PHP

PHP 5 Form Handling

Form Handling [1]

- The PHP superglobals $_GET and $_POST are used to collect form-data.
- **Example 1 using $_POST**
- **Example 2 using $_GET**

Form Handling [1]

- **GET vs. POST**
  - Both GET and POST create an array (e.g. array(key => value, 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.

Form Handling [1]

- **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.
  - **Note:** GET should NEVER be used for sending passwords or other sensitive information!

Form Handling [1]

- **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**.

Form Validation [1]

PHP Form Validation Example

* required field.

Name: *

E-mail: *

Website:

Comment:

Gender: Female Male *

Submit

Your Input:

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

Form Validation [1]

- Form Element

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

- **What is the $_SERVER["PHP_SELF"] variable?**
- 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.

Form Validation [1]

- Form Element

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

- **What is the htmlspecialchars() function?**
- 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.

Form Validation [1]

- **PHP 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.

Form Validation [1]

- **PHP Form Security**

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

So far, so good.

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

```
http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('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>
```

Form Validation [1]

- **PHP Form Security**
  - How To 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;script&gt;alert('hacked')&lt;/script&gt;">
```

The exploit attempt fails, and no harm is done!

Form Validation [1]

- **Validate Form Data With PHP**
  - 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>

Form Validation [1]

- Required Field Validation
  - Example 4

Form Validation [1]

- Validate Name

```
$name = test_input($_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.

- Example 5

Form Validation [1]

- Validate Email

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";
}
```

- Example 5

Form Validation [1]

- 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";
}
```

- Example 5

Form Validation [1] [2]

- Example 6 for filter_var() function

- To repopulate data in case of wrong input.
  - Example : 4T.php

References

1. https://www.w3schools.com/php/
2. http://w3schools.sinsixx.com/php/php_ref_filter.asp.htm

Thank you….