Filtrado de datos de entrada en PHP

El filtrado de datos de entrada o inputs es un aspecto fundamental para evitar ataques como los XSS Cross-Site Scripting

Contenido modificable

Si ves errores o quieres modificar/añadir contenidos, puedes crear un pull request. Gracias

Indice de contenido

1. Validación con expresiones regulares
2. Especificar la codificación de caracteres
3. La extensión de filtrado de PHP

1. Validación con expresiones regulares

En cualquier formulario de cualquier aplicación siempre hay que validar los datos que introducen los usuarios. Una validación simple puede consistir en asegurarse de que se rellena un campo, se puede hacer fácilmente con empty():

```
<form action="index.php" method="post">
Nombre: <input type="text" name="nombre"><br>
Email: <input type="text" name="email"><br>
Enviar <input type="submit" name="enviar">
</form>
<?php
if(isset($_POST['enviar'])) {
    if(empty($_POST["nombre"])){
        echo "El nombre es requerido";
    } else {
    $nombre = $_POST["nombre"];
}
    // Más código...
}
```

Si el nombre no se rellena, el formulario devolverá "El nombre es requerido".

Para validar campos de una forma más precisa (y compleja) se utilizan expresiones regulares. Para ello PHP dispone de la función _preg_match()_ que exige dos parámetros obligatorios, la expresión regular y el string que tiene que comprobar. El siguiente es un ejemplo que asegura que el nombre son sólo sean letras y el email tenga un formato adecuado de emails (empleando el mismo formulario html anterior):

```
<?php
$errores = array();
if(isset($_POST['enviar'])) {
    // Requerimos el nombre:
    if (empty($_POST["nombre"])) {
        $errores[] = "El nombre es requerido <br>";
    } else {
        $nombre = $_POST["nombre"];
        // Queremos que el nombre de usuario sólo tenga letras
        if (!preg_match("/^[a-zA-Z]+/", $nombre)) {
            $errores[] = "Sólo se permiten letras como nombre de usuario <br>";
        }
    }
    // Requerimos el email también:
    if (empty($_POST["email"])) {
        $errores[] = "El email es requerido <br>";
    } else {
```

```
            if (!preg_match("/([\w\-]+\@[\w\-]+\.[\w\-]+)/", $email)) {
                $errores[] = "Formato de email incorrecto";
            }
        }
        if(empty($errores)) {
            echo "Nombre: $nombre <br>";
            echo "Email: $email <br>";
        } else {
            var_dump($errores);
        }
    }
```

Con las expresiones regulares se consigue un nivel de precisión muy alto. De todas formas, PHP tiene una extensión de filtrado con la que se pueden realizar filtros más rápidos para la validación y saneamiento de los datos. Si es posible evitar el uso de la expresión regular sin afectar al nivel de validez de los datos, mejor.

2. Especificar la codificación de caracteres

El data sanitization o la sanitización de datos consiste en modificar los inputs de entrada de forma que se eliminen algunos caracteres indeseables y se normalicen los datos para emplearlos o guardarlos de forma segura. Si por ejemplo tenemos un sistema de búsqueda en el que mostramos lo que se ha buscado al mostrar los resultados:

```
if(isset($_GET['query'])){
    echo '<p>Los resultados de tu búsqueda ', htmlspecialchars($_GET['query'], ENT_QUOTES), ' son: </p>'; // Código para mostrar
```

Con la función htmlspecialchars() reducimos el riesgo, pero siguen siendo posibles los ataques XSS. Con ENT_QUOTES se consigue que todas las comillas simples ' y dobles " se filtren, pero se pueden realizar ataques sin utilizar comillas. Tenemos que asegurarnos de utilizar la misma codificación de caracteres en la que se define el documento HTML para reducir más los riesgos:

```
// Definimos la cofificación en el HTTP header:
header('Content-Type: text/html; charset=UTF-8');
// Definimos la codificación empleada al convertir caracteres:
echo htmlspecialchars($_GET['query'], ENT_QUOTES, 'UTF-8');
```

3. La extensión de filtrado de PHP

La extensión de filtrado de PHP permite validar o sanitizar los datos, especialmente cuando vienen de fuentes externas (como usuarios). La extensión se divide entre filtros de validación, para comprobar que los datos cumplen ciertos requisitos, y en filtros de saneamiento, que limpiará los datos de forma que se eliminen los caracteres no deseados.

Las banderas u opciones de filtrado sirven para modificar el comportamiento en el saneamiento o la validación según las necesidades. Por ejemplo, si se pasa FILTER_FLAG_PATH_REQUIRED en una url, se exige que se precise una ruta concreta: /seccion en http://www.ejemplo.com/seccion.

Funciones de filtrado

- filter_list

```
array filter_list (void)
```

Devuelve un array con todos los filtros soportados.

- filter_var

```
mixed filter_var (mixed $variable [, int $filter = FILTER_DEFAULT [, mixed $options ]] )
```

Filtra una variable $variable con el filtro indicado en $filter. El filtro por defecto FILTER_DEFAULT equivale a FILTER_UNSAFE_RAW, y no filtra realmente nada.

```
// EJEMPLO 1
$var = filter_var('hola', FILTER_VALIDATE_BOOLEAN); // Valor: FALSE
// EJEMPLO 2
$opciones = array(
    'options' => array(
        'default' => 3, // Valor a devolver si falla el filtro
        'min_range' => 0
    ),
    'flags' => FILTER_FLAG_ALLOW_OCTAL,
);
$var = filter_var('12345', FILTER_VALIDATE_INT, $opciones); // Valor: 12345
```

Si se quiere validar el integer 0 con FILTER_VALIDATE_INT devuelve que "Integer is not valid". Para solucionar esto:

```
if (filter_var($int, FILTER_VALIDATE_INT) === 0 || !filter_var($int, FILTER_VALIDATE_INT) === false) {
    echo("El integer es válido");
} else {
    echo("El integer NO es válido");
}
```

Para asegurarnos de que un string es un email, con la extensión de filtrado de PHP podemos sanitizar y validar un email:

```
$email = "nombre@ejemplo.com";
// Primero eliminamos cualquier carácter que pueda dar problemas
$email = filter_var($email, FILTER_SANITIZE_EMAIL);
// Luego validamos el email
if (!filter_var($email, FILTER_VALIDATE_EMAIL) === false) {
    echo("$email es una dirección de email válida");
} else {
    echo("$email NO es una dirección de email válida");
}
```

También podemos sanitizar y validar una URL:

```
$url = "http://www.google.com";
// Eliminar cualquier carácter que pueda dar problemas
$url = filter_var($url, FILTER_SANITIZE_URL);
// Luego validamos la URL
if (!filter_var($url, FILTER_VALIDATE_URL) === false) {
    echo("$url es una URL válida");
} else {
    echo("$url no es una URL válida");
}
```

- filter_var_array

```
mixed filter_var_array (array $data [, mixed $definition [, bool $add_empty = true ]] )
```

Devuelve los datos del array $data y opcionalmente los filtra. $definition puede ser otro array que define filtros específicos para cada uno de los elementos de $data, o simplemente una constante de filtro como FILTER_VALIDATE_URL, por ejemplo.

```
$inputs = ["<a>perro</a>", "gato"];
$animales = filter_var_array($inputs, FILTER_SANITIZE_STRING);
var_dump($inputs);
/*
 Devuelve
array (size=2)
  0 => string '<a>perro</a>' (length=38)
  1 => string 'gato' (length=4)
 */
var_dump($animales);
/*
 Devuelve
array (size=2)
  0 => string 'perro' (length=5)
  1 => string 'gato' (length=4)
*/
```

- filter_input

```
mixed filter_input (int $type, string $variable_name [, int $filter = FILTER_DEFAULT [, mixed $options ]] )
```

Toma una variable externa de un determinado tipo y opcionalmente la filtra. Estas variables pueden venir de $_GET, $_POST, $_COOKIE, $_SERVER, $_ENV, por lo que type deberá ser uno de los siguientes valores: INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER o INPUT_ENV. Los filtros y opciones funcionan de la misma forma que en las funciones anteriores.

```
$busqueda = filter_input(INPUT_GET, 'buscar', FILTER_SANITIZE_SPECIAL_CHARS);
$url = filter_input(INPUT_GET, 'buscar', FILTER_SANITIZE_ENCODED);
echo "Has buscado $busqueda.\n";
echo "Buscar otra vez";
// Ejemplo con "5f F$ "·$ %!"
// Busqueda: 5f F$ "·$ %!
// URL: 5f%20F%24%20"·%24%20%25%21
```

- filter_input_array

```
mixed filter_input_array (int $type [, mixed $definition [, bool $add_empty = true ]] )
```

```
<form action="index.php" method="post">
    Nombre de usuario: <input type="text" name="usuario"><br>
    Email: <input type="text" name="email"><br>
    Enviar <input type="submit" name="enviar">
</form>
<?php
if(isset($_POST['enviar'])) {
$usuario = filter_input_array(INPUT_POST, FILTER_SANITIZE_ENCODED);
var_dump($usuario);
}
/* Si pasamos:
  usuario = diego<a href>
  email = diego@ejemplo.com'$"
  Devuelve:
array (size=3)
  'usuario' => string 'diego%3Ca%20href%3E' (length=19)
  'email' => string 'diego%40ejemplo.com%27%24%22' (length=28)
  'enviar' => string 'Submit' (length=6)
*/
```

- filter_has_var

```
bool filter_has_var ( int $type, string $variable_name)
```

Comprueba si existe una variable en un tipo concreto (GET, POST, COOKIE...)

```
$_GET['hey'] = "hola";
echo filter_has_var(INPUT_GET, 'hey') ? 'Si' : 'No';
// No devolverá 'Sí' a no ser que accedas desde URL con
// el parámetro hey y el valor hola:
// http://ejemplo.com/index.php?hey=hola
```

Constantes de filtrado

Las constantes de filtrado son:

Constante	Validación
FILTER_VALIDATE_BOOLEAN	Valida un booleano
FILTER_VALIDATE_EMAIL	Valida una dirección de email
FILTER_VALIDATE_FLOAT	Valida un float
FILTER_VALIDATE_INT	Valida un integer
FILTER_VALIDATE_IP	Valida una dirección IP
FILTER_VALIDATE_REGEXP	Valida una expresión regular
FILTER_VALIDATE_URL	Valida una URL
FILTER_SANITIZE_EMAIL	Elimina caracteres peligrosos de una dirección de email
FILTER_SANITIZE_ENCODED	Elimina caracteres especiales
FILTER_SANITIZE_MAGIC_QUOTES	Hace lo mismo que la función addslashes()
FILTER_SANITIZE_NUMBER_FLOAT	Elimina todo menos dígitos, +, - y opcionalmente .,eE
FILTER_SANITIZE_NUMBER_INT	Elimina todo menos dígitos
FILTER_SANITIZE_SPECIAL_CHARS	Elimina caracteres especiales
FILTER_SANITIZE_STRING	Elimina etiquetas y caracteres especiales de un string
FILTER_SANITIZE_STRIPPED	Alias de FILTER_SANITIZE_STRING
FILTER_SANITIZE_URL	Elimina caracteres peligrosos de una URL
FILTER_UNSAFE_RAW	No hace nada, igual que FILTER_DEFAULT
FILTER_CALLBACK	Llama a una función callback definida por el usuario