**What is PHP?**

* PHP is an acronym for "PHP: Hypertext Preprocessor"
* PHP is a widely-used, open source scripting language
* PHP scripts are executed on the server
* PHP is free to download and use

**What is a PHP File?**

* PHP files can contain text, HTML, CSS, JavaScript, and PHP code
* PHP code are executed on the server, and the result is returned to the browser as plain HTML
* PHP files have extension ".php"

**What Can PHP Do?**

* PHP can generate dynamic page content
* PHP can create, open, read, write, delete, and close files on the server
* PHP can collect form data
* PHP can send and receive cookies
* PHP can add, delete, modify data in your database
* PHP can be used to control user-access
* PHP can encrypt data
* With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

**Why PHP?**

* PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
* PHP is compatible with almost all servers used today (Apache, IIS, etc.)
* PHP supports a wide range of databases
* PHP is free. Download it from the official PHP resource: [www.php.net](http://www.php.net/)
* PHP is easy to learn and runs efficiently on the server side
* A PHP script is executed on the server, and the plain HTML result is sent back to the browser.

**Basic PHP Syntax**

* A PHP script can be placed anywhere in the document.
* A PHP script starts with **<?php** and ends with **?>**:

<?php  
// PHP code goes here  
?>

* The default file extension for PHP files is ".php".
* A PHP file normally contains HTML tags, and some PHP scripting code.
* PHP statements end with a semicolon (;).

**Comments in PHP : # // /\*..\*/**

**PHP Case Sensitivity**

In PHP, all keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are NOT case-sensitive. However; all variable names are case-sensitive.

**Creating (Declaring) PHP Variables**

In PHP, a variable starts with the $ sign, followed by the name of the variable:

<?php $txt = "Hello world!";  
$x = 5;  
$y = 10.5;?>

After the execution of the statements above, the variable **$txt** will hold the value **Hello world!**, the variable **$x** will hold the value **5**, and the variable **$y** will hold the value **10.5**.

**Note:** When you assign a text value to a variable, put quotes around the value.

**Note:** Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

Think of variables as containers for storing data.

**Rules for PHP variables:**

* A variable starts with the $ sign, followed by the name of the variable
* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* Variable names are case-sensitive ($age and $AGE are two different variables)

Remember that PHP variable names are case-sensitive!

**Output Variables**

The PHP echo statement is often used to output data to the screen.

$txt = "W3Schools.com";  
echo "I love $txt!";

echo "I love " . $txt . "!";

$x = 5;  
$y = 4;  
echo $x + $y;

**PHP is a Loosely Typed Language**-we do not have to tell PHP which data type the variable is.PHP automatically converts the variable to the correct data type, depending on its value.In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

**PHP Variables Scope**-In PHP, variables can be declared anywhere in the script.

The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

Local , global , static

**Global and Local Scope**

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function. A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function. The global keyword is used to access a global variable from within a function. PHP also stores all global variables in an array called $GLOBALS[*index*]. The index holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

**PHP The global Keyword**

The global keyword is used to access a global variable from within a function.

<?php  
$x = 5;  
$y = 10;  
function myTest() {  
    global $x, $y;  
    $y = $x + $y;  
}  
myTest();  
echo $y; // outputs 15

?>

PHP also stores all global variables in an array called $GLOBALS[*index*]. The index holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

function myTest() {  
    $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];  
}

PHP The static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

function myTest() {  
    static $x = 0;  
    echo $x;  
    $x++;  
}  
myTest();  
myTest();  
myTest();

o/p: 012

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Note: The variable is still local to the function.

PHP echo and print Statements

echo and print are more or less the same. They are both used to output data to the screen.

The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

The echo statement can be used with or without parentheses: echo or echo().

echo "<h2>" . $txt1 . "</h2>";  
echo "Study PHP at " . $txt2 . "<br>";  
echo $x + $y;

echo "<h2>PHP is Fun!</h2>";

echo "This ", "string ", "was ", "made ", "with multiple parameters.";

The print statement can be used with or without parentheses: print or print().

print "<h2>PHP is Fun!</h2>";

print "<h2>" . $txt1 . "</h2>";  
print "Study PHP at " . $txt2 . "<br>";  
print $x + $y;

**PHP supports the following data types:**

**String**-can be any text inside quotes.can use single or double quotes: $x = "Hello";

**Integer**-An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.Rules for integers:

* An integer must have at least one digit
* An integer must not have a decimal point
* An integer can be either positive or negative
* Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

<?php   
$x = 5985;  
var\_dump($x);// var\_dump() function returns the data type and value  
?>

**Float** (floating point numbers - also called double)- A float (floating point number) is a number with a decimal point or a number in exponential form. $x = 10.365;

**Boolean**-$x = true;

**Array**-$cars = array("Volvo","BMW","Toyota");

**Object**-An object is a data type which stores data and information on how to process that data.In PHP, an object must be explicitly declared.First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods.

class Car {  
    function Car() {  
        $this->model = "VW";  
    }  
}  
// create an object  
$herbie = new Car();  
// show object properties  
echo $herbie->model;  
?>

**NULL**- Null is a special data type which can have only one value: NULL.A variable of data type NULL is a variable that has no value assigned to it.**Tip:** If a variable is created without a value, it is automatically assigned a value of NULL.Variables can also be emptied by setting the value to NULL. $x = null;

**Resource**-The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP.A common example of using the resource data type is a database call.

PHP String Functions

1. strlen() returns the length of a string. echo strlen("Hello world!"); // outputs 12
2. str\_word\_count() counts the number of words in a string. echo str\_word\_count("Hello world!"); // outputs 2
3. strrev() reverses a string echo strrev("Hello world!"); // outputs !dlrow olleH
4. strpos() searches for a specific text within a string.If a match is found, the function returns the character position of the first match. If no match is found, it will return FALSE. echo strops ("Hello world!", "world"); // outputs 6
5. str\_replace() replaces some characters with some other characters in a string. echo str\_replace ("world", "Dolly", "Hello world!"); // outputs Hello Dolly!

**PHP Constants**

A constant is an identifier (name) for a simple value. The value cannot be changed during the script.A valid constant name starts with a letter or underscore (no $ sign before the constant name).**Note:** Unlike variables, constants are automatically global across the entire script.To create a constant, use the define() function.

define(*name*, *value*, *case-insensitive*)

Parameters:

*name*: Specifies the name of the constant

*value*: Specifies the value of the constant

*case-insensitive*: Specifies whether the constant name should be case-insensitive. Default is false

define("GREETING", "Welcome to W3Schools.com!");  
echo GREETING;

define("GREETING", "Welcome to W3Schools.com!", true);  
echo greeting;

Constants are automatically global and can be used across the entire script.

define("GREETING", "Welcome to W3Schools.com!");  
  
function myTest() {  
    echo GREETING;  
}

**PHP Operators**

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

Arithmetic operators - +,-,\*,/,%,\*\*

Assignment operators : x=y ,x+=y , x-=y x\*=y ,x/=y ,x%=y

Comparison operators : == , ===(identical,equal and same type),!= <>(not equal both),!==,>,<, >=,<=

Increment/Decrement operators-pre and post

Logical operators: and ,or,xor,&&,||,!

String operators : .(concatenation-$x.$y) , .=(concatenation assignmrnt $x.=y$ -appends $y to $x

Array operators :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operttor | Name | Example | Result |  |
| + | Union | $x + $y | Union of $x and $y |  |
| == | Equality | $x == $y | Returns true if $x and $y have the same key/value pairs |  |
| === | Identity | $x === $y | Returns true if $x and $y have the same key/value pairs in the same order and of the same types |  |
| != | Inequality | $x != $y | Returns true if $x is not equal to $y |  |
| <> | Inequality | $x <> $y | Returns true if $x is not equal to $y |  |
| !== | Non-identity | $x !== $y | Returns true if $x is not identical to $y |  |

In PHP we have the following conditional statements:

if statement - executes some code if one condition is true

if...else statement - executes some code if a condition is true and another code if that condition is false

if...elseif....else statement - executes different codes for more than two conditions

switch statement - selects one of many blocks of code to be executed

In PHP, we have the following looping statements:

while - loops through a block of code as long as the specified condition is true

do...while - loops through a block of code once, and then repeats the loop as long as the specified condition is true

for - loops through a block of code a specified number of times

foreach - loops through a block of code **for each element in an array**

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

foreach ($*array*as$*value*) {  
    *code to be executed;*  
}

For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

$colors = array("red", "green", "blue", "yellow");   
  
foreach ($colors as $value) {  
    echo "$value <br>";  
}

**Functions in php**

function functionName() {  
    code to be executed;  
}

Note: A function name can start with a letter or underscore (not a number).

Tip: Give the function a name that reflects what the function does!

Function names are NOT case-sensitive.

Arguments can be adde d using comma in the function.

Can also use default parameters

Return statement to return the result from fuction

**Create an Array in PHP**

In PHP, the array() function is used to create an array:

array();

In PHP, there are three types of arrays:

**Indexed arrays** - Arrays with a numeric index

**Associative arrays** - Arrays with named keys

**Multidimensional arrays** - Arrays containing one or more arrays

**PHP Global Variables - Superglobals**

Several predefined variables in PHP 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.

The PHP superglobal variables are:

**PHP $GLOBALS**

$GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).PHP stores all global variables in an array called $GLOBALS[*index*]. The index holds the name of the variable.

<?php   
$x = 75;   
$y = 25;  
   
function addition() {   
    $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];   
}  
addition();   
echo $z;   
?> prints 100

In the example above, since z is a variable present within the $GLOBALS array, it is also accessible from outside the function!

**PHP $\_SERVER**

The following table lists the most important elements that can go inside $\_SERVER:

|  |  |
| --- | --- |
| Element/Code | Description |
| $\_SERVER['PHP\_SELF'] | Returns the filename of the currently executing script |
| $\_SERVER['GATEWAY\_INTERFACE'] | Returns the version of the Common Gateway Interface (CGI) the server is using |
| $\_SERVER['SERVER\_ADDR'] | Returns the IP address of the host server |
| $\_SERVER['SERVER\_NAME'] | Returns the name of the host server (such as www.w3schools.com) |
| $\_SERVER['SERVER\_SOFTWARE'] | Returns the server identification string (such as Apache/2.2.24) |
| $\_SERVER['SERVER\_PROTOCOL'] | Returns the name and revision of the information protocol (such as HTTP/1.1) |
| $\_SERVER['REQUEST\_METHOD'] | Returns the request method used to access the page (such as POST) |
| $\_SERVER['REQUEST\_TIME'] | Returns the timestamp of the start of the request (such as 1377687496) |
| $\_SERVER['QUERY\_STRING'] | Returns the query string if the page is accessed via a query string |
| $\_SERVER['HTTP\_ACCEPT'] | Returns the Accept header from the current request |
| $\_SERVER['HTTP\_ACCEPT\_CHARSET'] | Returns the Accept\_Charset header from the current request (such as utf-8,ISO-8859-1) |
| $\_SERVER['HTTP\_HOST'] | Returns the Host header from the current request |
| $\_SERVER['HTTP\_REFERER'] | Returns the complete URL of the current page (not reliable because not all user-agents support it) |
| $\_SERVER['HTTPS'] | Is the script queried through a secure HTTP protocol |
| $\_SERVER['REMOTE\_ADDR'] | Returns the IP address from where the user is viewing the current page |
| $\_SERVER['REMOTE\_HOST'] | Returns the Host name from where the user is viewing the current page |
| $\_SERVER['REMOTE\_PORT'] | Returns the port being used on the user's machine to communicate with the web server |
| $\_SERVER['SCRIPT\_FILENAME'] | Returns the absolute pathname of the currently executing script |
| $\_SERVER['SERVER\_ADMIN'] | Returns the value given to the SERVER\_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as someone@w3schools.com) |
| $\_SERVER['SERVER\_PORT'] | Returns the port on the server machine being used by the web server for communication (such as 80) |
| $\_SERVER['SERVER\_SIGNATURE'] | Returns the server version and virtual host name which are added to server-generated pages |
| $\_SERVER['PATH\_TRANSLATED'] | Returns the file system based path to the current script |
| $\_SERVER['SCRIPT\_NAME'] | Returns the path of the current script |
| $\_SERVER['SCRIPT\_URI'] | Returns the URI of the current page |

$\_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.

<?php   
echo $\_SERVER['PHP\_SELF'];  
echo "<br>";  
echo $\_SERVER['SERVER\_NAME'];  
echo "<br>";  
echo $\_SERVER['HTTP\_HOST'];  
echo "<br>";  
echo $\_SERVER['HTTP\_REFERER'];  
echo "<br>";  
echo $\_SERVER['HTTP\_USER\_AGENT'];  
echo "<br>";  
echo $\_SERVER['SCRIPT\_NAME'];  
?> prints pHP/demo\_global\_server.php  
www.w3schools.com  
www.w3schools.com  
https://www.w3schools.com/pHP/showphp.asp?filename=demo\_global\_server  
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3396.99 Safari/537.36  
/pHP/demo\_global\_server.php

**PHP $\_REQUEST -** PHP $\_REQUEST is used to collect data after submitting an HTML form.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $\_REQUEST to collect the value of the input field:

<form method="post" action="<?php echo $\_SERVER['PHP\_SELF'];?>">  
  Name: <input type="text" name="fname">  
  <input type="submit">  
</form>  
  
<?php  
if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
    // collect value of input field  
    $name = $\_REQUEST['fname'];  
    if (empty($name)) {  
        echo "Name is empty";  
    } else {  
        echo $name;  
    }  
}  
?>

**PHP $\_POST-** PHP $\_POST is widely used to collect form data after submitting an HTML form with method="post". $\_POST is also widely used to pass variables.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to the file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $\_POST to collect the value of the input field:

<form method="post" action="<?php echo $\_SERVER['PHP\_SELF'];?>">  
  Name: <input type="text" name="fname">  
  <input type="submit">  
</form>  
  
<?php  
if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
    // collect value of input field  
    $name = $\_POST['fname'];  
    if (empty($name)) {  
        echo "Name is empty";  
    } else {  
        echo $name;  
    }  
}  
?>

**PHP $\_GET -** PHP $\_GET can also be used to collect form data after submitting an HTML form with method="get".

$\_GET can also collect data sent in the URL.

Assume we have an HTML page that contains a hyperlink with parameters:

<html>  
<body>  
<a href="test\_get.php?subject=PHP&web=W3schools.com">Test $GET</a>  
</body>  
</html>

When a user clicks on the link "Test $GET", the parameters "subject" and "web" are sent to "test\_get.php", and you can then access their values in "test\_get.php" with $\_GET.

The example below shows the code in "test\_get.php":

<?php   
echo "Study " . $\_GET['subject'] . " at " . $\_GET['web'];  
?>

**ARRays continued**

**PHP Indexed Arrays**

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:

$cars = array("Volvo", "BMW", "Toyota");

or the index can be assigned manually:

$cars[0] = "Volvo";  
$cars[1] = "BMW";  
$cars[2] = "Toyota";

count() function is used to return the length (the number of elements) of an array:

Loop Through an Indexed Array

To loop through and print all the values of an indexed array, you could use a forloop

$cars = array("Volvo", "BMW", "Toyota");  
$arrlength = count($cars);  
  
for($x = 0; $x < $arrlength; $x++) {  
    echo $cars[$x];  
    echo "<br>";  
}

**PHP Associative Arrays**

Associative arrays are arrays that use named keys that you assign to them.

There are two ways to create an associative array:

$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

or:

$age['Peter'] = "35";  
$age['Ben'] = "37";  
$age['Joe'] = "43";

Loop Through an Associative Array

To loop through and print all the values of an associative array, you could use a foreach loop

(as we are using strings as keys)

$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
foreach($age as $x => $x\_value) {  
    echo "Key=" . $x . ", Value=" . $x\_value;  
    echo "<br>";  
}

PHP - Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.

PHP understands multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

**The dimension of an array indicates the number of indices you need to select an element.**

For a two-dimensional array you need two indices to select an element

For a three-dimensional array you need three indices to select an element

PHP - Two-dimensional Arrays

A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays).

$cars = array  
  (  
  array("Volvo",22,18),  
  array("BMW",15,13),  
  array("Saab",5,2),  
  array("Land Rover",17,15)  
  );

<?php  
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";  
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";

for ($row = 0; $row < 4; $row++) {  
  echo "<p><b>Row number $row</b></p>";  
  echo "<ul>";  
  for ($col = 0; $col < 3; $col++) {  
    echo "<li>".$cars[$row][$col]."</li>";  
  }  
  echo "</ul>";  
}

**PHP - Sort Functions For Arrays**

In this chapter, we will go through the following PHP array sort functions:

sort() - sort arrays in ascending order

rsort() - sort arrays in descending order

asort() - sort associative arrays in ascending order, according to the value

ksort() - sort associative arrays in ascending order, according to the key

arsort() - sort associative arrays in descending order, according to the value

krsort() - sort associative arrays in descending order, according to the key

**PHP 5 Form Handling**

**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.

**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!

**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.**

**The Form Element**

The HTML code of the form looks like this:

<form method="post" action="<?php echohtmlspecialchars($\_SERVER["PHP\_SELF"]);?>">

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

**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.

**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.

**Big 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">

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>

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 exampl**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 echohtmlspecialchars($\_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.

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

$name = test\_input($\_POST["name"]);  
if (!preg\_match("/^[a-zA-Z ]\*$/",$name)) {  
  $nameErr = "Only letters and white space allowed";   
}

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";   
}

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";   
}

**Complete form**

<?php

$name = $mail = $gender = $comment = $site = "";

$nameerr = $mailerr = $gendererr = $siteerr ="";

/\*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.\*/

if($\_SERVER["REQUEST\_METHOD"] == "POST") {

if(!empty($\_POST["name"])) {

$name=test\_input($\_POST["name"]);

if(!preg\_match("/^[a-zA-Z ]\*$/", $name)){

$nameerr = "Only letters and spaces allowed";

}

}else {

$nameerr = "Name is reqd";

}

if(!empty($\_POST["mail"])) {

$mail=test\_input($\_POST["mail"]);

if(!filter\_var($mail, FILTER\_VALIDATE\_EMAIL)){

$mailerr = "Invalid email address";

}

}else {

$mailerr = "mail is reqd";

}

if(!empty($\_POST["site"])) {

$site=test\_input($\_POST["site"]);

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

$siteerr = "Invalid URL";

}

}

if(!empty($\_POST["comment"])) {

$comment=test\_input($\_POST["comment"]);

}

if(!empty($\_POST["gender"])) {

$gender=test\_input($\_POST["gender"]);

}else {

$gendererr = "gender is reqd";

}

}

/\*

We will also do two more things when the user submits the form:

Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)

Remove backslashes (\) from the user input data (with the PHP stripslashes() function)\*/

function test\_input($data) {

$data = trim($data);

$data = stripslashes($data);

$data=htmlspecialchars($data);

return $data;

}

?>

<html>

<head>

<style>

.error{

color:#f00;

font-size: 10px;

}

label{

min-width: 100px;

}

input,textarea{

border-radius: 5px;

margin: 10px;

padding: 10px;

}

</style>

</head>

<body>

<!--<form action="welcome.php" method="post">-->

<form method="post" action="<?php echo htmlspecialchars($\_SERVER["PHP\_SELF"])?>">

<label>Name:</label>

<input type="text" name="name">

<span class="error">\*<?php echo $nameerr;?></span><br/>

<label>Email:</label>

<input type="text" name="mail">

<span class="error">\*<?php echo $mailerr;?></span><br/>

<label>Website:</label>

<input type="text" name="site">

<span class="error"><?php echo $siteerr;?></span><br/>

<label>Comments:</label>

<textarea name="comment" rows="5" cols="40"></textarea><br/>

<label>Gender:</label>

<input type="radio" <?php if(isset($gender) && $gender=="male") echo "male checked"; ?>value="male" name="gender">Male

<input type="radio" <?php if(isset($gender) && $gender=="female") echo "female checked"; ?> value="female" name="gender">Female

<input type="radio" <?php if(isset($gender) && $gender=="other") echo "other checked"; ?> value="other" name="gender">Other

<span class="error">\*<?php echo $gendererr;?></span><br/>

<button type="submit">Submit</button>

</form>

</body>

</html>

<?php

echo "<b>"."Your input"."</b><br/>" ;

echo "Name: ".$name."<br/>";

echo "mail: ".$mail."<br/>";

echo "gender: ".$gender."<br/>";

echo "site: ".$site."<br/>";

?>

The PHP date() function is used to format a date and/or a time.

**The PHP Date() Function**

The PHP date() function formats a timestamp to a more readable date and time.

**Syntax** : date(*format*,*timestamp*)

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| format | Required. Specifies the format of the timestamp |
| timestamp | Optional. Specifies a timestamp. Default is the current date and time |

A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred

**Get a Simple Date**

The required *format* parameter of the date() function specifies how to format the date (or time).

Here are some characters that are commonly used for dates:

d - Represents the day of the month (01 to 31)

m - Represents a month (01 to 12)

Y - Represents a year (in four digits)

l (lowercase 'L') - Represents the day of the week

Other characters, like"/", ".", or "-" can also be inserted between the characters to add additional formatting.

**Get a Simple Time**

Here are some characters that are commonly used for times:

h - 12-hour format of an hour with leading zeros (01 to 12)

i - Minutes with leading zeros (00 to 59)

s - Seconds with leading zeros (00 to 59)

a - Lowercase Ante meridiem and Post meridiem (am or pm)

echo "The time is " . date("h:i:sa");

**Get Your Time Zone**

If the time you got back from the code is not the right time, it's probably because your server is in another country or set up for a different timezone.

So, if you need the time to be correct according to a specific location, you can set a timezone to use.

date\_default\_timezone\_set("America/New\_York");  
echo "The time is " . date("h:i:sa");

**Create a Date With PHP mktime()**

The optional *timestamp* parameter in the date() function specifies a timestamp. If you do not specify a timestamp, the current date and time will be used (as shown in the examples above).

The mktime() function returns the Unix timestamp for a date. The Unix timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.

**Syntax**

mktime(hour,minute,second,month,day,year)

$d=mktime(11, 14, 54, 8, 12, 2014);  
echo "Created date is " . date("Y-m-d h:i:sa", $d);

**Create a Date From a String With PHP strtotime()**

The PHP strtotime() function is used to convert a human readable string to a Unix time.

**Syntax**

strtotime(time,now)

$d=strtotime("10:30pm April 15 2014");  
echo "Created date is " . date("Y-m-d h:i:sa", $d);

$d=strtotime("tomorrow");  
echo date("Y-m-d h:i:sa", $d) . "<br>";  
$d=strtotime("next Saturday");  
echo date("Y-m-d h:i:sa", $d) . "<br>";  
$d=strtotime("+3 Months");  
echo date("Y-m-d h:i:sa", $d) . "<br>";

The example below outputs the dates for the next six Saturdays:

<?php  
$startdate = strtotime("Saturday");  
$enddate = strtotime("+6 weeks", $startdate);  
  
while ($startdate < $enddate) {  
  echo date("M d", $startdate) . "<br>";  
  $startdate = strtotime("+1 week", $startdate);  
}  
?>

The example below outputs the number of days until 4th of July:

<?php  
$d1=strtotime("July 04");  
$d2=ceil(($d1-time())/60/60/24);  
echo "There are " . $d2 ." days until 4th of July.";  
?>

The include (or require) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.

Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a websit

**PHP include and require Statements**

It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with the include or require statement.

**The include and require statements are identical, except upon failure:**

require will produce a fatal error (E\_COMPILE\_ERROR) and stop the script

include will only produce a warning (E\_WARNING) and the script will continue

So, if you want the execution to go on and show users the output, even if the include file is missing, use the include statement. Otherwise, in case of FrameWork, CMS, or a complex PHP application coding, always use the require statement to include a key file to the flow of execution. This will help avoid compromising your application's security and integrity, just in-case one key file is accidentally missing.

Including files saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. Then, when the header needs to be updated, you can only update the header include file.

include '*filename*'; or require '*filename*';

Use require when the file is required by the application.

Use include when the file is not required and application should continue when file is not found.

ile handling is an important part of any web application. You often need to open and process a file for different tasks.

**PHP Manipulating Files**

PHP has several functions for creating, reading, uploading, and editing files.

**Be careful when manipulating files!**

When you are manipulating files you must be very careful.

You can do a lot of damage if you do something wrong. Common errors are: editing the wrong file, filling a hard-drive with garbage data, and deleting the content of a file by accident.

**PHP readfile() Function**

The readfile() function reads a file and writes it to the output buffer.

Assume we have a text file called "webdictionary.txt", stored on the server, that looks like this:

AJAX = Asynchronous JavaScript and XML  
CSS = Cascading Style Sheets  
HTML = Hyper Text Markup Language  
PHP = PHP Hypertext Preprocessor  
SQL = Structured Query Language  
SVG = Scalable Vector Graphics  
XML = EXtensible Markup Language

The PHP code to read the file and write it to the output buffer is as follows (the readfile() function returns the number of bytes read on success)

echo readfile("webdictionary.txt");

The readfile() function is useful if all you want to do is open up a file and read its contents.

**PHP Open File - fopen()**

A better method to open files is with the fopen() function. This function gives you more options than the readfile() function. The first parameter of fopen() contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened.

$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");

The file may be opened in one of the following modes:

|  |  |
| --- | --- |
| **Modes** | **Description** |
| r | **Open a file for read only**. File pointer starts at the beginning of the file |
| w | **Open a file for write only**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file |
| a | **Open a file for write only**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist |
| x | **Creates a new file for write only**. Returns FALSE and an error if file already exists |
| r+ | **Open a file for read/write**. File pointer starts at the beginning of the file |
| w+ | **Open a file for read/write**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file |
| a+ | **Open a file for read/write**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist |
| x+ | **Creates a new file for read/write**. Returns FALSE and an error if file already exists |

**PHP Read File - fread()**

The fread() function reads from an open file.

The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

The following PHP code reads the "webdictionary.txt" file to the end:

fread($myfile,filesize("webdictionary.txt"));

**PHP Close File - fclose()**

The fclose() function is used to close an open file.

It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources!

The fclose() requires the name of the file (or a variable that holds the filename) we want to close: $myfile = fopen("webdictionary.txt", "r");  
// some code to be executed....  
fclose($myfile);

**PHP Read Single Line - fgets()**

The fgets() function is used to read a single line from a file.

$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");  
echo fgets($myfile);

**Note:** After a call to the fgets() function, the file pointer has moved to the next line

**PHP Check End-Of-File - feof()**

The feof() function checks if the "end-of-file" (EOF) has been reached.

The feof() function is useful for looping through data of unknown length.

myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");  
// Output one line until end-of-file  
while(!feof($myfile)) {  
  echo fgets($myfile) . "<br>";  
}

**PHP Read Single Character - fgetc()**

The fgetc() function is used to read a single character from a file.

while(!feof($myfile)) {  
  echo fgetc($myfile);  
}

**Note:** After a call to the fgetc() function, the file pointer moves to the next character.

Skipped part from file create to php exceptions

**MYSQL DB**

With PHP, you can connect to and manipulate databases.

MySQL is the most popular database system used with PHP.

What is MySQL?

* MySQL is a database system used on the web
* MySQL is a database system that runs on a server
* MySQL is ideal for both small and large applications
* MySQL is very fast, reliable, and easy to use
* MySQL uses standard SQL
* MySQL compiles on a number of platforms
* MySQL is free to download and use
* MySQL is developed, distributed, and supported by Oracle Corporation
* MySQL is named after co-founder Monty Widenius's daughter: My

The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.

Databases are useful for storing information categorically.

PHP + MySQL Database System

PHP combined with MySQL are cross-platform (you can develop in Windows and serve on a Unix platform)

**Facts About MySQL Database**

MySQL is the de-facto standard database system for web sites with HUGE volumes of both data and end-users (like Facebook, Twitter, and Wikipedia).

Another great thing about MySQL is that it can be scaled down to support embedded database applications.

PHP 5 and later can work with a MySQL database using:

**MySQLi extension** (the "i" stands for improved)

**PDO (PHP Data Objects)**

**Should I Use MySQLi or PDO?**

If you need a short answer, it would be "Whatever you like".

Both MySQLi and PDO have their advantages:

PDO will work on 12 different database systems, whereas MySQLi will only work with MySQL databases.

So, if you have to switch your project to use another database, PDO makes the process easy. You only have to change the connection string and a few queries. With MySQLi, you will need to rewrite the entire code - queries included.

Both are object-oriented, but MySQLi also offers a procedural API.

Both support Prepared Statements. Prepared Statements protect from SQL injection, and are very important for web application security.

MySQL Examples in Both MySQLi and PDO Syntax

In this, and in the following chapters we demonstrate three ways of working with PHP and MySQL:

MySQLi (object-oriented)

MySQLi (procedural)

PDO

**Open a Connection to MySQL**

<?php  
$servername = "localhost";  
$username = "username";  
$password = "password";  
  
// Create connection  
$conn = new mysqli($servername, $username, $password);  
  
// Check connection  
if ($conn->connect\_error) {  
    die("Connection failed: " . $conn->connect\_error);  
}   
echo "Connected successfully";  
?>

//mysqli procedural

// Create connection  
$conn = mysqli\_connect($servername, $username, $password);  
  
// Check connection  
if (!$conn) {  
    die("Connection failed: " . mysqli\_connect\_error());  
}

//pdo

try {  
    $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);  
    // set the PDO error mode to exception  
    $conn->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);  
    echo "Connected successfully";   
    }  
catch(PDOException $e)  
    {  
    echo "Connection failed: " . $e->getMessage();  
    }

Notice that in the PDO example above we have also specified a database (myDB). PDO require  a valid database to connect to. If no database is specified, an exception is thrown.

**Tip:** A great benefit of PDO is that it has an exception class to handle any problems that may occur in our database queries. If an exception is thrown within the try{ } block, the script stops executing and flows directly to the first catch(){ } block.

**Close the Connection**

The connection will be closed automatically when the script ends. To close the connection before,

**Example (MySQLi Object-Oriented)**

$conn->close();

**Example (MySQLi Procedural)**

mysqli\_close($conn);

//pdo

$conn = null;

**(MySQLi Object-oriented)**

// Create connection  
$conn = new mysqli($servername, $username, $password);  
// Check connection  
if ($conn->connect\_error) {  
    die("Connection failed: " . $conn->connect\_error);  
}   
  
// Create database  
$sql = "CREATE DATABASE myDB";  
if ($conn->query($sql) === TRUE) {  
    echo "Database created successfully";  
} else {  
    echo "Error creating database: " . $conn->error;  
}  
  
$conn->close();

//creating a db

<?php  
$servername = "localhost";  
$username = "username";  
$password = "password";  
  
// Create connection  
$conn = mysqli\_connect($servername, $username, $password);  
// Check connection  
if (!$conn) {  
    die("Connection failed: " . mysqli\_connect\_error());  
}  
  
// Create database  
$sql = "CREATE DATABASE myDB";  
if (mysqli\_query($conn, $sql)) {  
    echo "Database created successfully";  
} else {  
    echo "Error creating database: " . mysqli\_error($conn);  
}  
  
mysqli\_close($conn);  
?>

//pdo

try {  
    $conn = new PDO("mysql:host=$servername", $username, $password);  
    // set the PDO error mode to exception  
    $conn->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);  
    $sql = "CREATE DATABASE myDBPDO";  
    // use exec() because no results are returned  
    $conn->exec($sql);  
    echo "Database created successfully<br>";  
    }  
catch(PDOException $e)  
    {  
    echo $sql . "<br>" . $e->getMessage();  
    }  
  
$conn = null;

CREATE TABLE MyGuests (  
id INT(6) UNSIGNED AUTO\_INCREMENT PRIMARY KEY,  
firstname VARCHAR(30) NOT NULL,  
lastname VARCHAR(30) NOT NULL,  
email VARCHAR(50),  
reg\_date TIMESTAMP  
)

can specify other optional attributes for each column:

* NOT NULL - Each row must contain a value for that column, null values are not allowed
* DEFAULT value - Set a default value that is added when no other value is passed
* UNSIGNED - Used for number types, limits the stored data to positive numbers and zero
* AUTO INCREMENT - MySQL automatically increases the value of the field by 1 each time a new record is added
* PRIMARY KEY - Used to uniquely identify the rows in a table. The column with PRIMARY KEY setting is often an ID number, and is often used with AUTO\_INCREMENT

Each table should have a primary key column (in this case: the "id" column). Its value must be unique for each record in the table.