Chapter 3-part 2:Form Validation

Think SECURITY when processing PHP forms!

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

However, this part will show how to process PHP forms with security in mind!

Proper **validation** of form **data** is **important** **to protect** your **data** from **hackers** and **spammers**!

Example

PHP Form Validation Example

* required field.

Name: *

E-mail: *

Website:

Comment:

Gender: Female Male *

Submit

Your Input:

Example: Validation Rules

The validation rules for the previous form are :

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

Example: HTML Form

```
<html><body>
<form method="post" action="<?php
echo htmlspecialchars($_SERVER["PHP_SELF"]);?>"  >
Name:  <input type="text" name="name"><br/>
E-mail: <input type="text" name="email"><br/>
Website: <input type="text" name="website"><br/>
Comment: <textarea name="comment" rows="5" cols="40">
                </textarea>
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<input type="submit">
</form>
</body></html>
```

Important Note

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

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.

Important Note : PHP Form Security

The **$_SERVER["PHP_SELF"]** variable can be used by **hackers!**

If **PHP_SELF** is used in your page (as the previous example) then a user can enter a slash (/) in the URL and then some **Cross Site Scripting** (XSS) commands to execute.

Example:

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

Important Note : PHP Form Security

Example:

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

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=" http://www.example.com/test_form.php "
>
```

Important Note : PHP Form Security

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

If a user enters the normal **URL** in the address bar like **http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E** the above code will be translated to:

Hackers Code injection

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

Important Note : PHP Form Security

Be aware of that **any JavaScript code can be added inside the <script> tag!**

A **hacker can redirect** the **user to a file on another server**, and that **file can hold malicious code** that can alter the global variables or submit the form to another address to save the user data, for example.

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"]);
          ?>"  >
```

Will be translated into:

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

Validate Form Data With PHP

1. You have to pass all **super global variables** through **htmlspecialchars() function**.

2. When the user submits the form:
   a) **Strip unnecessary characters** (extra space, tab, newline) **from the user input data** (with the PHP **trim() function**)
   b) **Remove backslashes (\) from the user input data** (with the PHP **stripslashes() function**)

Create a function **test_input()** to do all the tests, and call it for each **$_POST variable.**

Example: test_input()

```php
<?php
	function test_input($data) {
		$data = trim($data);
		$data = stripslashes($data);
		$data = htmlspecialchars($data);

		return $data;
	}
?>
```

Example: Call test_input()

If the **REQUEST_METHOD** is **POST**, then the form has been submitted - and it should be validated.

Note that all input fields are optional. The script works even if the user does not enter any data.

```
<?php
$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"]);
        }?>
```