28. PHP Filter

PHP filters are used to validate and filter data coming from insecure sources, like user input.

28.1. What is a PHP Filter?

A PHP filter is used to validate and filter data coming from insecure sources.

To test, validate and filter user input or custom data is an important part of any web application.

The PHP filter extension is designed to make data filtering easier and quicker.

28.2. Why use a Filter?

Almost all web applications depend on external input. Usually this comes from a user or another application (like a web service). By using filters you can be sure your application gets the correct input type.

You should always filter all external data!

Input filtering is one of the most important application security issues.

What is external data?

- Input data from a form
- Cookies
- Web services data
- Server variables
- Database query results

28.3. Functions and Filters

To filter a variable, use one of the following filter functions:

- filter_var() - Filters a single variable with a specified filter
- filter_var_array() - Filter several variables with the same or different filters
- filter_input - Get one input variable and filter it
- filter_input_array - Get several input variables and filter them with the same or different filters

In the example below, we validate an integer using the filter_var() function:

```
<?php
$int = 123;

if(!filter_var($int, FILTER_VALIDATE_INT))
  {
  echo("Integer is not valid");
```

```
    }
  else
    {
    echo("Integer is valid");
    }
  ?>
```

The code above uses the "FILTER_VALIDATE_INT" filter to filter the variable. Since the integer is valid, the output of the code above will be: "Integer is valid".

If we try with a variable that is not an integer (like "123abc"), the output will be: "Integer is not valid".

For a complete list of functions and filters, visit our PHP Filter Reference.

28.4. Validating and Sanitizing

There are two kinds of filters:

Validating filters:

- Are used to validate user input
- Strict format rules (like URL or E-Mail validating)
- Returns the expected type on success or FALSE on failure

Sanitizing filters:

- Are used to allow or disallow specified characters in a string
- No data format rules
- Always return the string

28.5. Options and Flags

Options and flags are used to add additional filtering options to the specified filters.

Different filters have different options and flags.

In the example below, we validate an integer using the filter_var() and the "min_range" and "max_range" options:

```
  <?php
  $var=300;

  $int_options = array(
  "options"=>array
    (
    "min_range"=>0,
    "max_range"=>256
    )
  );
```

```
if(!filter_var($var, FILTER_VALIDATE_INT, $int_options))
  {
  echo("Integer is not valid");
  }
else
  {
  echo("Integer is valid");
  }
?>
```

Like the code above, options must be put in an associative array with the name "options". If a flag is used it does not need to be in an array.

Since the integer is "300" it is not in the specified range, and the output of the code above will be: "Integer is not valid".

For a complete list of functions and filters, visit our PHP Filter Reference. Check each filter to see what options and flags are available.

28.6. Validate Input

Let's try validating input from a form.

The first thing we need to do is to confirm that the input data we are looking for exists.

Then we filter the input data using the filter_input() function.

In the example below, the input variable "email" is sent to the PHP page:

```
<?php
if(!filter_has_var(INPUT_GET, "email"))
  {
  echo("Input type does not exist");
  }
else
  {
  if           (!filter_input(INPUT_GET,           "email",
FILTER_VALIDATE_EMAIL))
    {
    echo "E-Mail is not valid";
    }
  else
    {
    echo "E-Mail is valid";
    }
  }
?>
```

Example Explained

The example above has an input (email) sent to it using the "GET" method:

1. Check if an "email" input variable of the "GET" type exist
2. If the input variable exists, check if it is a valid e-mail address

28.7. Sanitize Input

Let's try cleaning up an URL sent from a form.

First we confirm that the input data we are looking for exists.

Then we sanitize the input data using the filter_input() function.

In the example below, the input variable "url" is sent to the PHP page:

```
<?php
if(!filter_has_var(INPUT_POST, "url"))
  {
  echo("Input type does not exist");
  }
else
  {
  $url = filter_input(INPUT_POST,
  "url", FILTER_SANITIZE_URL);
  }
?>
```

Example Explained

The example above has an input (url) sent to it using the "POST" method:

1. Check if the "url" input of the "POST" type exists
2. If the input variable exists, sanitize (take away invalid characters) and store it in the $url variable

If the input variable is a string like this "http://www.W3ååSchøøools.com/", the $url variable after the sanitizing will look like this:

http://www.W3Schools.com/

28.8. Filter Multiple Inputs

A form almost always consist of more than one input field. To avoid calling the filter_var or filter_input functions over and over, we can use the filter_var_array or the filter_input_array functions.

In this example we use the filter_input_array() function to filter three GET variables. The received GET variables is a name, an age and an e-mail address:

```
<?php
$filters = array
  (
  "name" => array
    (
    "filter"=>FILTER_SANITIZE_STRING
    ),
  "age" => array
    (
    "filter"=>FILTER_VALIDATE_INT,
    "options"=>array
      (
      "min_range"=>1,
      "max_range"=>120
      )
    ),
  "email"=> FILTER_VALIDATE_EMAIL
  );

$result = filter_input_array(INPUT_GET, $filters);

if (!$result["age"])
  {
  echo("Age must be a number between 1 and 120.<br>");
  }
elseif(!$result["email"])
  {
  echo("E-Mail is not valid.<br>");
  }
else
  {
  echo("User input is valid");
  }
?>
```

Example Explained

The example above has three inputs (name, age and email) sent to it using the "GET" method:

1. Set an array containing the name of input variables and the filters used on the specified input variables
2. Call the filter_input_array() function with the GET input variables and the array we just set
3. Check the "age" and "email" variables in the $result variable for invalid inputs. (If any of the input variables are invalid, that input variable will be FALSE after the filter_input_array() function)

The second parameter of the filter_input_array() function can be an array or a single filter ID.

If the parameter is a single filter ID all values in the input array are filtered by the specified filter.

If the parameter is an array it must follow these rules:

- Must be an associative array containing an input variable as an array key (like the "age" input variable)
- The array value must be a filter ID or an array specifying the filter, flags and options

28.9. Using Filter Callback

It is possible to call a user defined function and use it as a filter using the FILTER_CALLBACK filter. This way, we have full control of the data filtering.

You can create your own user defined function or use an existing PHP function

The function you wish to use to filter is specified the same way as an option is specified. In an associative array with the name "options"

In the example below, we use a user created function to convert all "_" to whitespaces:

```
<?php
function convertSpace($string)
{
return str_replace("_", " ", $string);
}

$string = "Peter_is_a_great_guy!";

echo filter_var($string, FILTER_CALLBACK,
array("options"=>"convertSpace"));
?>
```

The result from the code above should look like this:

```
Peter is a great guy!
```

Example Explained

The example above converts all "_" to whitespaces:

1. Create a function to replace "_" to whitespaces
2. Call the filter_var() function with the FILTER_CALLBACK filter and an array containing our function