5-4-2020

Seguridad en PHP

Seguridad en Desarrollo de Software

jesus israel ruiz granados
TECNOLOGICO DE ESTUDIOS SUPERIOR DE IXTAPALUCA

Tabla de contenido

Elementos importantes que se deben de proteger dentro de un sistema (PHP).

Inyección de SQL (sql-injection)

Muchos desarrolladores web no son conscientes de cómo las consultas SQL pueden ser manipuladas, y asumen que una consulta SQL es una orden fiable. Esto significa que las consultas SQL son capaces de eludir controles de acceso, evitando así las comprobaciones de autenticación y autorización estándar, e incluso algunas veces, que las consultas SQL podrían permitir el acceso a comandos al nivel del sistema operativo del equipo anfitrión.

La inyección directa de comandos SQL es una técnica donde un atacante crea o altera comandos SQL existentes para exponer datos ocultos, sobrescribir los valiosos, o peor aún, ejecutar comandos peligrosos a nivel de sistema en el equipo que hospeda la base de datos. Esto se logra a través de la práctica de tomar la entrada del usuario y combinarla con parámetros estáticos para elaborar una consulta SQL. Los siguientes ejemplos están basados en historias reales, desafortunadamente.

Debido a la falta de validación en la entrada de datos y a la conexión a la base de datos con privilegios de superusuario o de alguien con privilegios para crear usuarios, el atacante podría crear un superusuario en la base de datos.

Asociado con PostgreSQL

Ejemplo 1

Dividir el conjunto de resultados en páginas ... y crear superusuarios (PostgreSQL)

```
<?php

$índice    = $argv[0]; // ¡Cuidado, no hay validación en la entrada de datos!
$consulta  = "SELECT id, name FROM products ORDER BY name LIMIT 20 OFFSET $in
dice;";
$resultado = pg_query($conexión, $consulta);

?>
```

Un usuario común hace clic en los enlaces 'siguiente' o 'atrás' donde el $índice está codificado en el URL. El script espera que el $índice entrante sea un número decimal. Sin embargo, ¿qué pasa si alguien intenta irrumpir anteponiendo a la URL algo como lo siguiente empleando urlencode ()?

Error del programador

```
0;
insert into pg_shadow(usename,usesysid,usesuper,usecatupd,passwd)
    select 'crack', usesysid, 't','t','crack'
    from pg_shadow where usename='postgres';
--
```

Si esto sucediera, el script podría otorgar un acceso de superusuario al atacante. Observe que 0; es para proveer un índice válido a la consulta original y así finalizarla.

Una forma factible de obtener contraseñas es burlar las páginas de búsqueda de resultados. Lo único que el atacante necesita hacer es ver si hay variables que hayan sido enviadas y sean empleadas en sentencias SQL que no sean manejadas apropiadamente. Estos filtros se pueden establecer comunmente en un formulario anterior para personalizar las cláusulas WHERE, ORDER BY, LIMIT y OFFSET en las sentencias SELECT. Si la base de datos admite el constructor UNION, el atacante podría intentar anteponer una consulta entera a la consulta original para enumerar las contraseñas de una tabla arbitraria. Se recomienda encarecidamente utilizar campos de contraseña encriptadas.

Ejemplo 2

Enumerar artículos ... y algunas contraseñas (cualquier servidor de bases de datos)

```
<?php

$consulta  = "SELECT id, name, inserted, size FROM products
              WHERE size = '$tamaño'";
$resultado = odbc_exec($conexión, $consulta);

?>
```

La parte estática de la consulta se puede combinar con otra sentencia SELECT la cual revelará todas las contraseñas:

```
'
union select '1', concat(uname||'-'||passwd) as name, '1971-01-01', '0'
from usertable;
--
```

Si esta consulta (jugando con ' y --) fuera asignada a una de las variables utilizadas en $consulta, despertaría a la consulta "monstruo".

Las sentecias UPDATE de SQL también son susceptibles a ataques. Estas consultas también están amenazadas por el corte y la anteposición de una consulta completamente nueva. El atacante podría juguetear con la cláusula SET, aunque en este caso, debe poseer algo de información sobre los esquemas para manipular la consulta con éxito. Esta información puede adquirirse examinando los

nombres de las variables del formulario, o sencillamente mediante la fuerza bruta. No hay muchas convenciones de nombres para campos que almacenen contraseñas o nombres de usuarios.

Seguridad del Sistema de Archivos

Tabla de contenidos

PHP está sujeto a la seguridad integrada en la mayoría de sistemas de servidores con respecto a los permisos de archivos y directorios. Esto permite controlar qué archivos en el sistema de archivos se pueden leer. Se debe tener cuidado con los archivos que son legibles para garantizar que son seguros para la lectura por todos los usuarios que tienen acceso al sistema de archivos.

Desde que PHP fue diseñado para permitir el acceso a nivel de usuarios para el sistema de archivos, es perfectamente posible escribir un script PHP que le permita leer archivos del sistema como /etc/passwd, modificar sus conexiones de red, enviar trabajos de impresión masiva, etc. Esto tiene algunas implicaciones obvias, es necesario asegurarse que los archivos que se van a leer o escribir son los apropiados.

Considere el siguiente script, donde un usuario indica que quiere borrar un archivo en su directorio home. Esto supone una situación en la que una interfaz web en PHP es usada regularmente para gestionar archivos, por lo que es necesario que el usuario Apache pueda borrar archivos en los directorios home de los usuarios.

Ejemplo 1

Un control pobre puede llevar a ....

```php
<?php
// eliminar un archivo del directorio personal del usuario
$username = $_POST['user_submitted_name'];
$userfile = $_POST['user_submitted_filename'];
$homedir  = "/home/$username";

unlink("$homedir/$userfile");

echo "El archivo ha sido eliminado!";
?>
```

Dado que el nombre de usuario y el nombre del archivo son enviados desde un formulario, estos pueden representar un nombre de archivo y un nombre de usuario que pertenecen a otra persona, incluso se podría borrar el archivo a pesar que se supone que no estaría permitido hacerlo. En este caso, usted desearía usar algún otro tipo de autenticación. Considere lo que podría suceder si las variables enviadas son "../etc/" y "passwd". El código entonces se ejecutaría efectivamente como:

Ejemplo 2:

Un ataque al sistema de archivos

```php
<?php
// elimina un archivo desde cualquier lugar en el disco duro al que
// el usuario de PHP tiene acceso. Si PHP tiene acceso de root:
$username = $_POST['user_submitted_name']; // "../etc"
$userfile = $_POST['user_submitted_filename']; // "passwd"
$homedir  = "/home/$username"; // "/home/../etc"

unlink("$homedir/$userfile"); // "/home/../etc/passwd"

echo "El archivo ha sido eliminado!";
?>
```

Hay dos medidas importantes que usted debe tomar para prevenir estas cuestiones.

Únicamente permisos limitados al usuario web de PHP.

Revise todas las variables que se envían.

Aquí está una versión mejorada del script:

Ejemplo 3

Comprobación más segura del nombre de archivo.

```php
<?php
// elimina un archivo del disco duro al que
// el usuario de PHP tiene acceso.
$username = $_SERVER['REMOTE_USER']; // usando un mecanismo de aute
nticación
$userfile = basename($_POST['user_submitted_filename']);
$homedir  = "/home/$username";

$filepath = "$homedir/$userfile";

if (file_exists($filepath) && unlink($filepath)) {
    $logstring = "Se ha eliminado $filepath\n";
} else {
    $logstring = "No se ha podido eliminar $filepath\n";
}
$fp = fopen("/home/logging/filedelete.log", "a");
fwrite($fp, $logstring);
fclose($fp);

echo htmlentities($logstring, ENT_QUOTES);

?>
```

Sin embargo, incluso esto no está exento de defectos. Si la autenticación del sistema permite a los usuarios crear sus propios inicios de sesión de usuario, y un usuario eligió la entrada "../etc/", el sistema está expuesto una vez más. Por esta razón, puede que prefiera escribir un chequeo más personalizado:

Ejemplo 4

Comprobación más segura del nombre de archivo.

```
<?php
$username      = $_SERVER['REMOTE_USER']; // usando un mecanismo de
autenticación
$userfile      = $_POST['user_submitted_filename'];
$homedir       = "/home/$username";

$filepath      = "$homedir/$userfile";

if (!ctype_alnum($username) || !preg_match('/^(?:[a-z0-9_-
]|\.(?!\.))+$/iD', $userfile)) {
    die("nombre de usuario o nombre de archivo incorrecto");
}

//etc...
?>
```

Dependiendo de su sistema operativo, hay una gran variedad de archivos a los que debe estar atento, esto incluye las entradas de dispositivos (/dev/ o COM1), archivos de configuración (archivos /etc/ y archivos .ini), las muy conocidas carpetas de almacenamiento (/home/, Mis documentos), etc. Por esta razón, por lo general es más fácil crear una política en donde se prohíba todo excepto lo que expresamente se permite.

Datos Enviados por el Usuario

Las mayores debilidades de muchos programas PHP no son inherentes al lenguaje mismo, sino simplemente un problema generado cuando se escribe código sin pensar en la seguridad. Por esta razón, usted debería tomarse siempre el tiempo para considerar las implicaciones de cada pedazo de código, para averiguar el posible peligro involucrado cuando una variable inesperada es enviada.

Ejemplo 1

Uso Peligroso de Variables

```php
<?php
// eliminar un archivo del directorio personal del usuario .. ¿o
// quizás de alguien más?
unlink ($variable_malvada);
// Imprimir el registro del acceso... ¿o quizás una entrada de /etc
/passwd?
fwrite ($desc_archivo, $variable_malvada);
// Ejecutar algo trivial.. ¿o rm -rf *?
system ($variable_malvada);
exec ($variable_malvada);

?>
```

Usted debería examinar siempre, y cuidadosamente su código para asegurarse de que cualquier variable siendo enviada desde un navegador web sea chequeada apropiadamente, y preguntarse a sí mismo:

¿Este script afectará únicamente los archivos que se pretende?

¿Puede tomarse acción sobre datos inusuales o indeseados?

¿Puede ser usado este script en formas malintencionadas?

¿Puede ser usado en conjunto con otros scripts en forma negativa?

¿Serán adecuadamente registradas las transacciones?

Al preguntarse adecuadamente estas preguntas mientras escribe su script, en lugar de hacerlo posteriormente, usted previene una desafortunada reimplementación del programa cuando desee incrementar el nivel de seguridad. Al comenzar con esta mentalidad, no garantizará la seguridad de su sistema, pero puede ayudar a mejorarla.

Puede que también desee considerar la des habilitación de register_globals, magic_quotes, u otros parámetros convenientes que pueden causar confusión sobre la validez, fuente o valor de una determinada variable. Trabajar con PHP en modo error_reporting(E_ALL) también puede ayudarle a advertir variables que están siendo usadas antes de ser chequeadas o inicializadas (de modo que puede prevenir que datos inusuales produzcan operaciones inadvertidas).