

## PHP - CONFIGURACIÓN SEGURA DE PHP.INI

By freed0m

Creado 18 Jul 2005 - 20:24

PHP se ha convertido en un lenguaje de scripting esencial para gran parte de sitios Web, y si tenemos planeado utilizarlo, es fundamental realizar una configuración pormenorizada del mismo, a continuación se detalla la funcionalidad de cada una de las directivas del fichero de configuración.

### DESACTIVANDO OPCIONES NO RECOMENDABLES

- **Register\_globals**

Esta opción esta deshabilitada por defecto en la versión 4.0.2 de PHP. Es bastante habitual que por motivos funcionales tengamos la tentación de activarla ya que muchas aplicaciones requieren una buena revisión de la configuración por defecto para conseguir que funcionen con register\_globals desactivado.

Register\_globals transforma los parametros incluidos en las peticiones en parámetros globales de PHP.

Por ejemplo, supongamos que tenemos una URL con el parámetro name:

```
http://www.hacktimes.com/sayhello.php?name=freed0m
```

El código PHP que procesa la petición podría ser algo tan simple como:

```
< ? echo "Hola $name!";?>
```

Con estos ejemplos de programación sencilla es lógico que PHP siga extendiéndose, sin embargo se esta generando gran cantidad de código inseguro, por ejemplo:

```
< ?
if (isset($admin) == false)
{
die "Esta web es solo para administradores!";
}
?>
```

Teóricamente la variable \$admin toma el valor true cuando después de la autenticación del usuario se comprueba que pertenece al grupo de administradores, en la práctica añadiendo ?admin=1 a la URL, PHP creará la variable \$admin donde no exista.

- **allow\_url\_fopen**

Esta opción permite a los programadores tratar las URLs como si fueran ficheros, esta opción viene activada por defecto, es habitual usar los datos de la petición para identificar el nombre del fichero a leer, por ejemplo:

```
http://www.hacktimes.com/view.php?what=index.php
```

La aplicación usa el valor del parámetro `what` para llamar directamente al constructor `include()`:

El resultado de permitir este tipo de peticiones es que si en lugar de indicar en la petición el fichero `index.php` especificamos por ejemplo, `/etc/passwd`, podríamos abrir cualquier fichero del servidor.

La función `include()` muestra el contenido del fichero en la página web resultante.

Existen otras funciones como `readfile()` que también muestran el contenido del fichero por pantalla.

Con la función `allow_url_fopen?` si introducimos una URL en el parámetro `what` del ejemplo, PHP leerá y ejecutará el código de la dirección que se haya indicado como parámetro.

Por lo tanto en nuestra configuración del fichero `php.ini` especificaremos `allow_url_fopen = Off`

## ▪ Carga dinámica de módulos

PHP utiliza módulos para añadir funcionalidades de forma dinámica. A diferencia de Apache, PHP puede cargar módulos desde programas usando la función `d1()` desde un script.

Cuando se carga un modulo, se integra en PHP y se ejecuta con permisos totales. Por lo tanto es factible escribir una extensión para intentar eludir las limitaciones que se establecen a través de la configuración.

En referencia a las posibles complicaciones de seguridad derivadas de la función `d1()` hay un artículo de phrack titulado "Attacking Apache with builtin Modules in Multihomed Environments" por andi@void [www.phrack.org/phrack/62/p62-0x0a\\_Attacking\\_Apache\\_Modules.txt](http://www.phrack.org/phrack/62/p62-0x0a_Attacking_Apache_Modules.txt) [1]

El ataque describe como un usuario utiliza las opciones de PHP para añadir funcionalidades para cargar en la memoria de Apache código y tomar el control del servidor web.

Por lo tanto los módulos que se han de usar son aquellos que estan indicados en el fichero `php.ini` por lo que desactivaremos esta función, `enable_d1 = Off`.

- **Información sobre PHP en las cabeceras del servidor web**

PHP permite incluir su firma en el servidor Web, en la cabecera `Header` esta característica esta orientada a sacar estadísticas de uso de php, como se puede ver en [www.php.net/usage.php](http://www.php.net/usage.php) [2].

Existen varias peticiones que se pueden realizar sobre un servidor con `expose_php = On` para mostrar distintos mensajes:

La siguiente petición mostraría el logotipo de PHP: `?=PHPE9568F34-D428-11d2-A769-00AA001ACF42`.

Con esta petición saldría el logotipo de Zend: `?=PHPE9568F35-D428-11d2-A769-00AA001ACF42`.

Para mostrar una imagen que varía según la versión de PHP: `?=PHPE9568F36-D428-11d2-A769-00AA001ACF42`

Para los créditos de los Programadores: `?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000`.

Podemos deshabilitar en el fichero `php.ini` esta opción dando el valor `Off` a la variable `expose_php`:

```
expose_php = Off
```

Otra de las opciones para no mostrar esta información en las cabeceras es utilizando la directiva `SecServerSignature` de [modsecurity.org](http://modsecurity.org) [3] con el valor que queramos que se muestre entre comillas.

## DESACTIVANDO FUNCIONES Y CLASES

Mediante el uso de las directivas `disable_functions` y `disable_classes` es posible deshabilitar funciones y clases.

Uno de los ejemplos más claros es el de la función `openlog()`. Esta función junto con `syslog()` permite a los scripts en PHP enviar mensajes al `syslog`. Desafortunadamente, es posible modificar el nombre con el cual el proceso queda registrado en el `syslog`. Por lo tanto si se modifica este nombre, en grandes sistemas se complicaría la identificación de los datos almacenados en `syslog` con ese nombre.

Para evitar estas situaciones configuraremos `disable_functions = openlog`

Existen otras funciones de integración de PHP/Apache que son potencialmente peligrosas. Y son las siguientes:

```
apache_child_terminate  
apache_get_modules  
apache_get_version  
apache_getenv  
apache_note
```

```
apache_setenv
virtual
```

Si ninguno de los scripts de las aplicaciones que se ejecutan en PHP necesitan estas funciones es recomendable deshabilitarlas.

**NOTA:** En [drupal](#) [4] versión 4.5.4-1 para debian con los módulos del core de la aplicación no hay problemas deshabilitando estas funciones.  
En [Phpbb](#) [5] tampoco hay problemas con las versiones 2.0.13-6 también para debian.

## RESTRICCIONES DE ACCESO AL SISTEMA DE FICHEROS

La directiva más útil en relación con la seguridad en PHP es `open_basedir`. Esta directiva indica a PHP a que ficheros puede acceder y a cuales no. El valor de esta directiva es una lista de prefijos de ficheros separados por una coma en Unix y por punto y coma en Windows.

Las restricciones configuradas aquí, afectan a los scripts de PHP y a los ficheros de datos. Se recomienda que se active esta opción incluso en aquellos servidores con un único sitio web, y debe de apuntar un nivel por encima del directorio raíz del servidor web, la configuración sería `open_basedir = /var/www/`

Es importante recordar la diferencia entre establecer las restricciones a un prefijo frente a establecer las restricciones a un directorio, por ejemplo:

```
open_basedir=/var/www permitiría acceso tanto a ficheros de /var/www como de
/var/www2
```

... sin embargo si usamos ...

```
open_basedir=/var/www/ permitiría acceso a los ficheros que estén únicamente en
/var/www/.
```

**NOTA:** Independientemente de esta consideración no es posible controlar todos los módulos de php que desarrollan terceros, FraMe publicó en [Bugtraq PHP4 cURL functions bypass open\\_basedir](#) [6] como es posible eludir las restricciones de `open_basedir` mediante la extensión **cURL PHP**.

## ACTIVANDO OPCIONES DE REGISTRO

Por defecto no se registran todos los eventos en PHP. Hay mensajes importantes etiquetados con nivel `E_NOTICE` y que se pasan por alto.

Es importante activar el registro de todos los eventos con:

```
error_reporting = E_ALL
log_errors = On
```

Para ver cualquier error, es necesario activar el registro de errores. Para ello usaremos la opción de configuración `error_log`. Por defecto los errores irán a la salida estándar de error, en este caso, el log de error de Apache. Sin embargo podemos cambiar el destino de los mensajes de error especificando por ejemplo `syslog` en este caso `syslog`

registraría los eventos. O < logphp > en cuyo caso los mensajes se almacenarían en ese fichero de nombre logphp.

Si utilizamos un fichero distinto del de Apache para registrar los errores, estableceremos los permisos de acceso adecuados. A diferencia de los logs de Apache que se abren al comienzo cuando Apache aún esta corriendo como root, los logs de PHP se crean y escriben después, cuando el proceso ya esta ejecutándose con los permisos del usuario que inicia el servicio web. Por lo tanto los ficheros de log no se pueden guardar en el mismo sitio. Así que nos crearemos una carpeta y daremos permisos de acceso al usuario del servidor web (www-data en debian):

```
#mkdir /var/www/logs/php
#chown www-data /var/www/logs/php
```

En el fichero de configuración php.ini configuramos  
error\_log=/var/www/logs/php/php\_error\_log

En los servidores en producción se deberá desactivar la opción de mostrar errores de inicialización de PHP como errores de ejecución. Para ello:

```
display_errors = Off
display_startup_errors = Off
```

## ESTABLECIENDO LIMITES DE MEMORIA

- Php permite establecer la **cantidad de memoria** que consume un script. En previsión de scripts que puedan estar mal programados y que podrían llegar a consumir toda la memoria y bloquear el sistema. Para ello emplearemos:  
Establecemos este valor con:  
memory\_limit = 8M ; 8 MB es la cantidad de memoria por defecto.
- También podemos establecer el **tamaño máximo de las peticiones POST**, el valor de esta variable tiene que estar en concordancia con el tamaño máximo de ficheros que se puedan subir.  
Establecemos este valor con:  
post\_max\_size = 8M ; 8 MB es el tamaño máximo del POST por defecto.
- Es posible establecer el **tiempo máximo que puede estar PHP procesando la entrada de datos**. El limite por defecto son 60 segundos, es conveniente incrementar este valor si hay usuarios con conexiones lentas subiendo ficheros de gran tamaño. La métrica que se da en <http://www.apachesecurity.com> sobre este asunto es, para una velocidad de 5KBps es posible subir 300 KB en 60 segundos. Con esta medida podemos valorar si es necesario incrementar este valor y en cuanto.  
Establecemos este valor con:  
max\_input\_time = 60
- El limite de **tiempo que un script PHP puede estar ejecutándose** (sin incluir llamadas a procedimientos externos) esta establecido por defecto a 30 segundos. Este valor por defecto es bastante elevado, pero antes de reducirlo es necesario valorar el rendimiento de la aplicación para dar un limite acertado.

Para establecer este limite usaremos:  
`max_execution_time = 30`

## CONTROL DE SUBIDA DE FICHEROS

Las consideraciones relativas al control de "upload" de ficheros se establecen en el parámetro de `php.ini`, identificado por `file_uploads` que pondremos `file_uploads = Off` o a `On` si queremos permitir subir ficheros mediante las aplicaciones en `php`. Los parámetros relacionados con esta opción son `upload_max_filesize = 2M` donde se indica el tamaño máximo de ficheros que se pueden subir a la vez, y que por defecto es de 2 MB. Se pueden subir varios ficheros simultáneamente, y el tamaño total es el que se indique en este parámetro.

**NOTA:** Es necesario establecer en la configuración un valor levemente superior de `post_max_size` al de `max_filesize`.

Los ficheros se suben al servidor web antes de ser procesados por un script, y por defecto se almacenan en `/tmp`. Si se quiere cambiar esta configuración se especificará la ruta con la variable `upload_tmp_dir = /var/www/tmp`.

Esta ruta se habilitará con los comandos:

```
#mkdir /var/www/tmp
#chown www-data /var/www/tmp
```

## CONTROL DE SESIONES

HTTP es un protocolo sin control de estados. Lo que quiere decir que el servidor trata cada petición de usuario de forma independiente y sin tener en cuenta que ha sucedido antes. No solo no lo toma en cuenta si no que ni siquiera lo "recuerda". A la hora de realizar desarrollos que utilizan las sesiones para agrupar peticiones, puede ser un inconveniente que no exista un control de estados.

Las sesiones funcionan asignando una porción de información a cada usuario cuando conecta al sitio Web por primera vez. Esta porción de información se denomina identificador de sesión `sessionid`. El mecanismo mediante el cual se asigna el `sessionid` esta ideado para que el navegador del usuario devuelva información al servidor en cada petición.

El servidor Web utiliza la información del `sessionid` para identificar datos del usuario y peticiones anteriores. Desde que el identificador de sesión se utiliza para identificar a un usuario anterior, se podría decir que actúa como un identificador único de usuario temporal. Con esto surge la problemática de los ataques de secuestro de sesiones, donde conociendo el `sessionid` se podría suplantar la identidad del usuario autenticado, es decir, se podría conectar a la aplicación con los privilegios del usuario autenticado.

El soporte de sesiones en PHP permite a las aplicaciones recordar a un usuario, manteniendo cierta información entre peticiones. Por defecto, esta información se

almacena en la ruta `/tmp`.

La información de las sesiones tiene el siguiente formato:

```
sess_ed6128354913y91y3249183y42
```

Analizando este código se puede identificar que PHP utiliza identificadores de sesión cuando construye los nombres de los ficheros para los datos de las sesiones (el identificador de sesión después de `sess_`). Como consecuencia, cualquier usuario del sistema con acceso a la carpeta `/tmp` podría utilizar los identificadores de sesión y secuestrar la sesión de los usuarios activos. Para evitarlo, especificaremos a PHP como almacenar los datos de las sesiones en un directorio separado, con acceso únicamente para el usuario de Apache (`www-data`). Para ello:

```
# mkdir /var/www/sessions
# chown www-data /var/www/sessions
```

Indicamos esta ruta en el fichero `php.ini` mediante `session.save_path = /var/www/sessions`

Este cambio de configuración no soluciona todos los problemas. Los usuarios del sistema no podrán utilizar los identificadores de sesión si los permisos de la carpeta de sesiones `/var/www/sessions` están configurados para no permitirlo. Aún así, para cualquier usuario que pueda escribir y ejecutar scripts en PHP en el sistema, será muy sencillo escribir un programa que obtenga la lista de identificadores de sesión ya que el script se ejecutará con los privilegios del usuario que ejecuta el servicio Web.

**NOTA:**Un directorio de sesiones nunca ha de ser compartido por varias aplicaciones, grupos de usuarios, o sitios Web.

Las fugas de identificadores de sesión y los intentos de secuestro pueden evitarse con la ayuda de la opción `session.referer_check`. Activando esta opción PHP analizará el contenido de la cabecera `Referer` en busca de la cadena que le indiquemos. Se debería establecer este valor a una cadena que contenga parte del nombre de dominio del sitio Web.

En nuestro caso podríamos indicar `session.referer_check = hacktimes.com`

Como la cabecera `Referer` contiene la URL de la página anterior que ha visitado el usuario, contendrá el nombre del dominio del sitio Web en todas las peticiones. Sin embargo, si alguien sigue un link desde otro sitio y llega a el site con un identificador valido de sesión `sessionid`, PHP rechazará la petición. **Esta protección no ha de tomarse muy en serio** . Esta opción se diseño para invalidar sesiones que hubieran sido comprometidas por usuarios de forma accidental al publicar enlaces que contienen identificadores de sesión. Sin embargo, protegerán de ataques sencillos de `cross-site request forgery` (CSRF), estos ataques consisten en la creación de un sitio web que pide al navegador del usuario los identificadores de sesión de otro site. Cuando se puede controlar completamente la petición, también se puede controlar el contenido de la cabecera `Referer`, haciendo que esta verificación sea ineficaz.

Con esta opción activa, incluso aquellos usuarios con navegadores con soporte para cookies (y que estén usando las cookies para la gestión de la sesión) tendrán sesiones no

validas si siguen un enlace desde cualquier sitio a nuestro site. Por lo tanto, aunque `session.referer_check` no soluciona completamente ningún problema, es recomendable activarlo aunque habrá que establecer medidas efectivas que eviten el secuestro de sesión en el desarrollo de las aplicaciones en PHP.

## ACTIVANDO LAS OPCIONES DE MODO SEGURO (SAFE\_MODE)

- `safe_mode` documentado en [php.net/manual/es/features.safe-mode.php](http://php.net/manual/es/features.safe-mode.php) [7] es una aproximación a establecer elementos de seguridad para los desarrollos en PHP. Activando esta opción, `safe_mode = On` el motor de PHP impone una serie de restricciones, haciendo que la ejecución de scripts sea más segura. El modo seguro esta implementado como un conjunto de verificaciones especiales en el código fuente de PHP, aunque no hay garantías de que estas comprobaciones estén en todos los sitios que deberían. De todas formas recomendamos activarla `safe_mode = On`

- **Restricciones de acceso a ficheros**

Las verificaciones que establece `safe_mode` tienen su mayor impacto en el acceso a ficheros. Con este modo se realizan verificaciones especiales antes de cualquier operación sobre un fichero. Se comprobará que el `uid` del propietario del fichero coincida con el `uid` de la cuenta de usuario que posea el script. Esta comprobación puede generar problemas en los siguientes casos:

- Si existe más de un usuario con permisos de escritura en el arbol de directorios del servidor web. Tarde o temprano un script de otro usuario intentará acceder a un fichero propiedad de otro usuario.

- Si las aplicaciones crean ficheros en tiempo de ejecución.

Este segundo caso es el que genera más problemas, la mayoría de las aplicaciones en PHP se usan como CMS (Content Management Systems) o sistemas de gestión de contenidos y todos ellos crean ficheros. La solución más sencilla para evitar este problema pasa por tener la cuenta de desarrollo y la cuenta de Apache en el mismo grupo, y relajar la verificación de `uid` usando en su lugar el `gid`.

Para ello estableceremos `safe_mode_gid = On` .

**Nota:** Como la mayoría de los scripts en PHP utilizan otros scripts o librerías, se pueden hacer excepciones para estos casos. Si un directorio esta especificado en la ruta de ficheros incluidos `include_path` y esta especificado en `safe_mode_include_dir`, la verificación de `uid / gid` se obviará.

- **Restricciones de variables de entorno**

La posibilidad de escritura en variables de entorno usando la función `putenv()` esta restringida cuando se usa el modo seguro.

La primera de las siguientes dos directivas `safe_mode_allowed_env_vars`, contiene una lista de prefijos separados por coma que indican que variables de entorno pueden ser modificadas. La segunda directiva

`safe_mode_protected_env_vars` prohíbe que ciertas variables de entorno separadas por coma, puedan ser modificadas.

```
# allow modification of variables beginning with PHP_
safe_mode_allowed_env_vars = PHP_
# no one is allowed to modify LD_LIBRARY_PATH
safe_mode_protected_env_vars = LD_LIBRARY_PATH
```

## ▪ Restricciones de ejecución de procesos externos

El modo seguro establece restricciones en cuanto a la ejecución de procesos externos. Sólo aquellos binarios indicados en el directorio "seguro" podrán ser ejecutados en los scripts desde PHP, indicaremos la ruta de estos binarios con:

```
safe_mode_exec_dir = /var/www/bin
```

Las funciones afectadas son:

```
exec()
system()
passthru()
popen()
```

Algunos metodos de programación no funcionarán en modo seguro:

```
shell_exec()
backtick operator
```

## ▪ Otras restricciones del modo seguro

`d1` Deshabilitada en modo seguro.

`set_time_limit` No tiene validez en modo seguro. Tampoco tiene validez la directiva `max_execution_time`.

`header` En modo seguro, el `uid` del script se añade a la cabecera `WWW-Authenticate HTTP`.

`apache_request_header` En modo seguro las cabeceras que empiezan con `Authorization` no se muestran.

`mail` El quinto parámetro (`additional_parameters`) esta desactivado.

Normalmente este parámetro es enviado en línea de comandos por el programa que envía el mensaje (por ejemplo, `postfix`).

`variables PHP_AUTH` Las variables `PHP_AUTH_USER`, `PHP_AUTH_PW`, y `AUTH_TYPE` no están disponibles en modo seguro.

## PROTECCIONES MÁS ALLÁ DE PHP.INI

Existen proyectos desarrollados exclusivamente para establecer más mecanismos de control en relación con la seguridad en php, por ejemplo [hardened-php.net](http://www.hardened-php.net) [8] es un parche para PHP que habilita nuevas opciones de configuración desde php.ini, estas características afectan a elementos como:

- Elementos de registro o log, permitiendo gestionar los eventos en función de distintas clasificaciones además de la posibilidad de establecer prioridades y criticidades. Podremos enviar los eventos a scripts de registro externos, o incluir las direcciones IP en los eventos extrayéndolas de las cabeceras X-Forwarded-For.
- Filtrado del contenido de GET, POST o COOKIE, en cuanto a número de variables, longitud máxima del nombre de la variable, longitud máxima del índice del array, longitud máxima del valor de la variable, máxima profundidad del array, etc.
- Permite limitar el número máximo de ficheros que se pueden subir.
- Protección del núcleo de PHP y extensión contra vulnerabilidades de formato de cadena (format string).
- Protecciones en relación con la gestión de la memoria.
- Protecciones en tiempo de ejecución, como:
  - Límite de profundidad de ejecución.
  - Registro de peticiones fallidas de extensiones de bases de datos \*SQL.
  - Permite detener un script tras un fallo de ejecución de SQL.
  - Es posible prohibir múltiples cabeceras HTTP en llamadas a `header()` por defecto.
- Límites en relación con ficheros como, filtro en la longitud del nombre de un fichero, filtro de URL, filtro de ficheros subidos, permite truncar el nombre de los ficheros.
- Control del límite de memoria para que no pueda crecer por encima del límite configurado.

Las anteriores son una serie de funcionalidades que aporta este desarrollo hoy en día, contamos con que se sigan aportando nuevas funcionalidades en posteriores versiones.

### Bibliografía:

<http://www.php.net>  
<http://www.apachesecurity.net>  
<http://www.hardened-php.net>

---

### Source URL:

<http://www.hacktimes.com/?q=node/22>

### Links:

- [1] [http://www.phrack.org/phrack/62/p62-0x0a\\_Attacking\\_Apache\\_Modules.txt](http://www.phrack.org/phrack/62/p62-0x0a_Attacking_Apache_Modules.txt)
- [2] <http://www.php.net/usage.php>
- [3] <http://www.modsecurity.org>

[4] <http://www.drupal.org>

[5] <http://www.phpbb.com>

[6] <http://www.securityfocus.com/archive/1/379657/2004-10-26/2004-11-01/0>

[7] <http://www.php.net/manual/es/features.safe-mode.php>

[8] <http://www.hardened-php.net>