فصل دوم

فرم های PHP

A Simple HTML Form

- The example below displays a simple HTML form with two input fields and a submit button:

```
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

A Simple HTML Form

- When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.
- To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

```
<html>
<body>

Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>

</body>
</html>
```

A Simple HTML Form

- The output could be something like this:

```
Welcome John
Your email address is john.doe@example.com
```

A Simple HTML Form

- The same result could also be achieved using the HTTP GET method:

```
<html>
<body>

<form action="welcome_get.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

A Simple HTML Form

- and "welcome_get.php" looks like this:

```
<html>
<body>

Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>

</body>
</html>
```

- The code above is quite simple. However, the most important thing is missing. You need to validate form data to protect your script from malicious code.

⚠ Think SECURITY when processing PHP forms!

This page does not contain any form validation, it just shows how you can send and retrieve form data.

GET vs. POST

- Both GET and POST create an array (e.g. array(key1 => value1, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.
- Both GET and POST are treated as $_GET and $_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.
- $_GET is an array of variables passed to the current script via the URL parameters.
- $_POST is an array of variables passed to the current script via the HTTP POST method.

When to use GET?

- Information sent from a form with the GET method is visible to everyone (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.
- GET may be used for sending non-sensitive data.

⚠ GET should NEVER be used for sending passwords or other sensitive information!

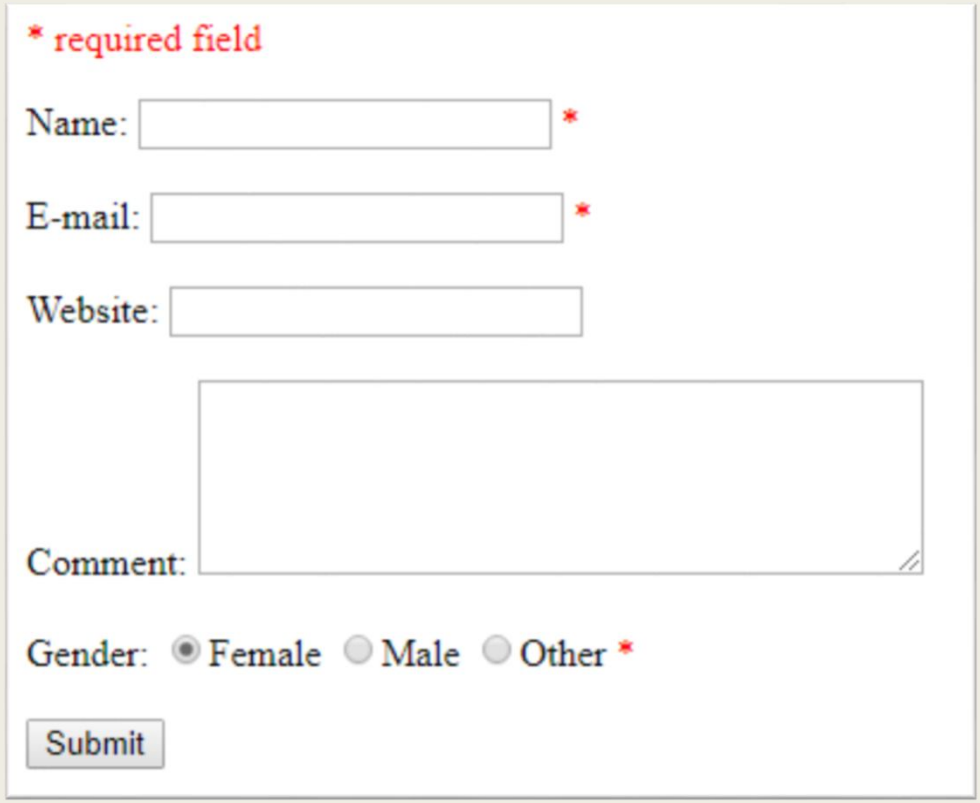
When to use POST?

- Information sent from a form with the POST method is invisible to others (all names/values are embedded within the body of the HTTP request) and has no limits on the amount of information to send.
- Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.
- However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

⚠ Developers prefer POST for sending form data.

PHP Form Validation

- The HTML form we will be working at in these chapters, contains various input fields: required and optional text fields, radio buttons, and a submit button:



PHP Form Validation

- The validation rules for the form above are as follows:

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

PHP Form Validation

- Text Fields
  - The name, email, and website fields are text input elements, and the comment field is a textarea. The HTML code looks like this:

```
Name: <input type="text" name="name">
E-mail: <input type="text" name="email">
Website: <input type="text" name="website">
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
```

PHP Form Validation

- Radio Buttons
  - The gender fields are radio buttons and the HTML code looks like this:

```
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<input type="radio" name="gender" value="other">Other
```

PHP Form Validation

- The Form Element
  - The HTML code of the form looks like this:

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

  - When the form is submitted, the form data is sent with method="post".

⚠ The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.

  - So, the `$_SERVER["PHP_SELF"]` sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

⚠ The `htmlspecialchars()` function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with < and >. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

PHP Form Validation-Form Security

- The $_SERVER["PHP_SELF"] variable can be used by hackers!
- If PHP_SELF is used in your page then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute.

⚠ Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users.

PHP Form Validation-Form Security

- Assume we have the following form in a page named "test_form.php":

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

- Now, if a user enters the normal URL in the address bar like "http://www.example.com/test_form.php", the above code will be translated to:

```
<form method="post" action="test_form.php">
```

- So far, so good.
- However, consider that a user enters the following URL in the address bar:

PHP Form Validation-Form Security

- However, consider that a user enters the following URL in the address bar:

```
http://www.example.com/test_form.php/%22%3E%3Cscript%3E
alert('hacked')%3C/script%3E
```

- In this case, the above code will be translated to:

```
<form method="post" action="test_form.php/">
<script>alert('hacked')</script>
```

- This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP_SELF variable can be exploited.

Avoid $_SERVER["PHP_SELF"] Exploits

- $_SERVER["PHP_SELF"] exploits can be avoided by using the htmlspecialchars() function. The form code should look like this:

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

- The htmlspecialchars() function converts special characters to HTML entities. Now if the user tries to exploit the PHP_SELF variable, it will result in the following output:

```
<form method="post" action="test_form.php/&quot;&gt;&lt;scri
pt&gt;alert('hacked')&lt;/script&gt;">
```

- The exploit attempt fails, and no harm is done!

Validate Form Data With PHP

- The first thing we will do is to pass all variables through PHP's htmlspecialchars() function.
- When we use the htmlspecialchars() function; then if a user tries to submit the following in a text field:

  <script>location.href('http://www.hacked.com')</script>

- - this would not be executed, because it would be saved as HTML escaped code, like this:

  <script>location.href('http://www.hacked.com')</script>

- The code is now safe to be displayed on a page or inside an e-mail.

Validate Form Data With PHP

- We will also do two more things when the user submits the form:
  1. Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
  2. Remove backslashes (\) from the user input data (with the PHP stripslashes() function)
- The next step is to create a function that will do all the checking for us (which is much more convenient than writing the same code over and over again).
- We will name the function test_input().

Validate Form Data With PHP

- Now, we can check each $_POST variable with the test_input() function, and the script looks like this:

```php
<?php // define variables and set to empty values
$name = $email = $gender = $comment = $website = "";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}
function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
```

Validate Form Data With PHP

- Notice that at the start of the script, we check whether the form has been submitted using `$_SERVER["REQUEST_METHOD"]`. If the REQUEST_METHOD is POST, then the form has been submitted - and it should be validated. If it has not been submitted, skip the validation and display a blank form.
- However, in the example above, all input fields are optional. The script works fine even if the user does not enter any data.

PHP Forms - Required Fields

- From the validation rules table on the previous page, we see that the "Name", "E-mail", and "Gender" fields are required. These fields cannot be empty and must be filled out in the HTML form.

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

PHP Forms - Required Fields

- In the following code we have added some new variables: $nameErr, $emailErr, $genderErr, and $websiteErr. These error variables will hold error messages for the required fields. We have also added an if else statement for each $_POST variable. This checks if the $_POST variable is empty (with the PHP empty() function). If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the test_input() function:

PHP Forms - Required Fields

```php
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
  }
  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
  }
```

PHP Forms - Required Fields

```
if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
  }
  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }
  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}
?>
```

PHP Forms - Display The Error Messages

- Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields):

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

Name: <input type="text" name="name">
<span class="error">* <?php echo $nameErr;?></span>
<br><br>
E-mail:
<input type="text" name="email">
<span class="error">* <?php echo $emailErr;?></span>
<br><br>
```

PHP Forms - Display The Error Messages

```
Website:
<input type="text" name="website">
<span class="error"><?php echo $websiteErr;?></span>
<br><br>
Comment: <textarea name="comment" rows="5" cols="40"><
/textarea>
<br><br>
Gender:
<input type="radio" name="gender" value="female">Femal
e
<input type="radio" name="gender" value="male">Male
<input type="radio" name="gender" value="other">Other
<span class="error">* <?php echo $genderErr;?></span>
<br><br>
<input type="submit" name="submit" value="Submit">

</form>
```

PHP Forms - Validate Name

- The code below shows a simple way to check if the name field only contains letters and whitespace. If the value of the name field is not valid, then store an error message:

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
  $nameErr = "Only letters and white space allowed";
}
```

⚠ The *preg_match()* function searches a string for pattern, returning true if the pattern exists, and false otherwise.

PHP Forms - Validate E-mail

- The easiest and safest way to check whether an email address is well-formed is to use PHP's filter_var() function.
- In the code below, if the e-mail address is not well-formed, then store an error message:

```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  $emailErr = "Invalid email format";
}
```

PHP Forms - Validate URL

- The code below shows a way to check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

```
$website = test_input($_POST["website"]);

if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-
z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website)) {

  $websiteErr = "Invalid URL";

}
```

PHP Forms - Validate Name, E-mail, and URL

```php
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
    // check if name only contains letters and
whitespace
    if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
      $nameErr = "Only letters and white space
allowed";
    }
  }
```

PHP Forms - Validate Name, E-mail, and URL

```php
if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }
```

PHP Forms - Validate Name, E-mail, and URL

```
if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
    // check if URL address syntax is valid (this
regular expression also allows dashes in the URL)
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[
-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website)) {
      $websiteErr = "Invalid URL";
    }
  }
```

PHP Forms - Validate Name, E-mail, and URL

```
if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}
?>
```

PHP Forms - Keep The Values

- To show the values in the input fields after the user hits the submit button, we add a little PHP script inside the value attribute of the following input fields: name, email, and website. In the comment textarea field, we put the script between the `<textarea>` and `</textarea>` tags. The little script outputs the value of the $name, $email, $website, and $comment variables.
- Then, we also need to show which radio button that was checked. For this, we must manipulate the checked attribute (not the value attribute for radio buttons):

PHP Forms - Keep The Values

```
Name: <input type="text" name="name" value="<?php echo $
name;?>">
E-mail:
<input type="text" name="email" value="<?php echo $email
;?>">
Website: <input type="text" name="website" value="<?php
echo $website;?>">
Comment: <textarea name="comment" rows="5" cols="40"><?p
hp echo $comment;?></textarea>
```

PHP Forms - Keep The Values

```
Gender:
<input type="radio" name="gender"
<?php if (isset($gender) &&
$gender=="female") echo "checked";?>
value="female">Female
<input type="radio" name="gender"
<?php if (isset($gender) &&
$gender=="male") echo "checked";?>
value="male">Male
<input type="radio" name="gender"
<?php if (isset($gender) &&
$gender=="other") echo "checked";?>
value="other">Other
```