PHP
PHP Security

Primary source: https://www.w3schools.com/php/

Internet Programming 2, Lesson 8

May 2024

Aside:

- Creating graphics in PPT – Order of objects, align, save picture as, file formats: png vs jpeg
- Using online tools to create graphics on the fly – dabutton factory
- Using online tools for generating CSS – use same site as above or others…

```
background:    #15d798;
border-radius: 11px;
padding:       20px 45px;
color:         #ff0000;
display:       inline-block;
font:          normal bold 26px/1 "Open Sans", sans-serif;
text-align:    center;
```

Addition to VRN system Problem

Given that we have not commenced with the letter C as yet {in the real world, which started with the B) , I would like to know how many numbers have been issued / will be issued by the time we reach the first VRN starting with the letter C. Since we still starting from AA00AA (we not at the stage of removing the Vowels) the question becomes, how many unique permutation can we get from the system starting at AA00AA to AZ99ZZ. In essence, we want a mechanism to establish how many permutations exist in a range of numbers. Once you problem solve this turn this aspect into a function, where it accepts a start and end number and the function returns the number of unique combinations between the given numbers (include both end points)

Some pointers wrt current progress:

[A] PHP scripts should be in the same file as the form:

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

[B] Execute the PHP Script only if the form was submitted:

```
 if ($_SERVER["REQUEST_METHOD"] == "POST") {
        // script code
}
```

[C] Avoid $_SERVER["PHP_SELF"] Exploits :

```
function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
```

[D] Check that mandatory fields contain data: set error message to display next to form input box

```
 if (empty($_POST["name"])) {
     $nameErr = "Name is required";
    } else {
     $name = $_POST["name"];
   }
```

[E] Retaining form data in the event of an error:

```
Surname: <input type="text" name="surname“ value="<?php echo $surname;?>">
```

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 &lt; and &gt;. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

Note on 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.

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">
```

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>
```

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.
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"]);?>">
```

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!

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:

&lt;script&gt;location.href('http://www.hacked.com')&lt;/script&gt;

The code is now safe to be displayed on a page or inside an e-mail.
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().
Now, we can check each $_POST variable with the test_input() function, and the script looks like this:

Example

```
<?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;
}
?>
```

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.
The next step is to make input fields required and create error messages if needed.

Filename: numbers4.php

```
 <?php
  $num1 = $num2 = $sum =0;
  if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $num1 = cleanInput($_POST["num1"]);
  $num2 = cleanInput($_POST["num2"]);
  $sum = $num1 + $num2;
    }
function cleanInput($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
    }
?>
```

Checking for empty required fields and displaying an error message

```
<h2>PHP Form Validation Example</h2>
<p><span class="error">* required field.</span></p>
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
  First Name(s): <input type="text" name="firstName" >
  <span class="error">* <?php echo $firstNameErr;?></span>
  <br><br>
  Surname: <input type="text" name="surname" >
  <span class="error">* <?php echo $surnameErr;?></span>
  <br><br>
  <input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>Your Input:</h2>";
echo $firstName;
echo "<br>";
echo $surname;
echo "<br>";
?>
```

```
<?php
// define variables and set to empty values
$firstNameErr = $surnameErr = "";                    // initialize to empty string so when script runs first time,
                                                     // nothing will be displayed – form code on next page
$firstName = $surname = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["firstName"])) {                  // check if field has a value
    $firstNameErr = "Name is required";              // if false, set an error message – will be displayed next to input
                                                     // box on form –code on previous form page
  } else {
    $firstName = test_input($_POST["firstName"]); // if it exists, assign it to a variable for further processing
          }
    if (empty($_POST["surname"])) {
    $surnameErr = "Name is required";
  } else {
    $surname = test_input($_POST["surname"]);
    }
}
function test_input($data) {
  // as per normal – previous notes
}
?>
```

Retaining form data in the event of an error:

- set the value property of input box to echo the value of the variable which will hold its value in the php script - $firstName and $surname in this case. Note if form is running for first time, this variable will have been set to the empty string when it was created and iniatilized – previous page

```
<h2>PHP Form Validation Example</h2>
<p><span class="error">* required field.</span></p>
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
   First Name(s): <input type="text" name="firstName" value="<?php echo $firstName;?>">
   <span class="error">* <?php echo $firstNameErr;?></span>
   <br><br>
   Surname: <input type="text" name="surname"  value="<?php echo $surname;?>">
   <span class="error">* <?php echo $surnameErr;?></span>
   <br><br>
   <input type="submit" name="submit" value="Submit">
</form>
<?php
echo "<h2>Your Input:</h2>";
echo $firstName;
echo "<br>";
echo $surname;
echo "<br>";
?>
```

Summary

PHP scripts should be in the same file as the form:

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

Execute the PHP Script only if the form was submitted:

```
 if ($_SERVER["REQUEST_METHOD"] == "POST") {
       // script code
}
```

Check that mandatory fields contain data: set error message to display next to form input box

```
 if (empty($_POST["name"])) {
     $nameErr = "Name is required";
   } else {
     $name = $_POST["name"];
  }
```

Avoid $_SERVER["PHP_SELF"] Exploits :

```
function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
```

Retaining form data in the event of an error:

```
Surname: <input type="text" name="surname“ value="<?php echo $surname;?>">
```