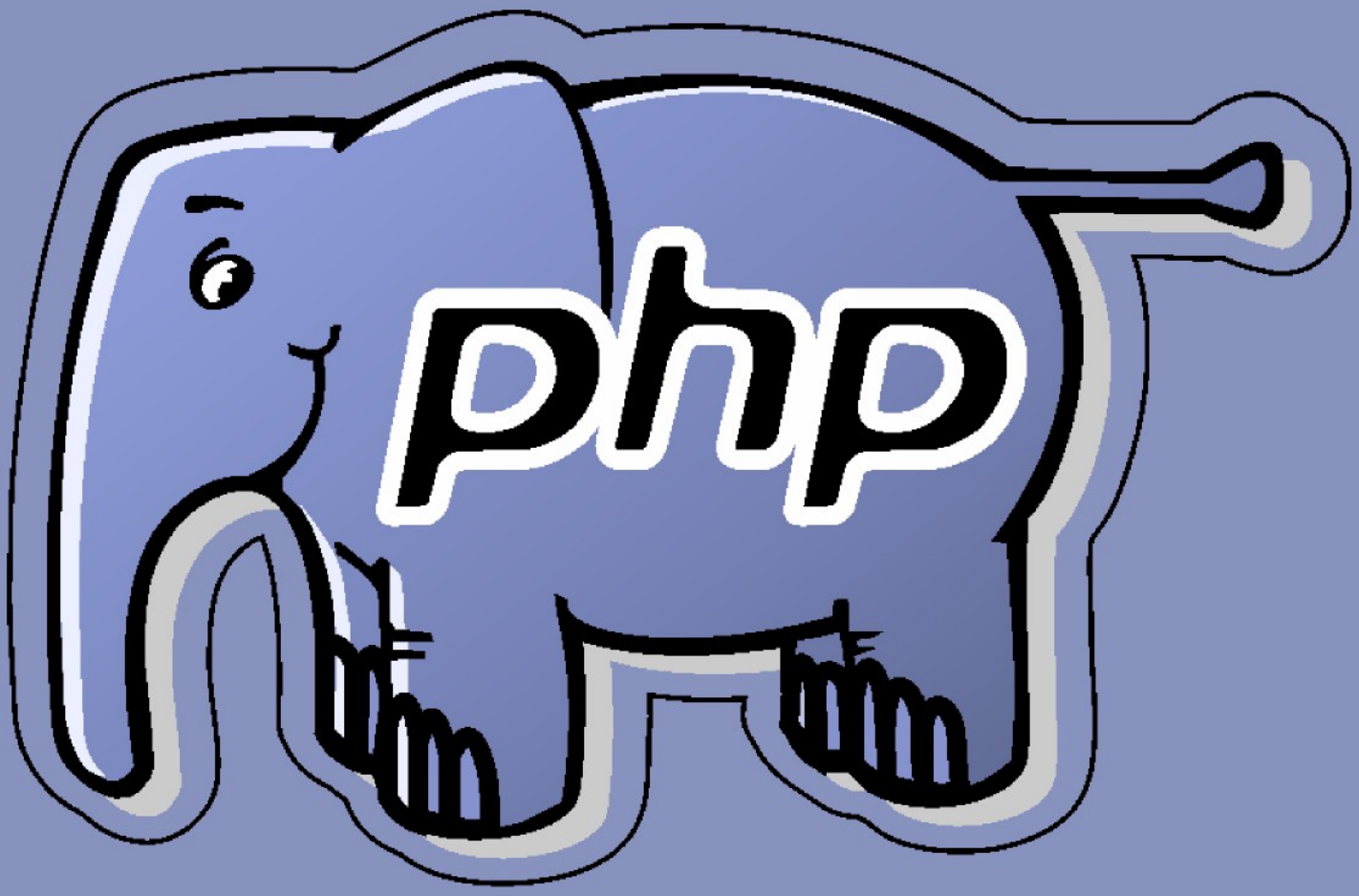
Programación en



Versión 2.0

php

G.U.G.L.E.R
Grupo Universitario de GNU/Linux
de Entre Ríos

F.C.y.T
Facultad de Ciencia y Tecnología
Universidad Autónoma de Entre Ríos

CONTENIDO

Capítulo 6

Seguridad en PHP

Introducción

Para cerrar el capítulo de seguridad en PHP, analizaremos otros tipos de ataque comunes y otros no tan comunes, pero que vale la pena conocerlos. Explicaremos para cada uno de ellos las herramientas que tenemos en PHP para mitigarlos.

Por último, veremos algunos aspectos a tener en cuenta sobre la configuración de PHP, de manera tal de poder reducir espacios vulnerables a ser utilizados por usuarios mal intencionados.

Seguridad de sesión

Dos formas populares de ataques de sesión son la fijación de sesión (session fixation) y secuestros de sesión (session hijacking). Considerando que la mayoría de los otros ataques descritos en este capítulo se pueden prevenir mediante el filtrado de datos de entrada y salida, los ataques de la sesión no.

Cuando un usuario se encuentra con una primera página en su aplicación que llama a *session_start()*, se crea una sesión para el usuario. PHP genera un identificador de sesión al

azar para identificar el usuario, y luego envía un encabezado *Set-Cookie* para el cliente. De forma predeterminada, el nombre de esta cookie es *PHPSESSID*, pero es posible cambiar el nombre de la cookie en el *php.ini* o utilizando la función *session_name()*. En visitas posteriores, el cliente identifica el usuario con la *cookie* y así es como la aplicación mantiene el estado.

Session fixation

Es posible, sin embargo, establecer el identificador de la sesión manualmente, forzando el uso de la misma. A este tipo de ataque se lo llama session fixation, ya que el atacante es quien fija la sesión.

Existen tres formas (métodos) de obtener un identificador de sesión válido en una aplicación Web:

- **Predicción** refiere a adivinar un identificador de sesión válido. Con el mecanismo nativo de sesiones de PHP, el identificador de sesión es muy aleatoria, y esto es poco probable que sea el punto más débil en su implementación.
- **Captura** de un identificador de sesión válido es el tipo más común de ataque período de sesiones, y existen numerosos modelos. Debido a que los identificadores de sesión suelen ser reproducidos en las cookies o como variables GET
- La **fijación** es el método más simple de obtener un identificador de sesión válido. Aunque no es muy difícil de defender en contra, si el mecanismo de sesión consiste en nada más que *session_start()*, que son vulnerables.

Esta última opción suele lograrse mediante la creación de un enlace y de anexar al mismo el identificador de sesión que el atacante desea dar.

```
<a href="http://example.org/index.php?PHPSESSID=1234">Haga click Aquí </a>
```

Mientras que el usuario accede a su sitio a través de ésta sesión, puede proporcionar información sensible o incluso credenciales de acceso en el tiempo que dure la misma. De esta manera el atacante puede ser capaz de acoplarse a dicha sesión y obtener acceso a la cuenta del usuario.

Para mitigar este tipo de ataque la solución es muy simple: cada vez que un usuario navegue de un script a otro, será necesario regenerar el identificador de sesión. PHP hace esta tarea simple con un *session_regenerate_id()*.

Ejemplo:

```
<?php
session_start();
if (!isset($_SESSION['initiated']))
{
```

```
    session_regenerate_id();
    $_SESSION['initiated'] = true;
}
```

Aquí podemos ver el uso de la función session_regenerate_id(). Al regenerar el id de la sesión en cada script, será mas difícil que un atacante pueda realizar un ataque de fijación de sessión.

Session hijacking

Viene a reforzar la identificación del cliente para evitar el robo del id de sesión, con lo cual utilizaremos información del header de http para obtener un identificador adicional del cliente que accede a nuestro sitio Web.

La idea detrás de session hijacking es la de almacenar en sesión información del navegador web que utilizó el cliente para loguearse. Esta información la guardaremos encriptada en alguna variable de sesión y por cada página que nuestro cliente visite, nos cercioraremos de que esta información no haya cambiado.

PHP nos provee de un arreglo llamado *$_SERVER* el cual contiene mucha información referida a la comunicación entre el cliente y el servidor, entre otras. En todo esto encontramos *$_SERVER['HTTP_USER_AGENT']* que contiene información sobre el cliente y mediante la cual podremos aplicar este mecanismo de seguridad.

Ejemplo:

proceso.php

```
<?php
        session_start();
        // Después de hacer un buen login
        …
        $_SESSION['HUA'] = md5($_SERVER['HTTP_USER_AGENT']);
}
```

En este primer script, luego de validado el login, obtenemos información sobre el navegador que utilizó el cliente para conectarse. Esta información es una cadena alfanumérica la cual encriptaremos y almacenaremos el resultado un una variable de sesión ($_SESSION['HUA']). De esta manera, no solo estamos validando el usuario y la contraseña sino que además verificamos que utilice siempre el mismo navegador en el tiempo que dure la sesión.

chequear.php

```
<?php
        session_start();
        // Después de hacer otros chequeos
        ...
```

```
        if ($_SESSION['HUA'] != md5($_SERVER['HTTP_USER_AGENT'])) {
            die("Usted cambio de navegador!!!!!");
}
```

Una vez identificado el navegador que utiliza el cliente y habiendo guardado información del mismo, lo que nos queda por hacer es, en cada script que queramos asegurar, agregar la validación de la información almacenada en sesión ($_SESSION['HUA']) y la obtenida de la conexión (md5($_SERVER['HTTP_USER_AGENT'])).

Command Injection

PHP tiene la capacidad de acceder directamente al sistema de ficheros de nuestro disco e incluso ejecutar comandos de una shell, como por ejemplo Bash. De esta forma, podemos crear, renombrar e incluso borrar archivos y directorios en nuestro disco rígido desde aplicaciones web. Esto permite a los desarrolladores un gran poder en cuanto a programación de script que requieran de este tipo de funciones, pero puede ser muy peligroso cuando se deja habilitada esta capacidad a un atacante.

Un ataque de inyección de código (command injection) se produce cuando un atacante es capaz de hacer que la aplicación ejecute código PHP de su elección. Esto puede tener consecuencias devastadoras tanto para su aplicación y sistema. La inyección de comandos se puede llevar a cabo desde inclusiones dinámicas no controladas. Ejecutando comandos del sistema operativo con las funciones *exec()*, *shell_exec()*, *passthru()* y *system()*.

Una manera que tenemos de mitigar este tipo de ataques es deshabilitando este tipo de funciones en PHP. Para esto, deberemos desde el archivo de configuración (php.ini) deberemos agregar las funciones que queramos anular.

```
disable_functions = exec, passthru, shel_exec, system
```

Hosting compartido

Hay una variedad de problemas de seguridad que surgen al utilizar un servidor compartido.
En el pasado, PHP ha tratado de resolver algunos de estos temas con la directiva *safe_mode*.
Sin embargo, como dice el manual de PHP, "es arquitectónicamente incorrecto para tratar de resolver este problema en el nivel de PHP". Por lo tanto, *safe_mode* ya no estará disponible a partir de PHP 6.

Sin embargo, hay dos directivas php.ini que siguen siendo importantes en un entorno de alojamiento compartido: *open_basedir* y *disable_classes*. Estas directivas no dependen de *safe_mode*, y seguirán estando disponibles en el futuro previsible

La directiva *open_basedir* proporciona la capacidad de limitar los archivos PHP que puede abrir a un árbol de directorios especificado. Cuando PHP intenta abrir un archivo con, por ejemplo, *fopen()* o *include()*, comprueba la ubicación del archivo. Si existe en el árbol de directorios especificada por *open_basedir*, entonces tendrá éxito, de lo contrario, él no podrá abrir el archivo

Usted puede configurar la directiva open_basedir en php.ini de manera tal que, por ejemplo, los archivos almacenados en el directorio */tmp* en un sistema linux no puedan ser habiartos ya que en dicho directorio se almacenan por defecto, entre otros, los archivos de sesión que crea PHP mediante *session_start()*.

Algunas opciones más de php.ini

A continuación veremos algunas opciones más a tener en cuenta en la configuración de nuestro PHP.

Etiquetas reducidas

Activar el reconocimiento de etiquetas reducidas en PHP. La apertura y cierre de etiquetas en php se puede dar mediante los tags: **<? ?>** , **<?php ?>** , **<script language="php"> </script>** y **<% %>.**De estas cuatro formas es que se puede incrustar código PHP para que sea interpretado por el servidor

```
short_open_tag = On
```

Esta directiva hará que el código PHP sea reconocido mediante las etiquetas **<? ?>** , **<?php ?>** , **<script language="php"> </script>,** pero no **<% %>.**

```
asp_tags = On
```

Esta es la directiva que hará que el código PHP sea reconocido mediante **<% %>.**
Verifique estas opciones ya que de tener alguna configuración distinta, los resultados que obtendrá en el cliente (navegador) puede que no sean los esperados.

Firma de PHP

Cada vez que accedemos a una página web se establece un flujo de información en el que viajan muchos datos. Algunos de éstos datos nos son accesibles desde el navegador, otros no. Entre este conjunto de datos, existen algunos que pueden ser utilizados por usuarios mal intencionados.

Entre estos datos que viajan en la cabecera HTTP se envía, de manera predeterminada, la versión de PHP que está corriendo en el servidor. Para desactivar la exposición de la firma (signature) de PHP en la cabecera http de nuestro servidor Web:

```
expose_php = Off
```

Tamaño de memoria

Si queremos establecer el tamaño máximo de memoria que puedeconsumir un script.

```
memory_limit = 128
```

Subida de Archivos

Permitir la subida de archivos desde HTTP usando un script PHP.

```
file_uploads = On
```

Especificar un directorio temporario para los archivos subidos al servidor utilizando HTTP y PHP.

```
upload_tmp_dir = /tmp
```

Especificar el tamaño máximo de los archivos a subir utilizando HTTP y un script PHP.

```
upload_max_filesize = 2M
```

Especificar una cantidad máxima de archivos a subir al servidor.

```
max_file_uploads = 20
```

Opciones de sesión

Especificar la ruta donde se guardaran los datos de las distintas sesiones que se creen en el servidor.

```
session.save_path = /tmp
```

Especificar el tiempo de vida de una sesión.

```
session.cookie_lifetime = 120
```