

# Seguridad en PHP: XSS, SQLI, CMDI y LFI

Mario Daniel Castro Almenzar

18 de mayo de 2023



UNIVERSIDAD  
DE GRANADA

## Índice

<b>1</b>	<b>Introducción</b>	<b>3</b>
<b>2</b>	<b>Seguridad en PHP. Principales amenazas</b>	<b>4</b>
2.1	XSS: Cross site scripting . . . . .	4
2.1.1	XSS No Persistente . . . . .	5
2.1.2	XSS Persistente . . . . .	5
2.2	CMD Injection . . . . .	7
2.3	SQL Injection . . . . .	9
2.4	LFI: Local File Inclusion . . . . .	12
<b>3</b>	<b>Conclusión</b>	<b>14</b>
<b>4</b>	<b>Bibliografía</b>	<b>15</b>

---

## Índice de figuras

1	Introducción . . . . .	3
2	Ejemplo ataque XSS Reflejado . . . . .	4
3	XSS No persistente . . . . .	5
4	Ejemplo validación de Datos . . . . .	6
5	Ejemplo sanitización de Datos . . . . .	6
6	Ejemplo comando CMD Injection . . . . .	7
7	Esquema SQL Injection . . . . .	9
8	Ejemplo básico de Local File Inclusion . . . . .	12
9	Ejemplo para evitar Path Traversal . . . . .	13

# 1. Introducción

PHP es uno de los lenguajes de programación más utilizados para el desarrollo del **backend** de servidores debido a su flexibilidad, su compatibilidad con diversas bases de datos y su facilidad de integración con HTML.

Con PHP, los desarrolladores pueden crear sitios web dinámicos que interactúan con bases de datos y ofrecen experiencias personalizadas a los usuarios. Sin embargo, los ciberdelincuentes hacen uso de este lenguaje para explotar las vulnerabilidades de los servidores web y poder acceder a la información almacenada o cualquier otra intención maliciosa.

Por esta razón, es de suma importancia que los desarrolladores sean conscientes y apliquen las mejores prácticas de seguridad y cómo implementarlas en su código. Ignorar la seguridad en PHP puede dar lugar a una serie de problemas, como la inyección de SQL, el Cross-Site Scripting (XSS), la inyección de comandos y la inclusión de archivos locales (LFI), entre otros.

En esta memoria, analizaremos en profundidad algunas de las amenazas de seguridad en PHP más comunes, proporcionaremos ejemplos de cómo pueden ser explotadas y discutiremos estrategias para mitigar estos riesgos y proteger nuestro servidor web.



Figura 1: Introducción

## 2. Seguridad en PHP. Principales amenazas

### 2.1. XSS: Cross site scripting

El **Cross site scripting** es una vulnerabilidad que permite inyectar un script malicioso en un sitio web para luego ser procesado y ejecutado. Usualmente, este procedimiento se fundamenta en la confianza que el sitio web deposita en la entrada de datos.

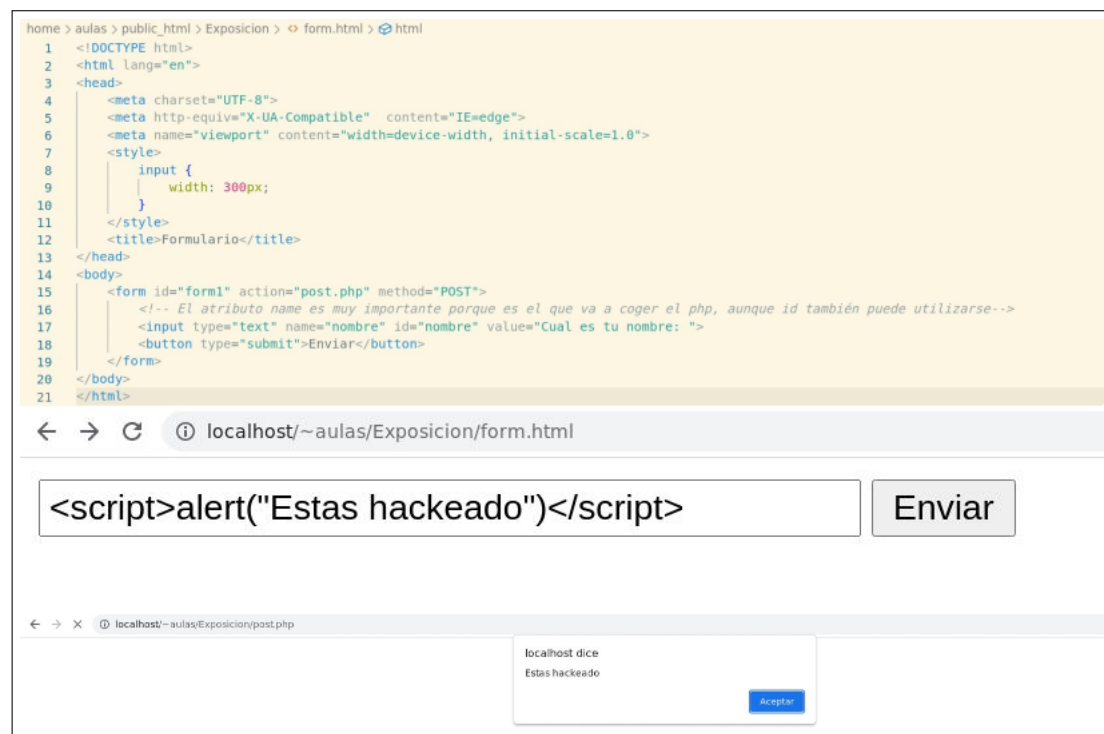


Figura 2: Ejemplo ataque XSS Reflejado

Una de las formas de realizar este tipo de ataques consiste en enviar intencionalmente una URL con el *payload*<sup>1</sup> al usuario objetivo con el propósito de obtener información personal del usuario, cookies de sesión o llevar a cabo técnicas de ingeniería social, entre otras prácticas.

Existen dos tipos de Ataques XSS: Persistente y No Persistente.

<sup>1</sup>conjunto de datos transmitidos útiles, que se obtienen de excluir cabeceras, metadatos, información de control y otros datos que son enviados para facilitar la entrega del mensaje.

### 2.1.1. XSS No Persistente

El **XSS No Persistente**, también conocido como *reflejado*, es un tipo de ataque en el que el script malicioso es parte de la solicitud que se envía al servidor web. No se almacena de manera persistente en el servidor o en la base de datos. En su lugar, el script se refleja fuera del servidor web, de ahí el término "reflejado". El ejemplo mostrado anteriormente se califica como XSS No Persistente.

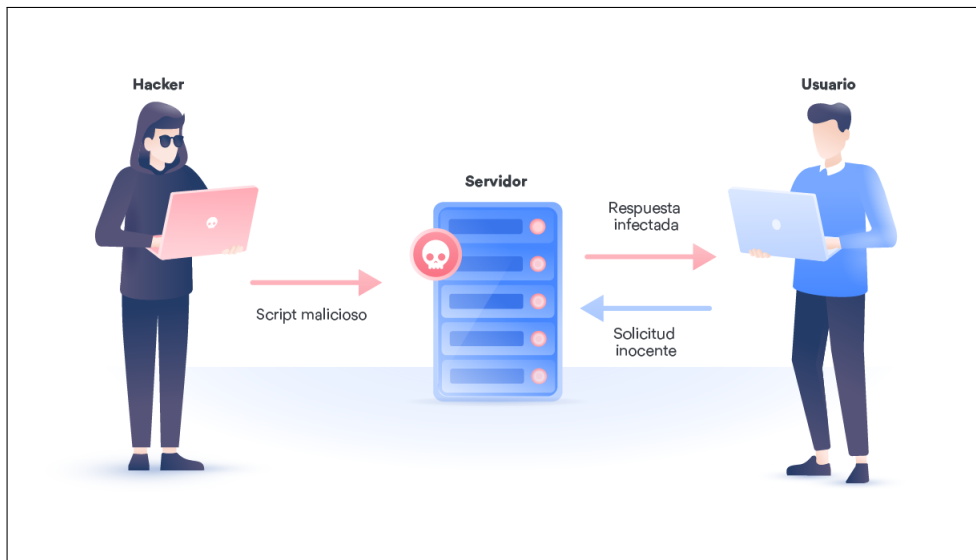


Figura 3: XSS No persistente

A pesar de que el código que se inyecta es en la página web, los que se ven afectados son los usuarios finales de la web, pudiendo robarse sus cookies, redirigir a otras web, o ejecutar código malicioso para ganar acceso a la máquina del usuario.

### 2.1.2. XSS Persistente

El **XSS Persistente**, también conocido como *almacenado*, es un ataque más peligroso que el XSS no persistente. En este caso, el script malicioso es enviado al servidor y almacenado de manera persistente. Esto puede suceder en varias partes del sitio web, como en comentarios de usuarios, publicaciones en foros, o en cualquier otro lugar donde los usuarios puedan ingresar datos que se guardarán en el servidor. Es un problema que debemos de solucionar de cara al Proyecto Final de prácticas para nuestro foro de quejas.

Al igual que con el XSS no persistente, los ataques XSS persistentes pueden resultar en robo de cookies, ejecución de acciones no autorizadas, y otros abusos de la confianza que el sitio tiene en el usuario.

Para prevenir este tipo de ataques, conviene que sigamos una serie de reglas de validación desde el servidor. Existen distintas técnicas como:

- **Data validation:** Asegurarnos de que los datos que se van a enviar por medio del formulario se rijan por un control específico, por ejemplo por expresiones regulares.

```
datos_validados.php
1  <?php
2  $email = $_POST['email'];
3
4  // Verificar si la dirección de correo es válida
5  if (filter_var($email, FILTER_VALIDATE_EMAIL)) {
6      echo("La dirección de correo electrónico es válida");
7  } else {
8      echo("La dirección de correo electrónico no es válida");
9  }
10 ?>
```

Figura 4: Ejemplo validación de Datos

- **Data sanitation:** Hacer que los datos cumplan con unas condiciones específicas y que ejecuten un resultado preciso de los datos .

```
datos_sanitizados.php
1  <?php
2  $data = $_POST['data'];
3
4  // Limpiar la cadena de entrada
5  $dataSanitized = filter_var($data, FILTER_SANITIZE_STRING);
6
7  echo "Los datos saneados son: " . $dataSanitized;
8  ?>
9
```

Figura 5: Ejemplo sanitización de Datos

Por medio de los métodos anteriores también conseguimos que se produzca un **output scraping**<sup>2</sup>.

Existen otras alternativas que no implican output scraping, como el uso de `html_entities()`

---

<sup>2</sup>Poder filtrar caracteres especiales que puedan ser interpretados por el navegador, por ejemplo los símbolos positivo y negativo.

## 2.2. CMD Injection

**Command Injection** (o inyección de comandos) es una vulnerabilidad que permite a un atacante inyectar y ejecutar comandos directamente en el sistema operativo en el que se ejecuta nuestra aplicación.

Al explotarla, los atacantes pueden generar distintos riesgos para un sitio web, entre ellos:

- Robar credenciales de los usuarios o del sistema donde corre la aplicación.
- Dejar temporalmente inaccesibles las conexiones al **backend** (SSH, Escritorio Remoto).
- Colocar aplicaciones maliciosas en el servidor.
- Modificar la aplicación (cambiar datos o incluso borrarla completamente).
- Escalar privilegios y obtener el control total del servidor.

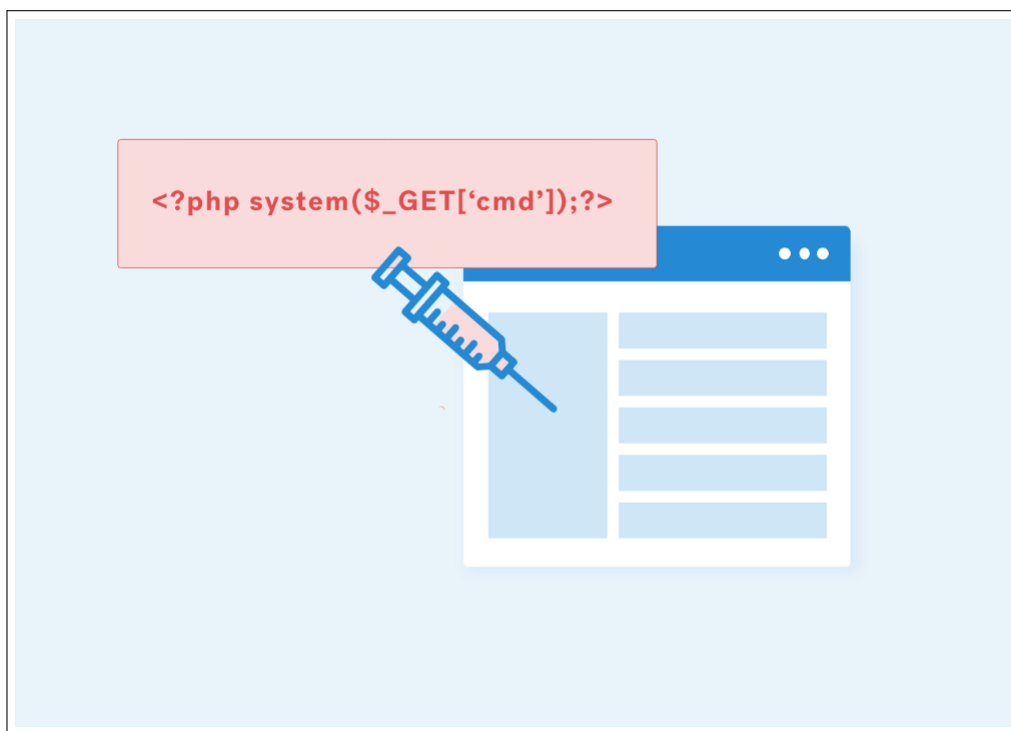


Figura 6: Ejemplo comando CMD Injection

Con PHP se pueden hacer **CMD Injections** cuando se utilizan funciones como `exec()`, `system()`, `passthru()`, `shell_exec()`, etc., que permiten ejecutar comandos del sistema operativo y no se valida o se escapa correctamente la entrada del usuario.

Un ejemplo de esto podría ser:

```
<?php
$cmd = $_GET['cmd'];
system($cmd);
?>
```

En este caso, un atacante podría ejecutar cualquier comando en el servidor simplemente pasándolo a través del parámetro `cmd` en la URL, por ejemplo:

`http://example.com/mypage.php?cmd=ls`

Para proteger nuestro código PHP contra la inyección de comandos, se debe:

- **Validar la entrada del usuario:** debemos asegurarnos de que la entrada del usuario es exactamente lo que se espera. Por ejemplo, si se espera un número, hay que cerciorarse de que la entrada sea un número.
- **Escapar la entrada del usuario:** Si necesitamos usar la entrada del usuario en un comando del sistema, debemos escaparlos correctamente. PHP tiene funciones como `escapeshellarg()` y `escapeshellcmd()` que pueden hacer esto.
- **Usar funciones de alto nivel cuando sea posible:** En lugar de ejecutar comandos del sistema operativo, es mejor usar funciones de alto nivel de PHP que hagan lo mismo. Por ejemplo, en lugar de usar `exec('rm ' . $file)`, es mejor usar `unlink($file)`.
- **Minimizar los permisos del usuario PHP:** es mejor ejecutar scripts php como un usuario con los permisos mínimos requeridos. De esta manera, incluso si un atacante logra ejecutar un comando, no podrá hacer mucho daño.
- **Usar funciones de seguridad de PHP:** PHP tiene funciones como `disable_functions` y `open_basedir` que pueden limitar las funciones y comandos que se pueden ejecutar.
- **Realizar pruebas de seguridad periódicas,** como las pruebas de penetración, para identificar y corregir las posibles vulnerabilidades.



## 2.3. SQL Injection

La inyección SQL es una técnica de inyección de código en bases de datos que consiste en la colocación de código malicioso mediante sentencias SQL, a través de la entrada de una página web.

Generalmente ocurre cuando se le pide a un usuario que ingrese, como su nombre de usuario/ID de usuario, y en lugar de un nombre/ID, el usuario le da una instrucción SQL que, sin saberlo, ejecutará en su base de datos. Por ejemplo:

```
txtUserId = getRequestString("UserId");  
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

Una de las primeras reglas que se aprenden acerca de SQLI es aquella basada en que  $1=1$  es siempre **true**. La idea es crear una instrucción para seleccionar un usuario con una identificación de usuario dada. En caso de que no haya validación para entradas "incorrectas", podría ingresarse una entrada como la siguiente:

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

Esta sentencia devolverá TODAS las filas de la tabla "Usuarios", ya que  $OR\ 1=1$  siempre es **true**. Ya podemos hacernos una pequeña idea de la peligrosidad de este tipo de inyecciones, por ejemplo, ¿y si la tabla Usuarios contiene tanto nombre como contraseñas?

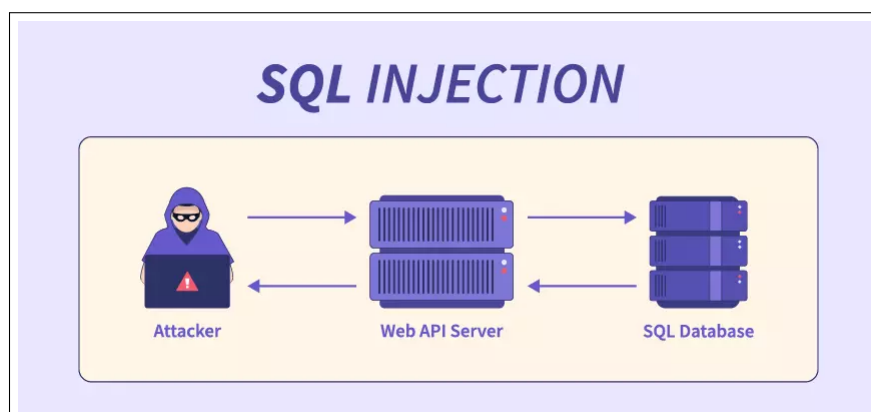


Figura 7: Esquema SQL Injection

A continuación vamos a ver cómo se podría evitar este tipo de inyecciones en formularios, de modo que se saniticen las cadenas de texto del input:

```
<?php

function limpiar_cadena($cadena){

    // También sirve para evitar ataques XSS:
    $cadena=str_ireplace("<script>", "", $cadena);
    $cadena=str_ireplace("</script>", "", $cadena);
    $cadena=str_ireplace("<script src=", "", $cadena);
    $cadena=str_ireplace("<script type=", "", $cadena);

    $cadena=str_ireplace("SELECT * FROM", "", $cadena);
    $cadena=str_ireplace("DELETE FROM", "", $cadena);
    $cadena=str_ireplace("INSERT INTO", "", $cadena);
    $cadena=str_ireplace("DROP TABLE", "", $cadena);
    $cadena=str_ireplace("DROP DATABASE", "", $cadena);
    $cadena=str_ireplace("TRUNCATE TABLE", "", $cadena);
    $cadena=str_ireplace("SHOW TABLES", "", $cadena);
    $cadena=str_ireplace("SHOW DATABASES", "", $cadena);

    $cadena=str_ireplace("<?php", "", $cadena);
    $cadena=str_ireplace(">", "", $cadena);
    $cadena=str_ireplace("^", "", $cadena);
    $cadena=str_ireplace("<", "", $cadena);
    $cadena=str_ireplace("[", "", $cadena);
    $cadena=str_ireplace("]", "", $cadena);
    $cadena=str_ireplace("==", "", $cadena);
    $cadena=str_ireplace(";", "", $cadena);
    $cadena=str_ireplace(":", "", $cadena);

    $cadena=trim($cadena);
    $cadena=stripslashes($cadena);

    return $cadena;
}

?>
```

Del código anterior, conviene conocer que:

- La llamada a la función `trim()` sirve para limpiar espacios en blanco al principio y al final de la cadena. Por ejemplo:  
" Mario " -> "Mario"

- La llamada a la función `stripslashes()` sirve para quitar las barras de un string con comillas escapadas. Por ejemplo:  
`" Is your name O\' reilly? " -> " Is your name O'reilly? "`
- La llamada a la función `str_ireplace()` sirve para lo mismo que `str_replace()` pero es insensible a mayúsculas y minúsculas. Esta última reemplaza todas las apariciones de la cadena de búsqueda con la cadena de reemplazo.

Por ejemplo:

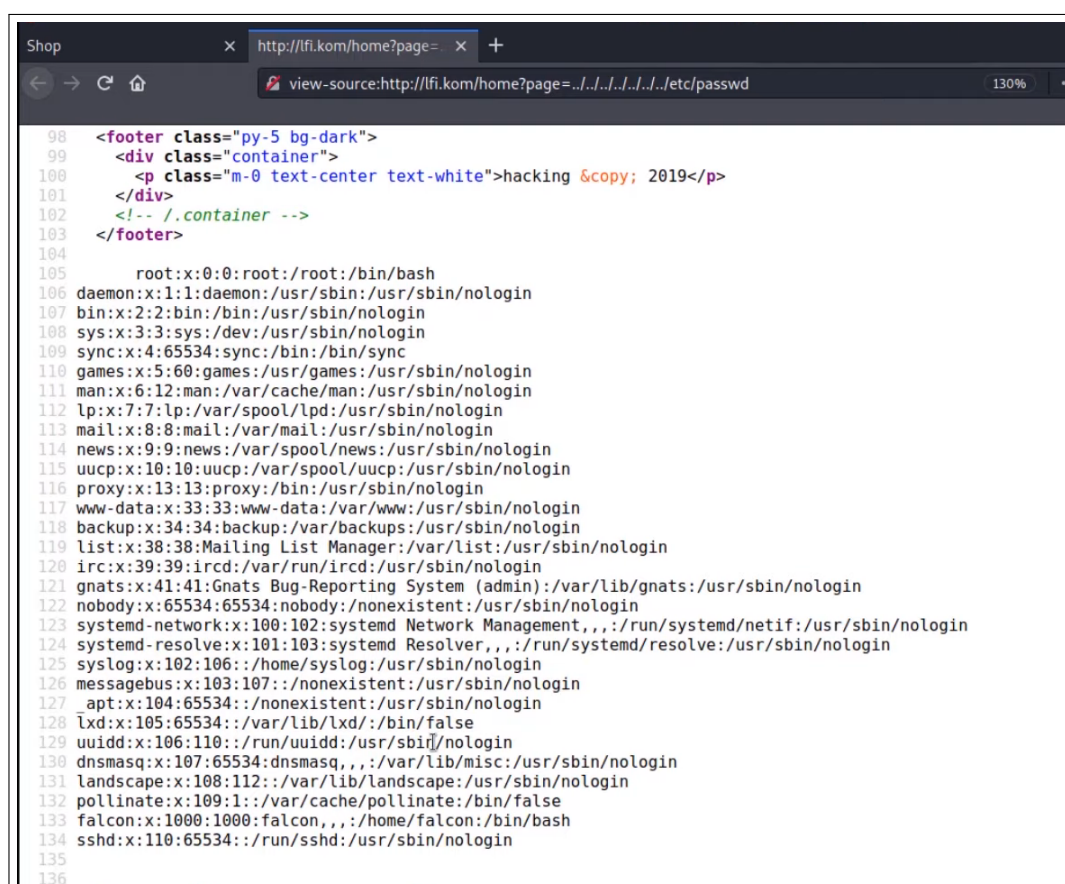
```
<?php
$texto "<script> Hola mundo </script>";
echo limpiar_cadena($texto);
Output "Hola mundo"
?>
```

## 2.4. LFI: Local File Inclusion

Local File Inclusion o LFI es una vulnerabilidad web que le permite a un atacante ejecutar código PHP por medio de un archivo almacenado en el servidor de la aplicación.

Por ejemplo, si accedemos a una url que no esté protegida correctamente y aplicamos la siguiente sentencia podremos acceder al contenido de `/etc/passwd`:

`http://ejemplo.com/home?page=../../../../../../../../etc/passwd`



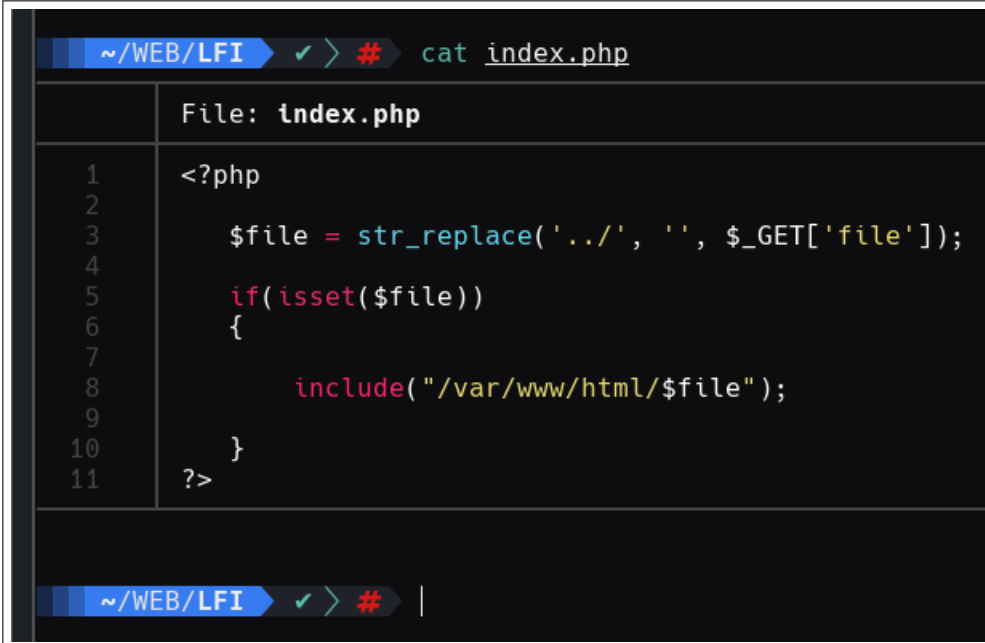
```
98 <footer class="py-5 bg-dark">
99 <div class="container">
100 <p class="m-0 text-center text-white">hacking &copy; 2019</p>
101 </div>
102 <!-- /.container -->
103 </footer>
104
105 root:x:0:0:root:/root:/bin/bash
106 daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
107 bin:x:2:2:bin:/bin:/usr/sbin/nologin
108 sys:x:3:3:sys:/dev:/usr/sbin/nologin
109 sync:x:4:65534:sync:/bin:/bin/sync
110 games:x:5:60:games:/usr/games:/usr/sbin/nologin
111 man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
112 lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
113 mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
114 news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
115 uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
116 proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
117 www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
118 backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
119 list:x:38:38:Mail List Manager:/var/list:/usr/sbin/nologin
120 irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
121 gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
122 nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
123 systemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/usr/sbin/nologin
124 systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin
125 syslog:x:102:106:./home/syslog:/usr/sbin/nologin
126 messagebus:x:103:107:./nonexistent:/usr/sbin/nologin
127 apt:x:104:65534:./nonexistent:/usr/sbin/nologin
128 lxd:x:105:65534:./var/lib/lxd:/bin/false
129 uidd:x:106:110:./run/uidd:/usr/sbin/nologin
130 dnsmasq:x:107:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
131 landscape:x:108:112:./var/lib/landscape:/usr/sbin/nologin
132 pollinate:x:109:1:./var/cache/pollinate:/bin/false
133 falcon:x:1000:1000:falcon,,,:/home/falcon:/bin/bash
134 sshd:x:110:65534:./run/sshd:/usr/sbin/nologin
135
136
```

Figura 8: Ejemplo básico de Local File Inclusion

Accediendo a la información anterior podemos conseguir la clave privada RSA para iniciar sesión sin contraseña. Esta es solo uno de los miles de huecos que pueden generarse en la barrera de seguridad de una página web si no se establecen mecanismos de prevención adecuados.

No obstante, entre estos agujeros de seguridad, se incluye la posibilidad de obtener una reverse shell. Existen varios métodos que le permiten evitar vulnerabilidades LFI en nuestra página web:

- Evitar por completo pasar nombres de archivo en la entrada del usuario. Esto incluye no solo la entrada directa del usuario, sino también otras fuentes de datos que el atacante puede manipular, por ejemplo, las cookies.
- Si la web requiere que use nombres de archivo de la entrada del usuario y no hay forma de evitarlo, se debe crear una *whitelist* de archivos seguros.
- En caso de no poder crear una *whitelist* porque se usan nombres de archivo arbitrarios (por ejemplo, si los usuarios cargan los archivos), deben almacenarse los nombres de archivo en la base de datos y usar identificadores de fila de tabla en la entrada del usuario. También se pueden usar asignaciones de URL para identificar archivos sin riesgo de inclusión de archivos locales. Esto último se suele usar con mucha frecuencia.



The image shows a terminal window with a dark background. At the top, a prompt shows the current directory as `~/WEB/LFI` and the command `cat index.php` has been executed. Below the prompt, the contents of `index.php` are displayed, with line numbers 1 through 11 on the left. The PHP code is as follows:

```
1 <?php
2
3     $file = str_replace('../', '', $_GET['file']);
4
5     if(isset($file))
6     {
7
8         include("/var/www/html/$file");
9
10    }
11    ?>
```

At the bottom of the terminal, the prompt `~/WEB/LFI` is visible again, followed by a vertical bar cursor.

Figura 9: Ejemplo para evitar Path Traversal

### 3. Conclusión

La **seguridad en servidores web es esencial** en la era actual de la información, donde los ciberataques son una amenaza constante. Las vulnerabilidades discutidas en este documento, como XSS, Command Injection, SQL Injection y Local File Inclusion (LFI), son solo algunas de las muchas posibles debilidades que pueden ser explotadas por actores malintencionados para comprometer un sistema.

El **XSS** puede ser prevenido mediante una adecuada validación y escape de la entrada del usuario, así como con el uso de encabezados HTTP y políticas de seguridad de contenido para controlar cómo y dónde se pueden cargar los scripts.

Por otra parte, las **CMD Injection** se pueden prevenir asegurándose de que los comandos del sistema operativo no se ejecuten con entrada del usuario, o si es absolutamente necesario, validando y escapando la entrada del usuario de manera segura. Asimismo, es esencial minimizar los permisos de los scripts PHP y realizar pruebas de seguridad periódicas.

La **inyección SQL** puede ser mitigada evitando la concatenación de consultas SQL y utilizando consultas preparadas o procedimientos almacenados, lo que asegura que la entrada del usuario se trate siempre como datos y no como parte de la consulta SQL. También es beneficioso emplear una estrategia de defensa en profundidad que incluya la limitación de los privilegios de la base de datos y la realización de auditorías de seguridad regulares.

**Para prevenir LFI**, se debe evitar pasar nombres de archivos en la entrada del usuario siempre que sea posible. Si esto no puede evitarse, entonces los nombres de archivo seguros deben ser almacenados en una lista blanca o la entrada del usuario debe limitarse a identificadores seguros.

**En conclusión**, garantizar la seguridad de un sitio web es un proceso continuo que requiere una comprensión de las posibles vulnerabilidades y cómo mitigarlas. Las buenas prácticas de codificación, el uso de funciones de seguridad adecuadas y la realización regular de pruebas de seguridad son esenciales para mantener una defensa robusta contra las amenazas de seguridad cibernética.

## 4. Bibliografía

---

- ¿Qué es Payload?  
<https://ayuda.acens.com/hc/es/articles/360018220377>
- ¿Qué es el XSS persistente?  
<https://keepcoding.io/blog/que-es-el-xss-persistente/>
- HTML Entities  
<https://www.php.net/manual/en/function.htmlentities.php>
- ¿Qué es CMD Injection y cómo prevenirla?  
<https://blog.hackmetrix.com/command-injection>
- SQL Injection  
[https://www.w3schools.com/sql/sql\\_injection.asp](https://www.w3schools.com/sql/sql_injection.asp)
- Función `str_replace()`  
<https://www.php.net/manual/en/function.str-replace.php>
- ¿Qué es Local File Inclusion?  
<https://keepcoding.io/blog/que-es-local-file-inclusion/>