Desarrollo de sitios web con PHP y MySQL

Tema 7: Seguridad

José Mariano González Romano
mariano@lsi.us.es

Tema 7: Seguridad

1. Seguridad en las aplicaciones web
2. Seguridad en PHP
3. Variables globales
4. Nombres de ficheros
5. Subida de ficheros
6. Bibliotecas
7. Formularios
8. Inyección SQL

Seguridad en las aplicaciones web

- ¿Cuánta seguridad es necesaria?
  - La seguridad supone un coste económico y de eficiencia. Hay que disponer de la adecuada, ni más ni menos
- Reglas:
  - El riesgo cero no es práctico
  - Hay diversas formas de mitigar el riesgo
  - No se puede gastar un millón para proteger un céntimo

Seguridad en las aplicaciones web

- Amenazas más importantes: Top 10
  - *The Open Web Application Security Project (OWASP) The Ten Most Critical Web Application Security Vulnerabilities (2007)* http://www.owasp.org/documentation/topten
    1. Cross Site Scripting (XSS)
    2. Injection Flaws
    3. Malicious File Execution
    4. Insecure Direct Object Reference
    5. Cross Site Request Forgery (CSRF)
    6. Information Leakage and Improper Error Handling
    7. Broken Authentication and Session Management
    8. Insecure Cryptographic Storage
    9. Insecure Communications
    10. Failure to Restrict URL Access

Seguridad en las aplicaciones web

- Seguridad en el cliente
  - Código móvil
- Seguridad en el servidor
  - Servidor web, servidor de bases de datos, lenguajes de servidor
- Seguridad en la aplicación
  - Control de acceso
  - Validación de datos de entrada
  - Programación segura
- Seguridad en la comunicación
  - Certificados digitales, SSL

Seguridad en PHP

- Primera recomendación:
  - Disponer siempre de versiones actualizadas de Apache y PHP
- Aspectos de PHP que pueden dar lugar a vulnerabilidades:
  - Variables globales
  - Nombres de ficheros
  - Subida de ficheros
  - Bibliotecas
  - Datos enviados desde formularios

Variables globales

- Cuando *register_globals* está activado en el fichero php.ini, PHP crea automáticamente variables globales a partir de los datos de los formularios y de las *cookies*
- Esto puede dar lugar a problemas como en el ejemplo siguiente:

```
<?PHP
   if (comprueba_privilegios())
    $superuser = true;
   ...
?>
```

Variables globales

- Una llamada a este *script* de la forma

  `pagina.php?superuser=1`

  permitiría obtener privilegios de superusuario
- Para resolver este problema existen tres soluciones:
  - Deshabilitar *register_globals* en el fichero php.ini
  - Inicializar las variables
  - Establecer el orden de las variables en PHP

Variables globales

- Deshabilitar *register_globals* en el fichero php.ini
  - La directiva *register_globals* del fichero php.ini establece si se admite o no la creación automática de variables globales
  - A partir de PHP 4.2.0 el valor por defecto de esta directiva es off, que es el valor recomendable

Variables globales

- Inicializar las variables
  - El problema anterior se soluciona dando un valor inicial a la variable $superuser:

```
<?PHP
  $superuser = false;
  if (comprueba_privilegios())
        $superuser = true;
  ...
?>
```

Variables globales

- Inicializar las variables
  - Es recomendable inicializar todas las variables antes de usarlas. Se puede usar la directiva *error_reporting*=E_ALL en php.ini para que se muestre un aviso cuando se use una variable que no haya sido previamente inicializada
  - En un entorno de producción debe evitarse la aparición de mensajes de aviso o error. Para ello se utilizan las siguientes directivas en php.ini:

```
display_errors = off
log_errors = on
error_log = /var/log/php_errors.log
```

  - Los errores irán al fichero especificado en lugar de mostrarse en la pantalla

Variables globales

- Establecer el orden de las variables en PHP
  - PHP crea automáticamente variables globales a partir del entorno (E), las *cookies* (C), la información del servidor (S) y los parámetros GET (G) y POST (P)
  - La directiva *variables_order* controla el orden de estas variables. El valor por defecto es “EGPCS”
  - Permitir la creación de variables globales desde parámetros GET y POST y desde *cookies* es potencialmente peligroso. Un posible valor para *variables_order* que evita esto es “ES”
  - En tal caso para acceder a los parámetros de los formularios y a las *cookies* se deben utilizar los arrays globales $_REQUEST, $_GET, $_POST y $_COOKIES

Variables globales

- Establecer el orden de las variables en PHP
  - Si se modifican las directivas *register_globals* y/o *variables_order* es preciso revisar los *scripts* existentes para adaptarlos a las nuevas circunstancias
  - Una forma puede ser la siguiente:

```
$edad = $_REQUEST[‘edad’];
...
```

Nombres de ficheros

- Es relativamente fácil construir un nombre de fichero que se refiera a algo distinto a lo que se pretende
- Sea el siguiente código:

```
include (“/usr/local/lib/bienvenida/$username”);
```

- Este código pretende mostrar un mensaje de bienvenida personalizado para el usuario. Aparentemente no es peligroso, pero ¿qué ocurriría si el usuario introduce como nombre la cadena “../../../../etc/passwd”?
  - Se mostraría el fichero de passwords del sistema

Nombres de ficheros

- Además hay que tener en cuenta que las funciones de manejo de ficheros como include() o require() admiten nombres de **ficheros remotos**, lo que podría provocar la ejecución de código maligno cargado de otro servidor. Sea, por ejemplo, el código

```
include ($libdir . "/conecta.php");
```

- Si un atacante modifica el valor de la variable $libdir a, pongamos por caso, "http://atacante/", y coloca en la raíz del mismo un fichero de nombre conecta.php, su código sería ejecutado
- Se puede desactivar la funcionalidad de acceso a ficheros remotos con la siguiente directiva en php.ini:

```
allow_url_fopen = off
```

Nombres de ficheros

- Para chequear nombres de ficheros se utilizan las funciones **realpath**() y **basename**(). La primera convierte direcciones relativas en absolutas y la segunda toma una ruta y devuelve la parte correspondiente al nombre del fichero. Ejemplo:

```
$file = $_POST['username'];
$file2 = basename (realpath($file));
if ($file2 != $file)
   die ("$file no es un username válido");
include ("/usr/local/lib/bienvenida/$file");
```

Nombres de ficheros

- Otra defensa contra los nombres de ficheros incorrectos es la directiva de php.ini open_basedir:

```
open_basedir = /alguna/ruta
```

- PHP limitará las operaciones sobre ficheros al directorio especificado y sus subdirectorios:

```
include (“/alguna/ruta/lib.inc”); // permitido
include (“/otra/ruta/lib.inc”);   // da error
```

Subida de ficheros

- La subida de ficheros permite a un usuario enviar cualquier fichero al servidor, lo cual encierra un gran peligro ya que un atacante puede subir un código maligno y luego ejecutarlo, causando más daño que cuando se incluye el código desde un servidor remoto
- Como recomendación general, debe evitarse utilizar el nombre enviado por el navegador (podría ser, por ejemplo, /etc/passwd). Es conveniente generar un nombre único para el fichero subido

Subida de ficheros

- Otro peligro es el tamaño de los ficheros. Aunque se limite el tamaño máximo en el formulario, los ficheros se reciben automáticamente y luego se comprueba su tamaño
- Es posible que un usuario intente provocar un ataque de denegación de servicio enviando varios ficheros de gran tamaño a la vez y llenando el sistema de ficheros utilizado por PHP para almacenarlos
- Para evitar esto se puede utilizar la directiva *post_max_size* de php.ini. El valor por defecto suele ser más elevado de lo necesario
- El campo oculto MAX_FILE_SIZE en los formularios es conveniente porque evita que comience la subida de un fichero si supera el tamaño permitido, pero puede saltarse con facilidad, por lo que no es suficiente. La directiva de php.ini sí lo es

Bibliotecas

- Es conveniente almacenar los ficheros de biblioteca fuera de la raíz de la web para evitar que puedan ser accedidos por su URL
- En tal caso debe hacerse saber a PHP la ubicación de los ficheros indicando la ruta completa en los include() y require() o bien mediante la directiva *include_path* en php.ini

```
include_path = ".:/usr/local/php:/usr/local/lib/myapp"
```

- Esto es particularmente importante cuando en el código de la biblioteca aparecen *passwords*, como es el caso de las funciones de conexión con bases de datos

Formularios

- Es recomendable validar todos los datos provenientes de formularios para asegurarse de que los valores recibidos son los esperados
- En general, cualquier información proveniente del exterior debe contemplarse como posiblemente contaminada y debe ser verificada antes de ser utilizada
- Sea el siguiente ejemplo:

```
print (“Nombre: “ . $nombre);
print (“Comentario: “ . $comentario);
```

Formularios

- Si el autor del comentario introdujo algún código HTML en el texto del mismo, el código será interpretado y sus efectos podrían ser graves
- Para evitar esto se puede utilizar la función **htmlspecialchars**(), que impide que se interpreten los caracteres especiales de HTML (<, >, &)
- El código quedaría de la siguiente manera:

```
print (“Nombre: “ . $nombre);
print (“Comentario: “ . htmlspecialchars($comentario));
```

Inyección SQL

- Inyección
  - Consiste en inyectar en la aplicación datos introducidos por el usuario. Esto es muy habitual y de por sí no es peligroso
  - Ejemplo: sea la instrucción

```
sql= "SELECT * FROM noticias WHERE id = $id";
```

  - Pulsando en el artículo de interés para el usuario se convierte en:

```
sql= "SELECT * FROM noticias WHERE id = 228";
```

Inyección SQL

- Inyección SQL
  - Consiste en inyectar un mandato dentro de una consulta SQL. Sea la consulta:

```
$consulta = “SELECT titulo FROM libros WHERE codigo
  = $codigo”;
```

  - siendo `$codigo` un valor introducido desde un formulario. Si el valor es `‘23’` la consulta será:

```
SELECT titulo FROM libros WHERE codigo = 23
```

  - Si el valor es `’23; DROP TABLE users’` la consulta es:

```
SELECT titulo FROM libros WHERE codigo = 23; DROP
  TABLE users
```

  - que destruiría la tabla de usuarios de MySQL

Inyección SQL

- Inyección SQL
  - Sea ahora el siguiente código muy habitual en una aplicación Web:

```
$consulta = “SELECT id FROM usuarios WHERE username
  = ‘$username’ AND password = ‘$password’”;
```

  - Si se introducen los valores **juan** como **username** y **Ag3n.da** como **password**, la consulta queda:

```
SELECT id FROM usuarios WHERE username = ‘juan’ AND
  password = ‘Ag3n.da’
```

Inyección SQL

- Inyección SQL
  - Se puede saltar la comprobación del password introduciendo el valor `juan'--` como username o el valor `' OR ''='` como password. Las consultas que quedarían en ambos casos son, respectivamente:

```
SELECT id FROM usuarios WHERE username = 'juan'--'
  AND password = ''
SELECT id FROM usuarios WHERE username = 'juan' AND
  password = '' OR ''=''
```

  - En el primer caso nótese que `--` es un comentario de línea en MySQL y provoca que se ignore todo lo que viene tras él en la línea

Inyección SQL

- Inyección SQL
  - La inyección SQL puede utilizarse para:
    - Cambiar valores de las consultas
    - Concatenar varias consultas
    - Añadir llamadas a función y procedimientos almacenados a una consulta
- Para evitar la inyección SQL es muy importante **validar los valores que se han de integrar en la consulta SQL**. En el primer caso, por ejemplo, `$codigo` debe ser un valor entero

Resumen

- De todo lo anterior podemos concluir las recomendaciones siguientes:
  - Validar todos los datos de entrada de la aplicación
    - **Aceptar únicamente datos válidos conocidos**
    - Rechazar datos no válidos conocidos
    - Sanear todos los datos
  - Configurar adecuadamente PHP a través del fichero php.ini
  - Seguir unas buenas prácticas en la programación
- Hay que tener en cuenta que cualquier cambio en la configuración de PHP afectará a los *scripts* de todos los usuarios y posiblemente a algunas herramientas, lo cual debe ser tenido en cuenta y estudiarse sus consecuencias antes de proceder a realizarlos