PHP-7-1

Sanear datos

Seguridad en las entradas

- En las páginas de internet son muy empleados los formularios en sus diversos usos, para autentificarse, enviar correos, hacer búsquedas, etc.
- Estos hacen las páginas mucho más atractivas, útiles e interactivas para los lectores, pero traen un riesgo enorme para la seguridad de un sitio web, ya que son la puerta de entrada de la gran mayoría de los ataques, y la inyección de código.

Seguridad en las entradas

- Existen varias soluciones para resolver el problema de cómo sanear los datos de usuario pero, en opinión de muchos expertos, el sistema más recomendable es hacer un doble saneamiento:

– Validando antes de almacenar los inputs en la base de datos.

– Validando antes de enviar los datos a la salida, en su procesamiento.

Seguridad en las entradas

- En la práctica, este sistema de doble saneamiento lo haremos en los siguientes pasos:
  - Antes de almacenar los inputs en la base de datos
    - Lo más recomendable es limitarse a validar los datos, sin intentar corregirlos. Si validan se aceptan, si no validan los rechazaremos.

Seguridad en las entradas

- Codificación de caracteres

Comprobaremos si la codificación de caracteres es la adecuada (UTF-8).

- Caracteres

Nos aseguraremos de que los datos sólo contienen caracteres permitidos.

- Tipo de datos

Comprobaremos que el tipo de dato es el esperado: los teléfonos deben contener sólo números, etc.

Seguridad en las entradas

- Longitud (min/max)

Es recomendable restringir la longitud de los input, ya que así limitamos las opciones de los atacantes.

- Formato de los datos

Comprobaremos que la estructura de los datos es consistente con lo esperado: los teléfonos deben parecer teléfonos, los email deben parecer email, etc.

Seguridad en las entradas

- Con lo expuesto, la regla de seguridad elemental que debemos tener en cuenta es proteger todas las puertas que abrimos en nuestras páginas, que son las entradas en los formularios o cualquier elemento que utilice la etiqueta <input>.
- Para ello debemos limpiar de código susceptible de interpretación, las cadenas recibidas en las entradas.

Eliminar etiquetas: strip_tags($cadena)

- Una manera de resolver este problema se obtiene utilizando la función **strip_tags($cadena)** que devuelve la cadena sin etiquetas.
- Hay que tener en cuenta que esta función elimina cualquier cosa que se interprete como una etiqueta, es decir, que empiece por "<".
- **Descripción**
  - string **strip_tags** (string $cadena [, string $etiquetas_ permitidas]).
- Esta función es segura en modo binario desde la versión de PHP 5.0.0

Eliminar etiquetas: strip_tags($cadena)

- Esta función intenta eliminar todas las etiquetas HTML y PHP de la cadena dada. Puede usar el parámetro opcional para especificar las etiquetas que no deben eliminarse.
- los comentarios HTML también se eliminan. Esta característica es intrínseca de la función y no se puede evitar mediante el parámetro *etiquetas_permitidas*

```php
<?php
$texto = '<p>Parrafo de prueba.</p><!-- Comentario -->
<strong> Mas texto</strong>';
echo strip_tags($texto);
echo "\n";

// Se permite la etiqueta <p>
echo strip_tags($texto, '<p>');
?>
```

```
Parrafo de prueba. Mas texto

Parrafo de prueba.

Mas texto
```

Eliminar espacios en inicio y fin de la cadena: trim

- **trim (string $cadena)**
  - Si los usuarios escriben por error espacios en blanco al principio o al final de las cajas de texto, y el programa PHP no tiene en cuenta esa posibilidad se pueden obtener resultados inesperados.
  - Este problema se puede resolver utilizando la función **trim($cadena),** que elimina los espacios en blanco iniciales y finales y devuelve la cadena sin esos espacio.

Utilización de variables

- Según lo expuesto y con lo visto hasta ahora, al incluir cualquier referencia a $_REQUEST[control] habría que :
    a. comprobar que el control esté definido con la función isset()
    b. eliminar las etiquetas con la función strip_tags()
    c. eliminar los espacios en blanco iniciales y finales con la función trim()
- Una manera de hacerlo sin complicar excesivamente el programa es guardar los valores de la matriz $_REQUEST en variables y realizar todas las comprobaciones al definir esas variables, y en el resto del código utilizaremos la variable en lugar del elemento de la matriz $_REQUEST

```php
<?php
if (isset($_REQUEST["enviar"])){

print "<pre>"; print_r($_REQUEST); print "</pre>\n";
if (isset($_REQUEST["texto"])) {
    $nombre = trim(strip_tags($_REQUEST["texto"]));
} else {
    $nombre = "";
}
if ($nombre == "") {
    print "<p>No ha escrito ningún texto</p>\n";
} else {
    print "<p>Su texto es $nombre</p>\n";
}
}
else{
?>
<form action 'validar3.php>
<input type='text' name='texto'/>
<input type='submit' name='enviar'/>
</form>

<?php
;}
?>
```

```
Array
(
    [texto] => hola mundo
    [enviar] => Enviar consulta
)
```

Su nombre es hola mundo

Salida de datos

- Cuando los datos recogidos les vamos a incorporar a una página web, hay que tener cuidado en algunos caracteres especiales, tales como
  - el carácter& (ampersan) que es el comienzo de llamamiento a entidades de caracteres.
  - el carácter " (comillas), ya que si una cadena se escribe dentro de otras comillas (por ejemplo, en el atributo value de una etiqueta input), la página no se verá correctamente y además no será válida.

Salida de datos

- Además de los caracteres ampersand (&) y comillas ("), también podrían dar problemas los caracteres comillas simples (') o las desigualdades (< y >).
- Una solucion sería sustituir cada uno de ellos con la función str_replace().
  - str_replace('&', '&', $nombre);
  - str_replace('« ' , '"', $nombre);
- Pero para evitar tener que ir sustituyendo cada carácter susceptible de interpretación por parte del interprete php, existe la siguiente solución a nivel general que **convierte todos estos caracteres especiales a entidades HTML.**

Formato:

htmlspecialchars($cadena_a_covertir, $flags, $character_set, $double_encode)

- **string**: La función nos devuelve una cadena de texto.

- **$cadena_a_convertir (Obligatorio)**: Es la cadena que queremos pasarle a la función para convertir los caracteres a HTML.

- **$flags (Opcional)**: Especifica cómo manejar los caraceres, codificación no válida y el tipo de documento (doctype) utilizado.

  - Las opciones de codificación de comillas son:
    - **ENT_COMPAT**: Codifica sólamente las comillas dobles.
    - **ENT_QUOTES**: Codifica sólamente las comillas simples y dobles.
    - **ENT_NOQUOTES**: No se codifica ninguna comilla.

- **$character set (Opcional)**: Tipo de codificación utilizada.

- **$double_encode (Opcional)**: TRUE o FALSE, Si está a TRUE la función convierte también las otras entidades HTML del string, por defecto es valor TRUE.

```php
<?php
if (isset($_REQUEST["enviar"])){

print "<pre>"; print_r($_REQUEST); print "</pre>\n";
if (isset($_REQUEST["texto"])) {
    $nombre = htmlspecialchars($_REQUEST["texto"], ENT_QUOTES,"utf-8" );
} else {
    $nombre = "";
}
if ($nombre == "") {
    print "<p>No ha escrito ningún texto</p>\n";
} else {
    print "<p>Su texto es $nombre</p>\n";
}
}
else{
?>
<form action 'validar3.php>
<input type='text' name='texto'/>
<input type='submit' name='enviar'/>
</form>
<?php
;}
?>
```

```
Array
(
    [texto] =>  "hola Mundo"
    [enviar] => Enviar consulta
)
```

Su texto es "hola Mundo"

htmlspecialchars($cadena_a_covertir, $flags, $character_set, $double_encode)

- Este ejemplo tiene el inconveniente de que solo transforma los caracteres especiales de html a texto plano, si quiero evitar los blancos del principio y final de la cadena debo anidar la función trim
- Y si quiero quitar físicamente el etiquetado debo también aplicar strip_tags() , pero con cuidado porque que se eliminará todo lo que esté entre marcas (<>).

```php
<?php
if (isset($_REQUEST["enviar"])){

print "<pre>"; print_r($_REQUEST); print "</pre>\n";
if (isset($_REQUEST["texto"])) {
    $nombre = htmlspecialchars(trim(strip_tags($_REQUEST["texto"])), ENT_QUOTES,"utf-8" );
} else {
    $nombre = "";
}
if ($nombre == "") {
    print "<p>No ha escrito ningún texto</p>\n";
} else {
    print "<p>Su texto es $nombre</p>\n";
}
}
else{
?>
<form action 'validar3.php>
<input type='text' name='texto'/>
<input type='submit' name='enviar'/>
</form>
<?php
;}
?>
```

```
Array
(
    [texto] =>  "hola Mundo"
    [enviar] => Enviar consulta
)
```

Su texto es "hola Mundo"

```php
<?php
if (isset($_REQUEST["enviar"])){

 print "<pre>"; print_r($_REQUEST); print "</pre>\n";
if (isset($_REQUEST["texto"])) {



    $nombre = strip_tags(trim(htmlspecialchars($_REQUEST["texto"], ENT_QUOTES, "UTF-8")));
 } else {
    $nombre = "";
}
if ($nombre == "") {
    print "<p>No ha escrito ningún texto</p>\n";
 } else {
    print "<p>Su texto es $nombre</p>\n";
}
}
else{
?>
 <form action 'validar3.php>
 <input type='text' name='texto'/>
 <input type='submit' name='enviar'/>
</form>
<?php
;}
 ?>
```

```
Array
(
    [texto] =>  "hola Mundo"
    [enviar] => Enviar consulta
)
```

Su texto es "hola Mundo"

Funciones de recogida de datos

- La forma más cómoda de tener en cuenta todos los aspectos comentados en los puntos anteriores es definir una función recogida():
  - La función tendrá como argumento el nombre del control que se quiere recibir y devolverá el valor recibido (o una cadena vacía si el control no se ha recibido).
  - Para tratar los datos recibidos se aplicarán únicamente las funciones **htmlspecialchars()** y **trim().**

```
:trim(htmlspecialchars($_REQUEST[$var], ENT_QUOTES, "UTF-8"))
```

Funciones de recogida de datos

- De esta manera, si se reciben etiquetas (<>), las etiquetas no se borrarán, sino que se conservarán como texto. Si se quisieran borrar las etiquetas, habría que aplicar las funciones **strip_tags()**, **trim()**, y **htmlspecialchars()**:

```
htmlspecialchars(trim(strip_tags($_REQUEST[$var])), ENT_QUOTES, "UTF-8")
```

- Una vez definida la función, al comienzo del programa se almacenarán en variables los datos devueltos por la función .
- En el resto del programa se trabaja con las variables.

Funciones de recogida de datos

- A continuación se define una función a modo de ejemplo que automatice la recogida de un dato.

```php
<?php
// FUNCIÓN DE RECOGIDA DE UN DATO
function recoge($var)
{
    $tmp = (isset($_REQUEST[$var]))
        ? trim(htmlspecialchars($_REQUEST[$var], ENT_QUOTES, "UTF-8"
        : "";
    return $tmp;
}

//COMPROBACIÓN DE LLEGADA DE FORMULARIO

if (isset($_REQUEST["enviar"])){

print "<pre>"; print_r($_REQUEST); print "</pre>\n";

// EJEMPLO DE USO DE LA FUNCIÓN DE RECOGIDA SANEADA
$nombre = recoge("texto");

if ($nombre == "") {
    print "<p>No ha escrito ningún nombre</p>";
} else {
    print "<p>Su nombre es $nombre</p>\n";
}
}
```

ASDFG Enviar consulta

```
Array
(
    [texto] =>

ASDFG


    [enviar] => Enviar consulta
)
```

Funciones de recogida de datos

- Se propone realizar una función para sanear los datos recogidos en un vector, para ello hay que tener en cuenta que los índices al poder ser asociativos pueden ser también susceptibles de entradas inesperadas.