Lecture outline

- PHP/SQL review
- some basic web attacks
- breaking and securing an example page
Recall: PHP MySQL functions

- `mysql_connect("server", "username", "password")` connects to the given server; returns FALSE on failure
- `mysql_select_db("database")` chooses the given database; returns FALSE if not found
- `mysql_query("query")` executes the given SQL query on the currently selected database; returns a result-set object, or FALSE if query fails
- `mysql_fetch_array(results)` returns one row from the given query result set as an associative array, or FALSE when no more rows remain
- `mysql_error()` returns a string representing the most recent MySQL-related error that has occurred

Complete PHP MySQL example

```php
# connect to world database on local computer
$db = mysql_connect("localhost", "traveler", "packmybags");
mysql_select_db("world");

# execute a SQL query on the database
$results = mysql_query("SELECT * FROM Countries WHERE population > 100000000

# loop through each country
while ($row = mysql_fetch_array($results)) {
} <?php
```
Complete example w/ error checking

# connect to world database on local computer
$db = mysql_connect("localhost", "traveler", "packmybags");
check_result($db);
check_result(mysql_select_db("world"));

# execute a SQL query on the database
$results = mysql_query("SELECT * FROM Countries WHERE population > 100000000");
check_result($results);

# loop through each country
while ($row = mysql_fetch_array($results)) {
    <li>{$row["name"]}, ruled by {$row["head_of_state"]}</li>
} />

# stops the page if any MySQL error occurred
function check_result($value) {
    if (!$value) {
        die("SQL error occurred: " . mysql_error());
    }
}

Simpsons database w/ passwords

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>email</th>
<th>password</th>
<th>id</th>
<th>name</th>
<th>teacher_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Bart</td>
<td><a href="mailto:bart@fox.com">bart@fox.com</a></td>
<td>bartman</td>
<td>10001</td>
<td>Computer Science 142</td>
<td>1234</td>
</tr>
<tr>
<td>404</td>
<td>Ralph</td>
<td><a href="mailto:ralph@fox.com">ralph@fox.com</a></td>
<td>catfood</td>
<td>10002</td>
<td>Computer Science 143</td>
<td>5678</td>
</tr>
<tr>
<td>456</td>
<td>Milhouse</td>
<td><a href="mailto:milhouse@fox.com">milhouse@fox.com</a></td>
<td>fallout</td>
<td>10003</td>
<td>Computer Science 190M</td>
<td>9012</td>
</tr>
<tr>
<td>888</td>
<td>Lisa</td>
<td><a href="mailto:lisa@fox.com">lisa@fox.com</a></td>
<td>vegan</td>
<td>10004</td>
<td>Informatics 100</td>
<td>1234</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>Krabappel</td>
</tr>
<tr>
<td>5678</td>
<td>Hoover</td>
</tr>
<tr>
<td>9012</td>
<td>Stepp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>10001</td>
<td>B-</td>
</tr>
<tr>
<td>123</td>
<td>10002</td>
<td>C</td>
</tr>
<tr>
<td>456</td>
<td>10001</td>
<td>B+</td>
</tr>
<tr>
<td>888</td>
<td>10002</td>
<td>A+</td>
</tr>
<tr>
<td>888</td>
<td>10003</td>
<td>A+</td>
</tr>
<tr>
<td>404</td>
<td>10004</td>
<td>D+</td>
</tr>
</tbody>
</table>
Web Security
breaking and securing web pages

CSE <= 190M

- until now, we have assumed:
  - valid user input
  - non-malicious users
  - nothing will ever go wrong
- this is unrealistic!

The real world

- in order to write secure code, we must assume:
  - invalid input
  - evil users
  - everybody is out to get you
- trust nothing
**HTML injection**

*a flaw where a user is able to inject arbitrary HTML content into your page*

- why is this bad? it allows others to:
  - disrupt the flow/layout of your site
  - put words into your mouth
  - (possibly) run JavaScript on other users’ computers
- kinds of injected content:
  - annoying: `results.php?name=<blink>lololol</blink>`
  - malicious and harmful: `onlinebanking.php?text=<script>transferMoneyTo("Evil Kevin", 1000, "USD");</script>`
    - injecting JavaScript content is called **cross-site scripting**

**Securing against HTML injection**

- one idea: disallow harmful characters
  - HTML injection is impossible without `< >`
  - can strip those characters from incoming input
  - or, just reject the entire request if they are present
- better idea: allow them, but **escape** them
  - `< >` → `&lt; &gt;`
  - PHP's `htmlspecialchars` function escapes HTML characters:
    ```php
    $username = htmlspecialchars($_REQUEST['username']);
    ```

**SQL injection**

*a flaw where the user is able to inject arbitrary SQL commands into your query*

- `$query = "SELECT name, ssn, dob FROM users
  WHERE username = '$username' AND password = '$password'";
- Password: `[OR '1'='1]`
- `$query = "SELECT name, ssn, dob FROM users
  WHERE username = '$username' AND password = ' OR '1'='1'";
- What will the above query return? Why is this bad?
Securing against SQL injection

- similar to securing against HTML injection, escape the string before you include it in your query
- use the PHP `mysql_real_escape_string` function

```php
$username = mysql_real_escape_string($_REQUEST['username']);
$password = mysql_real_escape_string($_REQUEST['password']);
$query = "SELECT name, ssn, dob FROM users WHERE username = '{$username}' AND password = '{$password}';";
```