PHP Form Handling

by Sophia

WHAT'S COVERED

In this lesson, you will be introduced to form handling and how PHP is used to handle form submissions. Specifically, you will see how to access the HTTP request objects that hold the form data to retrieve the submitted information. Finally, you will learn how to perform form handling securely.

Specifically, this lesson will cover the following:

1. Introduction to PHP Form Handling
2. Retrieving Form Data
3. Securing Form Submission Using POST

1. Introduction to PHP Form Handling

Web forms can be a powerful tool for gathering information and data from a website visitor. However, once the form is built in HTML and stylized with CSS, what next? What happens to the data after the user clicks the form's "submit" button, and how does it get into a database and generate a confirmation email?

The answer is that a server-side scripting engine and script code file on the server receive the form submission data as an HTTP request and process the data by extracting the field values from the request. The script can then process the values and do whatever is needed:

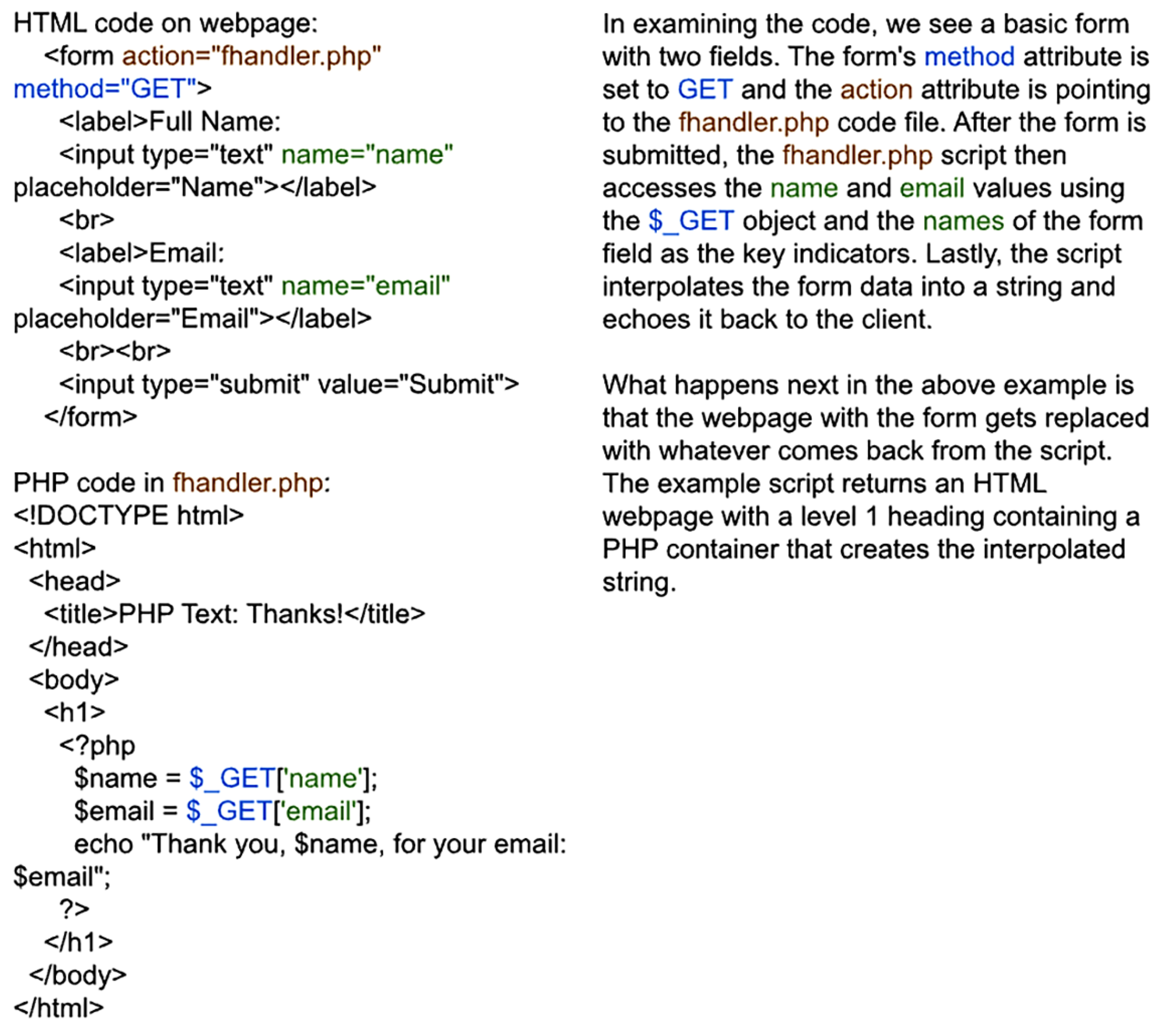
- Generate a confirmation email
- Store data inside a database
- Retrieve specific data from a database
- Validate credentials by comparing values to the database
- Write data to a file
- Store uploaded files to a particular directory on the server
- And more

2. Retrieving Form Data

Remember that when a web form is submitted, the contents of the form fields are packaged into an object. The object then gets put into an HTTP request object and sent to the server. On the server side, when using PHP, the data comes through as an object containing key/value pairs. The keys are the names of the fields (as specified in the HTML "name" attribute of the input tags), and the values are what the user entered or selected.

To access the values, we can reference either the **$_POST** or the **$_GET** object, depending on which HTTP method the form used (as specified in the HTML "method" attribute of the form tag). The following examples all use the GET method; POST will be discussed in the following section.

EXAMPLE Retrieving data from $_GET



HTML code on webpage:

```
   <form action="fhandler.php"
method="GET">
      <label>Full Name:
      <input type="text" name="name"
placeholder="Name"></label>
      <br>
      <label>Email:
      <input type="text" name="email"
placeholder="Email"></label>
      <br><br>
      <input type="submit" value="Submit">
   </form>
```

PHP code in fhandler.php:

```
<!DOCTYPE html>
<html>
  <head>
    <title>PHP Text: Thanks!</title>
  </head>
  <body>
    <h1>
      <?php
        $name = $_GET['name'];
        $email = $_GET['email'];
        echo "Thank you, $name, for your email:
$email";
      ?>
    </h1>
  </body>
</html>
```

In examining the code, we see a basic form with two fields. The form's method attribute is set to GET and the action attribute is pointing to the fhandler.php code file. After the form is submitted, the fhandler.php script then accesses the name and email values using the $_GET object and the names of the form field as the key indicators. Lastly, the script interpolates the form data into a string and echoes it back to the client.

What happens next in the above example is that the webpage with the form gets replaced with whatever comes back from the script. The example script returns an HTML webpage with a level 1 heading containing a PHP container that creates the interpolated string.

But what if we wanted the user to remain on the same page after submitting the form? This would be helpful because it allows the PHP script to report errors to the user and give them a chance to resolve the issue in the form. Furthermore, in such cases, we can reuse the data from the original submission attempt to repopulate the fields so that the user does not need to retype everything in case of an error.

In order to do this, we need to include a couple of changes to the webpage. First, the page will need to be saved as a .php file. Second, we need to make some changes to the form's action attribute. And finally, we can use PHP to detect a form submission, process the submission to generate the response message, and update the field's value attribute with the submitted data.

EXAMPLE Submitting a form to itself

```
<body>
  <div id="results">
      <?php if(isset($_GET['submit'])) {echo "Thank you, {$_GET['name']}, for
      your email: echo {$_GET['email']}";} ?>
  </div>
  <div>
    <form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>"
      method="GET">
    <label>Full Name:
    <input
      type="text"
      name="name"
      placeholder="Name"
      value="<?php if(isset($_GET['name'])){ echo $_GET['name'];} ?>">
      </label>
    <br>
    <label>Email:
    <input
      type="text"
      name="email"
      placeholder="Email"
      value="<?php if(isset($_GET['email'])){ echo $_GET['email']; }?>">
      </label>
    <br><br>
    <input type="submit" name="submit" value="Submit">
    </form>
  </div>
</body>
```

Notice the five changes to the form's webpage:

1. Starting at the bottom with the "submit" button, we added the name attribute. This was done so that we can detect if the user clicked the "submit" button or not and thus whether or not to repopulate the fields and

render the results.

2. The next two changes up from the “submit” button are the value attributes for the two fields. In this case, we want to repopulate the fields with the submitted data ONLY IF the “submit” button has been clicked. To do this, we use a PHP container, which is an “if” structure that makes a call to `isset( $_GET['name'])`. The PHP function isset( ) returns “true” if the object or variable does indeed exist. In this case, if the page was loaded for the first time, then isset() should return a “false,” leaving the field blank. Otherwise, if the form had been submitted, then the variable would exist, and isset() would return “true.”

3. Next up from the input fields is the action attribute for the form itself. We used a PHP container to echo the name of this current file. We could simply use the name of the file itself; however, this is another potential pitfall when maintaining a PHP website and could result in a wide range of issues should there be a filename mismatch. To avoid this issue, we have used the command `echo $_SERVER["PHP_SELF"]`.

4. Furthermore, there is a security vulnerability inherent with the use of `$_SERVER["PHP_SELF"]`, and to get around this vulnerability, we simply need to wrap the command in the htmlspecialchars() function.

5. Finally, the last change at the top was to add a similar PHP code that detects the existence of the “submit” button value (thanks to the “submit” button’s name attribute) and, if true, displays a confirmation message that interpolates the user’s submitted data. In scenarios wherein you needed to perform additional processing, such as generating and sending an email or connecting to and submitting the data to a database, you would use the same logic: Detect that the “submit” button was clicked using isset() and then perform any PHP processing as needed.

TERMS TO KNOW

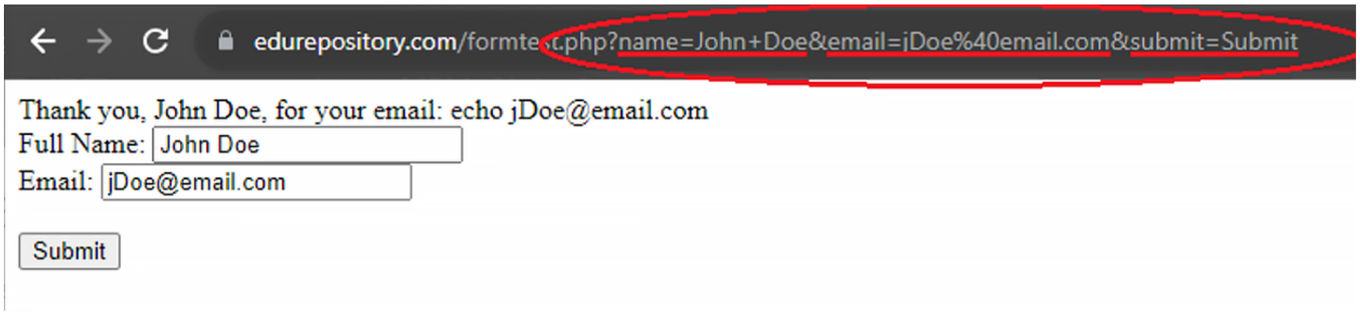
$_POST
The object containing all of the information that was submitted from an HTTP POST request, typically used to retrieve data from a submitted web form.

$_GET
The object containing all of the information that was submitted from an HTTP GET request.

---

3. Securing Form Submission Using POST

One thing you may have noticed if you attempted the code from the previous section is that your data from the form is displayed right in the navigation bar as part of the URL!


edurepository.com/formtest.php?name=John+Doe&email=jDoe%40email.com&submit=Submit

Thank you, John Doe, for your email: echo jDoe@email.com
Full Name: John Doe
Email: jDoe@email.com

This would pose a massive security concern, especially when transmitting sensitive personal data and information. This is due to the fact that even though you may be on an encrypted connection, the request URL itself is still clearly visible. To ensure that the data within the form gets encrypted, we need to swap the HTTP GET method for the POST method.

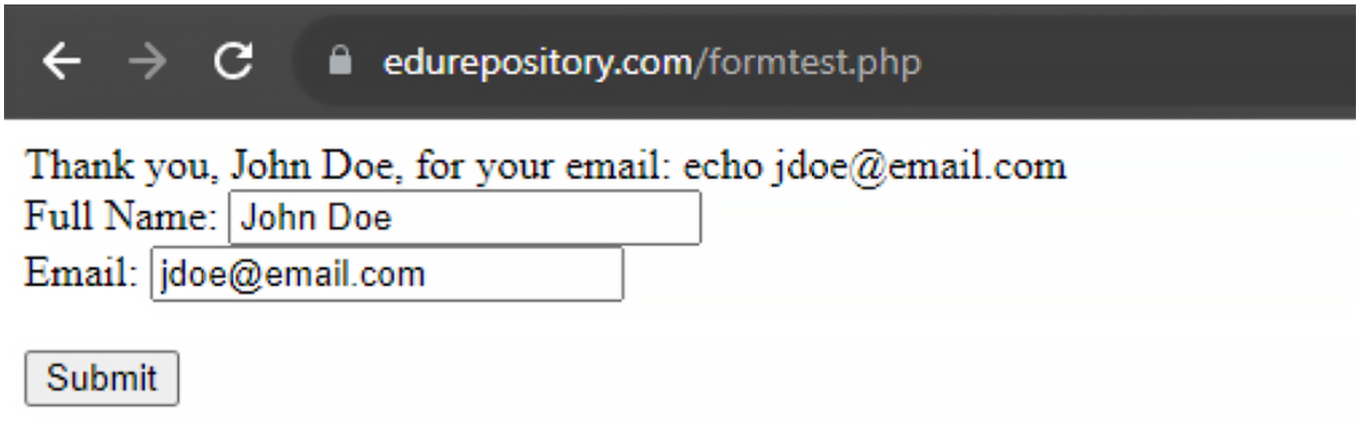
Making this switch is actually easy when dealing with web forms. All you need to do in order to use the GET method is to swap GET for POST anywhere in your code. On the front end, the form's action attribute will be changed to POST, and on the back end, any reference to $_GET needs to simply be swapped with $_POST.

EXAMPLE Submitting a form securely using POST

```
<body>
  <div id="results">
      <?php if(isset($_POST['submit'])) {echo "Thank you, {$_POST['name']}, for
      your email: echo {$_POST['email']}";} ?>
  </div>
  <div>
    <form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>"
      method="POST">
    <label>Full Name:
    <input
      type="text"
      name="name"
      placeholder="Name"
      value="<?php if(isset($_POST['name'])){ echo $_POST['name'];} ?>">
      </label>
    <br>
    <label>Email:
    <input
      type="text"
      name="email"
      placeholder="Email"
      value="<?php if(isset($_POST['email'])){ echo $_POST['email']; }?>">
      </label>
    <br><br>
```

```
        <input type="submit" name="submit" value="Submit">
        </form>
      </div>
    </body>
```

Testing the above script, we get the exact same results on the page, but no data has been added to the URL:



edurepository.com/formtest.php

Thank you, John Doe, for your email: echo jdoe@email.com
Full Name: John Doe
Email: jdoe@email.com

The data has instead been placed inside of the "body" of the HTTP request as a **payload**. This ensures that the data from the form gets properly encapsulated and thus encrypted as the payload of the HTTP request.

TERM TO KNOW

Payload
The portion of an HTTP request that contains the actual data being communicated.

SUMMARY

In this lesson, you learned how to process and **handle form** submissions using the server-side PHP scripting language. You saw how to **retrieve form data** from the HTTP request object and how it can be used within the script for processing. Lastly, you learned how to make the **form submission more secure using the POST** method and to secure the form's action attribute from attack using the htmlspecialchars() function.

Source: This Tutorial has been adapted from "The Missing Link: An Introduction to Web Development and Programming " by Michael Mendez. Access for free at https://open.umn.edu/opentextbooks/textbooks/the-missing-link-an-introduction-to-web-development-and-programming. License: Creative Commons attribution: CC BY-NC-SA.

TERMS TO KNOW

$_GET

The object containing all of the information that was submitted from an HTTP GET request.

$_POST

The object containing all of the information that was submitted from an HTTP POST request, typically used to retrieve data from a submitted web form.

Payload

The portion of an HTTP request that contains the actual data being communicated.