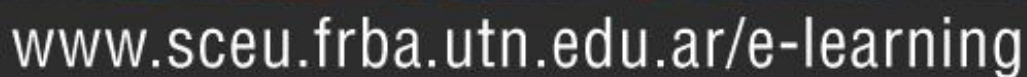




Centro de e-Learning

EXPERTO UNIVERISTARIO EN MySQL Y PHP NIVEL AVANZADO





Módulo II

PHP AVANZADO



Seguridad.



Presentación de la Unidad:

En el apunte Seguridad que forma parte del material de la presente unidad hemos visto algunas consideraciones a tener en cuenta a la hora de definir nuestra configuración de PHP.

También hemos hablado sobre los ataques mas comunes que puede sufrir un sitio web como XSS y SQL Injection.

A continuación veremos dos ejemplos prácticos para evitar este tipo de ataques.

Además hacemos mención de la nueva API de PHP 5.5 para codificar contraseñas.



Objetivos:

- ❖ Preparar nuestro sitio web para evitar que sea atacado a través de Cross Site Scripting (XSS).
- ❖ Preparar nuestro sitio web para evitar que sea atacado a través de SQL Injection.
- ❖ Hacer uso de la API para codificar contraseñas en PHP 5.5



Temario:

Ejemplo 1: Filtro para evitar XSS

Ejemplo 2: Filtro para evitar SQL Injection

Uso de la API para codificar contraseñas de PHP 5.5



CONSIGNAS PARA EL APRENDIZAJE COLABORATIVO

En esta Unidad los participantes se encontrarán con diferentes tipos de consignas que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

1. Los foros asociados a cada una de las unidades.
2. La Web 2.0.
3. Los contextos de desempeño de los participantes.

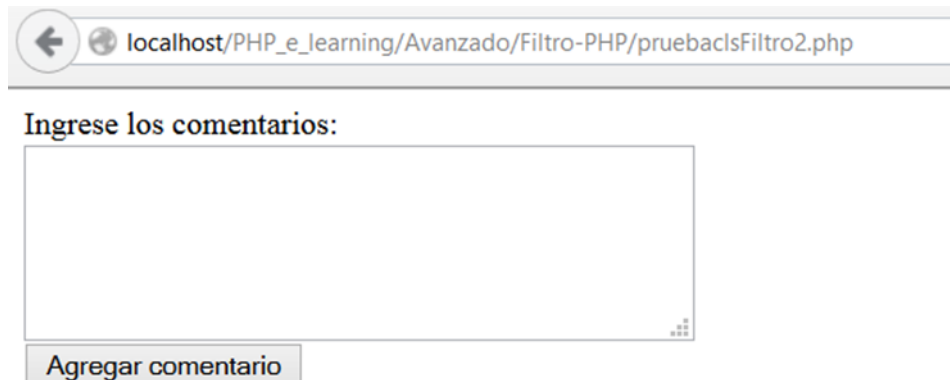


Es importante que todos los participantes realicen las actividades sugeridas y compartan en los foros los resultados obtenidos.

EJEMPLO 1:

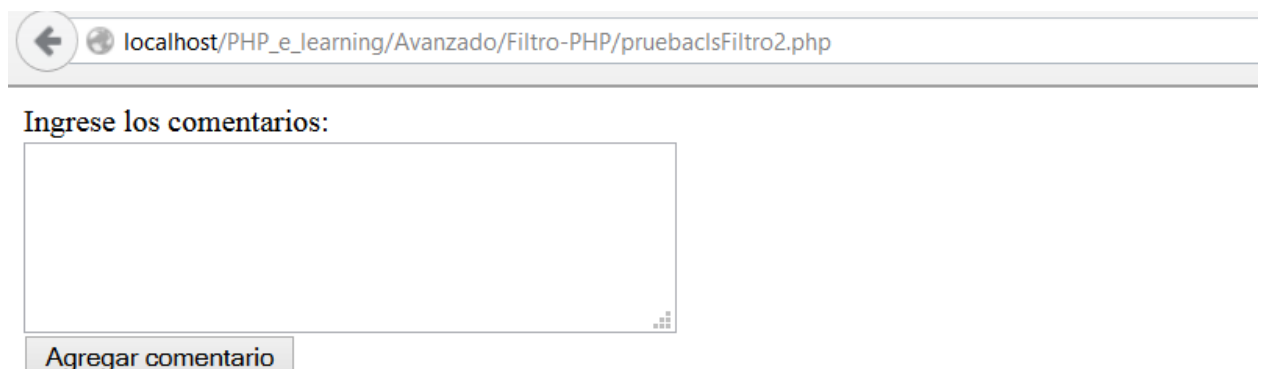
En la carpeta Filtro-PHP adjunta a los ejemplos de la unidad van a encontrar un archivo pruebaclsfiltro2.php

Si lo ejecutamos en el navegador vamos a ver un textarea habilitado para dejar nuestros comentarios.



Imaginen que esta es una típica página de algún periódico o blog en la cual a continuación de un artículo presentado habilita a los usuarios a dejar su comentario. Éste es introducido por el usuario en el textarea y es mostrado inmediatamente a continuación del artículo.

Entonces una vez ingresado el comentario el mismo es mostrado de la siguiente manera:



Muy buena información, el artículo está muy bien redactado, muy preciso y claro. Felicitaciones

Que sucederia si en lugar de escribir un comentario como el que acabamos de hacer

Centro de Formación, Investigación y Desarrollo de Soluciones de e-Learning.

UTN - FRBA. Secretaría de Cultura y Extensión Universitaria

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148 // e-learning@sceu.frba.utn.edu.ar

algun usuario introdujera algo así:

```
<script type="text/javascript">window.location="http://www.google.com.ar"</script>
```

Esto ocasionaría que todos los visitantes a esta página sean redireccionados a otro sitio. Para evitar que este tipo de acciones sucedan en nuestra página lo que se puede hacer es filtrar los campos de ingreso de datos y descartando el código que produce el daño.

Para ello creamos una clase que reciba los datos a filtrar y devuelve los datos filtrados.

Esto es lo que hace la clase `clsFiltro.php`. La misma tiene una propiedad con las etiquetas permitidas:

```
private $tagsPermitidos = array('<b>','<p>','<br>');
```

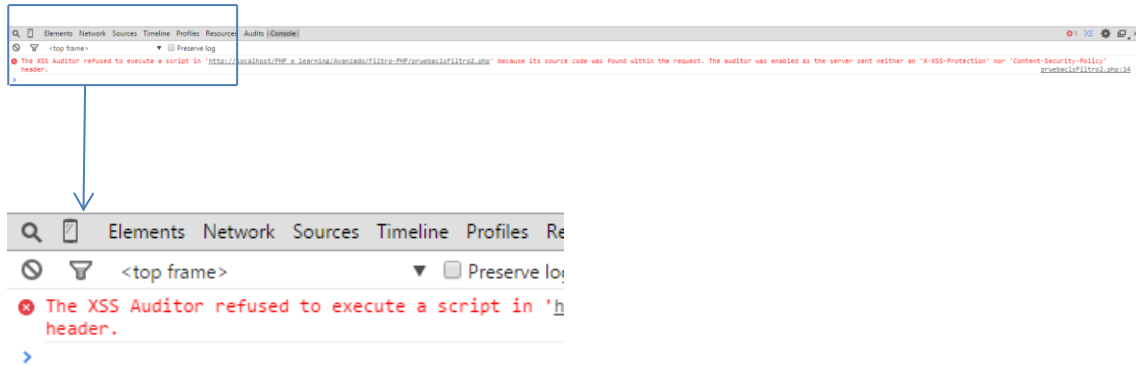
En ella definimos las etiquetas permitidas, pero puede ser que haya alguna etiqueta que no la tenemos permitida por defecto pero que querramos permitirle eventualmente entonces al constructor se le pueden enviar esas etiquetas. Cuando instanciamos la clase el constructor hace la combinación de ambas etiquetas, las definidas por defecto mas las permitidas momentaneamente.

```
public function __construct($tags=false){  
    if (!empty($tags) && is_array($tags)){  
        $this->tagsPermitidos = array_merge($this->tagsPermitidos,$tags);  
    }  
}
```

Lo que tenemos que hacer entonces es llamar al método `filtrar` para filtrar los datos ingresados. El método evalúa si el parámetro recibido es un array. Si no lo es llama al método `procesar()` con los datos recibidos, que es el método que filtra las etiquetas no permitidas. Y si es un array llama al método `procesar()` por cada uno de los elementos del array y lo devuelve filtrado.

Pueden probar el ejemplo `pruebaclsFiltro2.php` comentando y descomentando la utilización del filtro e ingresando el código javascript en el textarea de comentarios.

Nota: Van a poder comprobarlo en Firefox ya que en Chrome el navegador por si solo detecta el ataque XSS. Si prueban el script sin la utilización del filtro en Chrome habiendo habilitado previamente las herramientas para el desarrollador verán en la consola lo siguiente:



EJEMPLO 2:

Sobre la misma idea de filtrar los datos para evitar ataques XSS podemos utilizar también un filtro para evitar ataques de inyección SQL.

Tomemos el ejemplo del login en inicio.html

Si ya estas registrado podes ingresar aqui:

Usuario:

Password:

Ingresar

Ingresando nuestro usuario y password podemos acceder al contenido del sistema, si el usuario y/o password son incorrectos nos rechaza el ingreso mostrándonos el mensaje de usuario y/o password invalidos.

Pero que sucedería si ingresáramos

Usuario: pepe

Password: pepe' or 1 = '1

El usuario y password ingresados serán utilizados en el query que figura en login.php:

```
"SELECT * FROM usuarios WHERE usuario = '$_POST['usuario']'" AND password = '$_POST['password']'" "
```

Lo que haría que nuestro query quede de la siguiente manera:

```
"SELECT * FROM usuarios WHERE usuario = 'pepe' AND password = 'pepe' or 1 = '1' "
```

Este query selecciona los usuarios siempre que usuario sea igual a pepe y password sea

Centro de Formación, Investigación y Desarrollo de Soluciones de e-Learning.

UTN - FRBA. Secretaría de Cultura y Extensión Universitaria

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148 // e-learning@sceu.frba.utn.edu.ar



igual a pepe o 1 sea igual a 1. Lo que dará como resultado todos los usuarios ya que la condición $1=1$ es siempre verdadera.

Lo que podemos hacer entonces, entre otras cosas, para evitar este tipo de ataques es filtrar los datos que se ingresan.

Agregamos en el script login.php la inclusión de la clase clsFiltroSql.php, la instanciamos y llamamos al método filtrar para filtrar los datos que se ingresan.

```
include ('clsFiltroSql.php');
```

```
$filtro = new clsFiltroSql();
```

```
$_POST = $filtro->filtrar($_POST);
```

Al igual que en la clase clsFiltro que utilizamos para evitar los ataques XSS, el método filtrar está preparado para recibir un string o un array, con lo cual podemos llamarlo pasándole el array completo de POST de una sola vez.

NUEVA API PARA CODIFICAR CONTRASEÑAS DE PHP 5.5

La novedad más importante de PHP 5.5 relacionada con la seguridad es la creación de una nueva API para codificar y verificar contraseñas. Además de ser muy fácil de utilizar, esta API sigue todas las buenas prácticas de seguridad recomendadas para aplicaciones web modernas.

Internamente la API utiliza la función `crypt()` y está disponible desde la versión 5.5.0 de PHP. Si utilizas una versión anterior de PHP, siempre que sea igual o superior a 5.3.7, existe una librería con las mismas funcionalidades que la nueva API:

github.com/ircmaxell/password_compat.

CODIFICANDO CONTRASEÑAS

La función más importante de la nueva API es `password_hash()`, que codifica la contraseña que le pases con el algoritmo indicado:

```
$password = password_hash('admin1234', PASSWORD_DEFAULT);  
// $password =  
$2y$10$a8ThA3xOuWB6QPeFL/VRZeynSxoT6chW4RS1uh5l0xOBEePFnUNpG
```

El primer argumento de la función es la contraseña original sin codificar y el segundo argumento debe ser una de las dos siguientes constantes:

- `PASSWORD_DEFAULT`, codifica la contraseña utilizando el algoritmo `bcrypt` y el resultado es una cadena de 60 caracteres de longitud, cuyos primeros caracteres son `$2y$10$`. El algoritmo utilizado y la longitud de la contraseña codificada cambiarán en las próximas versiones de PHP, cuando se añadan algoritmos todavía más seguros. Si guardas las contraseñas en una base de datos, la recomendación es que reserves 255 caracteres para ello y no los 60 que se pueden utilizar actualmente.
- `PASSWORD_BCRYPT`, a pesar de su nombre, codifica la contraseña utilizando el algoritmo `CRYPT_BLOWFISH`. Al igual que en el caso anterior, la contraseña codificada ocupa 60 caracteres en total, siendo los primeros caracteres `$2y$`.

```
$password = password_hash('admin1234', PASSWORD_DEFAULT);  
// $password =  
$2y$10$a8ThA3xOuWB6QPeFL/VRZeynSxoT6chW4RS1uh5l0xOBEePFnUNpG
```

```
$password = password_hash('admin1234', PASSWORD_BCRYPT);  
// $password =  
$2y$10$1Y2nAYh7gVThKyL2H/FkIuAqU0Z4ZZNUz3rqFNSbq7RiVoyWsXM5a
```

Aunque te cueste creerlo, estas funciones para codificar contraseñas se han diseñado a propósito para que sean muy lentas y para que siempre tarden el mismo tiempo en

Centro de Formación, Investigación y Desarrollo de Soluciones de e-Learning.

UTN - FRBA. Secretaría de Cultura y Extensión Universitaria

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148 // e-learning@sceu.frba.utn.edu.ar

obtener el resultado. De esta forma se evitan los ataques de seguridad conocidos como [timing attack](#).

El tiempo empleado en codificar una contraseña se denomina "*coste*" y se puede configurar mediante el tercer argumento opcional de la función `password_hash()`. El coste por defecto es 10 (por eso el prefijo de las contraseñas anteriores es \$2y\$10\$) y su valor debe estar comprendido entre 04 y 31. Si tu servidor es muy potente, puedes subir el coste de la siguiente manera:

```
$password = password_hash('admin1234', PASSWORD_DEFAULT);  
// $password =  
$2y$10$a8ThA3xOuWB6QPeFL/VRZeynSxoT6chW4RS1uh5l0xOBEePFnUNpG  
  
$password = password_hash('admin1234', PASSWORD_DEFAULT, array('cost'  
=> 12));  
// $password =  
$2y$12$hrxMLCk3ZfKNEcY1lpTpMOD09TnHO5v1Jp5CR4xYud7a4RACWoRFm  
  
$password = password_hash('admin1234', PASSWORD_DEFAULT, array('cost'  
=> 20));  
// $password =  
$2y$20$UX1pRARpFTNsUn8czY8AE.8OakctlMk6.or6OmQnQ.mpyzHvm29ki
```

La recomendación oficial para elegir el coste más adecuado para tu servidor es que el tiempo empleado en codificar una contraseña sea entre 0.1 y 0.5 segundos.

Además del coste, la función `password_hash()` también permite configurar la opción `salt`, que es el valor inicial que se utiliza para codificar la contraseña. Se recomienda no establecer este valor a mano y dejar que sea PHP quien lo calcule automáticamente. Si quieres saber cómo establecer este valor a mano, consulta la [documentación de password_hash\(\)](#).

Si hasta ahora guardabas en la base de datos las contraseñas originales porque no sabías cómo codificarlas, ya no tienes excusa. Gracias a la nueva API para codificar las contraseñas, hacer las cosas bien desde el punto de vista de la seguridad cuesta todavía menos tiempo y esfuerzo que hacerlas mal.

COMPROBANDO CONTRASEÑAS

Codificar las contraseñas antes de guardarlas en la base de datos es sólo la primera parte del problema. Después, cada vez que un usuario quiera conectarse a tu aplicación, hay que comprobar que la contraseña proporcionada es efectivamente la misma que se guardó codificada.

La nueva función `password_verify()` simplifica este trabajo al máximo, ya que solamente hay que pasarle como primer argumento la contraseña del usuario y como

Centro de Formación, Investigación y Desarrollo de Soluciones de e-Learning.

UTN - FRBA. Secretaría de Cultura y Extensión Universitaria

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148 // e-learning@sceu.frba.utn.edu.ar

segundo argumento la contraseña codificada:

```
$original = 'admin1234';  
$codificado =  
'$2y$10$a8ThA3xOuWB6QPeFL/VRZeynSxoT6chW4RS1uh5l0xOBEePFnUNpG';  
  
$iguales = password_verify($original, $codificado);  
  
if ($iguales) {  
    echo 'Puedes pasar a la zona privada';  
} else {  
    echo 'La contraseña indicada no es correcta';  
}
```

Como las contraseñas codificadas por `password_hash()` ya contienen en su interior el coste y el salt utilizados para codificar la contraseña original, no hay que indicar ningún parámetro más a la función `password_verify()`.

OTRAS UTILIDADES

La nueva API para codificar contraseñas incluye otras dos funciones menos importantes. La primera es `password_get_info()`, que devuelve un array con información de la contraseña codificada que se le pasa como argumento:

```
$info =  
password_get_info('$2y$20$UX1pRArPfTNsUn8czY8AE.8OakctlMk6.or6OmQnQ.mp  
zyHvm29ki');  
  
$info = array(  
    'algo'      => 1,          // identificador del algoritmo utilizado  
    'algoName' => 'bcrypt',    // nombre del algoritmo utilizado  
    'options'   => array(  
        'cost' => 20           // coste utilizado en la codificación  
    )  
);
```

Si en tu aplicación modificas con el tiempo el algoritmo o las opciones para codificar las contraseñas, necesitarás comprobar si una determinada contraseña debe ser recodificada o si cumple las nuevas opciones. Para ello puedes utilizar la función `password_needs_rehash()`, pasando la contraseña codificada como primer argumento, el algoritmo que se debe utilizar como segundo argumento y opcionalmente, las opciones del algoritmo como tercer argumento.

En el siguiente ejemplo, se pasa una contraseña codificada con el algoritmo por defecto y un coste de 10. Como las características deseadas son el algoritmo por defecto y un coste de 20, el resultado de la función `password_needs_rehash()` será `true`:



```
$hayQueRecodificar = password_needs_rehash(  
    '$2y$10$a8ThA3xOuWB6QPeFL/VRZeynSxoT6chW4RS1uh5l0xOBEEpFnUNpG',  
    PASSWORD_DEFAULT,  
    array('cost' => 20)  
);  
  
// $hayQueRecodificar = true
```