

PHP SEGURO

Filtrando datos...

CHEBYTE

chebyte at gmail.com

La programación segura de PHP es esencial para no comprometer la seguridad de su servidor. Es una práctica que no hay que dejar de realizar.

Hay varias metodologías existentes para llevar un desarrollo seguro de su aplicación pero a lo largo de este artículo solo nos concentraremos en ver el filtrado de datos.

El filtrado de datos, es uno de los puntos de más importancia a la hora de programar una aplicación segura, y la que más impacto tiene a la hora de analizar la seguridad, debido que estos datos pueden ser manipulados por cualquier usuario.

Cuando me refiero a filtros de datos, me refiero a someter las variables como `$_GET`, `$_POST`,..., a diferentes pruebas, armadas según nuestras necesidades, para que cumplan nuestros requisitos.

En este artículo también se hablará de algunas técnicas de ataques con ejemplos pero básicos, ya que solo se intenta ilustrar al usuario los riesgos de estos.

Conceptos

Voy a definir algunos conceptos que se debería tener en cuenta, para el posterior entendimiento de este artículo.

Expresiones Regulares: Las expresiones regulares son una serie de caracteres que forman un patrón, normalmente representativo de otro grupo de caracteres mayor, de tal forma que podemos comparar el patrón con otro conjunto de caracteres para ver las coincidencias. Las expresiones regulares están disponibles en casi cualquier lenguaje de programación, pero aunque su sintaxis es relativamente uniforme, cada lenguaje usa su propio *dialecto*.

En php las funciones que soportan expresiones regulares son: `ereg()`, `ereg_replace()`, `preg_match()`, `eregi()` `eregi_replace()`

XSS: (Cross Site Scripting) - tipo de vulnerabilidad surgida como consecuencia de errores de filtrado de las entradas del usuario en aplicaciones web.

Se trata de usar diversas técnicas para inyectar código de marcas (html), código ejecutable en la máquina cliente (Javascript/VBScript/ActiveX) o código ejecutable en el servidor (PHP/ASP) en las entradas de aplicaciones web con el fin de conseguir muy diversos objetivos limitados por la capacidad del lenguaje inyectado para vulnerar al cliente o al servidor de la aplicación web.

SQL Injection: Error que se puede producir en páginas web, aunque puede producirse en cualquier programa que haga una llamada a una base de datos SQL, cuando se realiza una consulta usando como parte de ella datos que no han sido correctamente tratados (normalmente no escapando los caracteres delimitadores ' o ") pudiendo lograr que la consulta produzca resultados no previstos. Dependiendo de diversos factores, el problema puede suponer desde autorizar un acceso inválido hasta obtener o modificar todos los datos de la base de datos, e incluso en casos extremos, lograr ejecutar un código no previsto en el servidor.

Técnicas de filtrado

* Una forma interesante y limpia de trabajar con filtros de datos, sería someter las variables de entrada a varios procesos de limpieza (testeo), al pasar las etapas correctamente se podría generar un array (clean), con los datos ya verificados.

Ejemplo

Test

```
if(preg_match("/^[a-z0-9-]{4,12}$/i",$_POST['user_login'])) {  
    clean['user_login'] = $_POST['user_login'];  
}  
else{  
  
    ERROR  
  
}
```

Expresión regular que controla si el nombre de usuario ingresado es una cadena que va desde 4 a 12 caracteres, y contiene solo caracteres de "a" hasta la "z" y de "0" a "9".

* Limitar las entradas(forms) solamente a los tipos de datos que deba recibir.

Por ejemplo en un form donde se deba ingresar el teléfono de un usuario, solo se debe permitir números(integer).

Ejemplo

```
Test
    clean=array();
    $_POST['user_telephone']; // variable a filtrar
    if(is_number($_POST['user_telephone'])) {
    $clean['user_telephone'] = $_POST['user_telephone']; //Variable
válida
    }
    .
    .
    .
```

* El uso de funciones para expresiones regulares (ereg,preg_match,etc.) para filtrar los datos,es muy recomendado.

Ejemplo

Test

```
if (!(eregi("^([_a-z0-9-]+)(\\.[_a-z0-9-]+)*@([a-z0-9-]+)(\\.[a-z0-9-]+)*\\.[a-z]{2,4}$", $_POST['email']))) {
```

ERROR

```
}
```

Expresión regular que verifica si el formato del email ingresado es válido.

* Limpiar las entradas de caracteres no deseados.

(*)Ejemplo

```
//FUNCION
```

```
function htmlclean($input) {
```

```
    $sb_convert = $input;
```

```
    $sb_input = array("<", ">", "(", ")");
```

```
    $sb_output = array("&lt;", "&gt;", "&#40;", "&#41;");
```

```
    $output = str_replace($sb_input, $sb_output, $sb_convert);
```

```
    return $output;
```

```
}
```

```
//////////
```

Código principal

```
clean=array();
```

```
$clean['user_name'] = htmlclean($_POST['user_name']);
```

Filtrando ataques

* Ataques XSS(Cross Site Scripting).

Esta falla suele producirse por varias razones

** Variables no inicializadas correctamente.

** Ausencia de control de datos.

Ejemplo

Código principal

```
echo $_POST['user_name']; // Cadena ingresada <script>alert  
( 'chebyte' )</script>
```

Este falla es muy común en los guestbook(libro de visita), en donde la firma del usuario(user_name), no es sometida a ningún tipo de filtros.

Entonces al mostrar la firma, se le permite al usuario inyectar cualquier tipo de código, como el del ejemplo, que genera una ventana en el cliente con el texto "chebyte".

**Soluciones

* Inicializar variable.

* Filtrar la variable.

Ejemplo

Código principal

```
clean=array();
```

```
$_POST['user_name']; // Contiene la siguiente cadena ingresada por el usuario <script>alert('chebyte')</script> (xss)
```

```
$clean['user_name'] = htmlentities($_POST['user_name']); // Imprime <script>alert('chebyte')</script> como texto plano.
```

También se podría utilizar el ejemplo anterior() para filtrar, cuya diferencia es que htmlentities filtra todo el html, en cambio en el ejemplo anterior podemos decirle a nuestra función que debe filtrar.*

* Ataques de SQL Injection.

Esta falla suele producirse por razones como

** Ausencia de control de datos.

Ejemplo

Código principal

```
SELECT * FROM tb_user WHERE tb_user='$_POST[form_user]' AND  
tb_passwd='$_POST[form_passwd]'
```

Un usuario podría manipular el formulario de autenticación dejando la consulta de la siguiente forma

```
SELECT * FROM tb_user WHERE tb_user='' or ''='' AND tb_passwd='' or  
''=''
```

Con lo cual se obtiene acceso inmediato, ya que la consulta (autenticación) es válida.

Esto se logra ingresando en los campos del formulario

```
usuario = ' or ''='
```

```
password = ' or ''='
```

**Soluciones

*Filtrar los datos

Ejemplo1

Código principal

```
clean=array();  
$_POST['user_name']; // Contiene la siguiente cadena ingresada por  
el usuario ' or ''=' (sql injection)  
  
$clean['user_name'] = addslashes($_POST['user_name']); // Contiene  
\' or \'\'=\'(sql injection anulado)  
  
* addslashes filtra ('), ("), (\) y NULL.
```

Ejemplo2

Código principal

```
clean=array();

$_POST['user_name']; // Contiene la siguiente cadena ingresada por
el
                    usuario ' or ''=' (sql injection)

$clean['user_name'] = mysql_real_escape_string($_POST['user_name']);
// Contiene \' or \'\'=\' (sql injection anulado)

*mysql_real_escape_string filtra  (\n),(\r),(\),('),(").
```