JOIN Orders ON Employees.Employee_ID=Orders.Employee_ID
  ```

2.5 Race Conditions

- Let's say you're in college and you and your roommate both desperately depend on having milk. If either one of you comes back to the dorm, opens the refrigerator, and finds no milk, he will immediately go out a store and buy more. But what if you both come back at separate times and find no milk and you happen to go to separate stores. Then you might both come back with a bottle of milk and thus your room will have too much.
- In the real world, you might solve this problem by leaving a note for your roommate. In the world of programming, you might actually put a padlock on the refrigerator. Now your roommate can't even open the refrigerator to decide whether he needs to buy milk or not.
- Let's imagine that we're designing a schema for Project 2. We might have a `users` table that has fields for `username`, `password`, and `cash`. Then we'll have a separate `portfolios` table that has fields for `username`, `symbol`, and `shares`. That way, we can store all the different stocks that a given user might own, whether it be 1 or 500+. If we wanted to optimize even further, we might add a `uid` field to the `users` table, that way we wouldn't have to repeat something so long as a username for the `portfolios` table.
- Let's say we are a single user who initiates two stock purchases in two different browser windows. If one of the browser windows decides to sleep after checking the user's balance and storing it in a variable, the other browser might subtract the sale value from the wrong balance. This would only be a problem if we use the MyISAM storage engine, however, since with InnoDB we can implement row-level locking. Even with MyISAM we can implement table-locking:

  ```
  LOCK TABLES account WRITE;
  SELECT balance FROM account WHERE number = 2;
  UPDATE account SET balance = 1500 WHERE number = 2;
  UNLOCK TABLES;
  ```

  In this case, we want to make sure that the `SELECT` and `UPDATE` operations don't execute separately. Instead of using table-locking, we might use row-level locking using InnoDB:

  ```
  START TRANSACTION;
  UPDATE account SET balance = balance - 1000 WHERE number = 2;
  UPDATE account SET balance = balance + 1000 WHERE number = 1;
  SELECT balance FROM account WHERE number = 2;
  # suppose account # 2 has a negative balance!
  ROLLBACK;
  ```

  Transactions ensure that all the SQL statements therein are executed without interruption.

2.6 C$75 Finance

- In terms of registration, you're going to want to perform some validation and then insert a username and password into your database if everything checks out.
- Once a user is logged in, you should offer him the ability to look up a stock quote. You won't, of course, have to store all the stock data yourself, but can query Yahoo as needed in order to obtain the most recent stock info.
- In terms of the sell functionality, we only ask that you be able to liquidate all of a user's stock at once. In other words, they should be able to sell all shares of a particular stock at once. Thus, you need only create a simple hyperlink if you so desire.