
Hashing, Cifrado y Certificados

Laura Bezanilla Matellán

2 de noviembre de 2022

Índice

PARTE 1	2
1. Resumen de cómo se emplean md5sum y sha1sum para generar hashcodes	2
2. Diferencias al construir el hash asociado a diferentes cadenas de caracteres	3
3. Obtener el hashcode de un fichero usando ambas herramientas.....	4
4. Herramienta hashid	5
5. Modificar propiedades de un fichero y volver a obtener el hashcode	6
6. Herramienta John The Ripper	7
7. Programa buildhash para calcular los hashcode del fichero y sus propiedades	9
8. Programa checkhash para comprobar que los ficheros no se han modificado.....	10
 PARTE 2	 11
1. Dibuja el diagrama de red lo más detallado posible	11
2. Configurar la máquina de <i>mallet</i> para que actúe como <i>router</i>	11
3. Configuración de la máquina de <i>bob</i>	15
4. Técnica de <i>pivoting</i> de post-explotación	16
5. Configuración de la máquina de <i>alice</i>	16
6. Pruebas a nivel de red para comprobar que las máquinas se pueden comunicar.....	17
7. Diferencias entre las reglas de tipo INPUT, OUTPUT y FORWARD en <i>iptables</i>	19
8. Configurar el firewall para que solo se puedan realizar peticiones de tipo ICMP	20
9. Configurar el firewall para que solo una máquina pueda conectarse por ssh con otra ..	24
10. Conclusiones	26

PARTE 1

1. Resumen de cómo se emplean md5sum y sha1sum para generar hashcodes

La criptografía proporciona diferentes herramientas que sirven para proteger los datos en la red. Actualmente, lo recomendable es que en aquellas aplicaciones informáticas donde se tenga que almacenar tanto el *login* como el *password* de los usuarios se deberían aplicar técnicas criptográficas para proteger dicha información sensible.

¿Cómo se puede proteger la información en la red? Para contestar correctamente a esa pregunta es necesario explicar que para poder aplicar la criptografía simétrica se necesitan tres cosas:

- Un algoritmo.
- Un mensaje en texto claro que se quiere proteger.
- Una clave para poder generar el criptograma.

Teniendo el algoritmo y el mensaje solo hay que aplicarle la clave para generar el criptograma. Con el método inverso, si se coge el criptograma y se le aplica el mismo algoritmo y clave, que previamente se utilizó para cifrar, se obtendrá el mensaje en texto claro.

Sin embargo, para generar un *hash* lo que es verdaderamente necesario es tener una función.

Las funciones de *hash* son herramientas criptográficas cuyas características son:

- Son funciones de una sola vía. Es decir, desde el punto de vista computacional, en términos de tiempo y procesador, es muy fácil dado un texto claro obtener la función de hash. Sin embargo, dado un resumen según la teoría es complicado obtener el texto que ha generado dicho resumen.
- El resumen que proporcionan tiene la misma longitud independientemente del tamaño de la entrada proporcionada. La entrada de una función hash puede ser diversa desde un texto a un fichero. Por este motivo adoptan comúnmente el nombre de funciones resumen.
- Las funciones *hash* garantizan la integridad de la información. En términos informáticos, la integridad significa que la información no puede ser modificada. Si se modifica un solo bit del texto claro, la salida generada va a ser totalmente diferente. En la actualidad se utilizan como mecanismos para poder demostrar si la información ha sido alterada o no.

- Las funciones de *hash* son vulnerables al problema de las colisiones. Existe una remota posibilidad de que dos mensajes de entrada diferentes produzcan el mismo mensaje de salida. Debido a este hecho no se puede garantizar la integridad de la información.

La aplicación MD5 genera un resumen de 32 caracteres en hexadecimal, es decir 128 bits. En el caso de SHA1 se genera un resumen de 40 caracteres en hexadecimal o 160 bits. Esto significa que para cualquier entrada que se elija el resumen siempre va a tener uno de estos dos tamaños dependiendo de la función *hash* utilizada.

Aunque en el pasado han sido usadas y adoptadas de forma generalizada, se han descubierto varias vulnerabilidades durante los últimos años. Por lo que actualmente se recomienda usar algoritmos más seguros como SHA-256 o SHA-512 ya que producen un resumen con un mayor número de bits.

Ambas herramientas pueden ser útiles para comprobar la integridad de un fichero tras una descarga, por ejemplo, pero ya no son aceptables desde el punto de vista del criptoanálisis.

2. Diferencias al construir el hash asociado a diferentes cadenas de caracteres

MD5SUM es una herramienta que permite obtener el resumen MD5 de una cadena de caracteres. Otra posibilidad es poder calcular el resumen a partir de un fichero. La máquina de *mallet* tiene en su directorio *home* un fichero ya preparado previamente para poder calcular su resumen como se muestra en la siguiente imagen.

```
mallet@mallet:~$ ls
claves.txt Desktop Downloads Exploits testttttt.pdf
mallet@mallet:~$
mallet@mallet:~$ md5sum testttttt.pdf
76379ef9ddf935d80b397646ece7f0bc testttttt.pdf
```

Figura 1. Obtener resumen MD5 a partir de un fichero

Existe otra opción que sería obtener el resumen a partir de una cadena de caracteres. Esto se puede hacer gracias a las tuberías que proporciona Linux. Su funcionamiento es simple, la salida que produce el primer comando es redireccionada como entrada al segundo comando. Así se podría hacer con varios comandos.

```
mallet@mallet:~$ echo hola | md5sum
916f4c31aaa35d6b867dae9a7f54270d -
```

Figura 2. Obtener resumen MD5 a partir de un texto

Para poder comprobar que realmente esta herramienta ha generado un *hash* de 32 caracteres en hexadecimal se utilizará las tuberías junto con el comando `wc` para contar el número de caracteres que tiene el resumen.

```
mallet@mallet:~$ echo hola | md5sum
916f4c31aaa35d6b867dae9a7f54270d -
mallet@mallet:~$
mallet@mallet:~$
mallet@mallet:~$ echo hola | md5sum | wc
1      2      36
```

Figura 3. Comprobación del número de caracteres del resumen

En la *Figura 3* se puede observar cómo ese comando ha devuelto que el número de caracteres son 36 en una sola línea con dos palabras. Anteriormente se ha explicado que MD5 generaba resúmenes de 32 caracteres, no de 36. Esto se debe a que al final del *hash* se han añadido 4 caracteres los cuales están resaltados en verde.

Por otro lado, en el caso de querer obtener el resumen con la herramienta SHA1, esta herramienta va a generar 40 caracteres en hexadecimal como se puede comprobar en la siguiente imagen.

```
mallet@mallet:~$ shasum testttttt.pdf
bcf3202e577284717465cbe7d6ebbd76e7223dc0 testttttt.pdf
mallet@mallet:~$
mallet@mallet:~$ echo pepe | shasum | wc
1      2      44
```

Figura 4. Obtener resumen SHA1 a partir de un fichero

3. Obtener el hashcode de un fichero usando ambas herramientas

En esta sección se va a poner un ejemplo para comprobar que de verdad las funciones de *hash* sirven para garantizar la integridad de la información.

Se va a crear un nuevo fichero cuyo contenido son las contraseñas más comunes que la mayoría de las personas usan actualmente.

```
mallet@mallet:~$ cat claves.txt
admin
1234
qwerty
```

Figura 5. Fichero de prueba con claves

Para esta comprobación se va a usar el algoritmo de *hashing* SHA-512 para evitar el problema de las colisiones.

En primer lugar, se va a generar el resumen MD5 y SHA512 para el mismo fichero como se muestra en la siguiente imagen.

```
mallet@mallet:~$ ls
claves.txt Desktop Downloads Exploits testttttt.pdf
mallet@mallet:~$
mallet@mallet:~$ sha512sum claves.txt
dccbae20f78036609c6d9822ce9e5a4ef4bcd25457d9a12cb42af99f9f3f6971851cff01c77e4962f88852403baf763f44b299ed0
39f17d9f8c775f95f742fcc claves.txt
mallet@mallet:~$
mallet@mallet:~$ md5sum claves.txt
9d7e1ef72c57c89e4ad19f0c02a7ca7c claves.txt
```

Figura 6. Generar resumen MD5 y SHA512 para el fichero de las claves

Como se ha mencionado anteriormente, el principio de integridad consiste en modificar el fichero que contiene las claves. Se va a añadir un retorno de carro en la última contraseña. Aparentemente el contenido es el mismo, pero en realidad hay un carácter ASCII más.

```
mallet@mallet:~$ nano claves.txt
mallet@mallet:~$
mallet@mallet:~$ md5sum claves.txt
dc261b5408d0bdd78257e238d3eb0643 claves.txt
```

Figura 7. Repetición del cálculo del hash después de realizar un cambio

En la *Figura 7* se puede observar como el resumen ha cambiado de una ejecución a otra. Si se conoce el MD5 original del fichero, se vuelve a calcular el resumen y los resultados del *hash* son diferentes se puede afirmar que el contenido del fichero ha sido modificado.

4. Herramienta hashid

Esta herramienta se usa para tratar de detectar cual ha sido la función *hash* utilizada para crear el resumen.

Sin embargo como la máquina de *mallet* no tiene la aplicación instalada, este ejemplo se va a realizar sobre una máquina Kali Linux particular.

En primer lugar, se calculará el resumen MD5 para un mensaje en texto claro. Esto es así porque la herramienta *hashid* necesita como argumento de entrada un resumen previamente calculado. Estos pasos se podrán comprobar en la siguiente imagen.

```
(thegoodhacker@kali)-[~]
$ echo hola | md5sum
916f4c31aaa35d6b867dae9a7f54270d -

(thegoodhacker@kali)-[~]
$ hashid 916f4c31aaa35d6b867dae9a7f54270d
Analyzing '916f4c31aaa35d6b867dae9a7f54270d'
[+] MD2
[+] MD5
[+] MD4
[+] Double MD5
[+] LM
[+] RIPEMD-128
[+] Haval-128
[+] Tiger-128
[+] Skein-256(128)
[+] Skein-512(128)
[+] Lotus Notes/Domino 5
[+] Skype
[+] Snefru-128
[+] NTLM
[+] Domain Cached Credentials
[+] Domain Cached Credentials 2
[+] DNSSEC(NSEC3)
[+] RAdmin v2.x

(thegoodhacker@kali)-[~]
$
```

Figura 8. Uso de la aplicación *hashid*

El resultado que proporciona esta herramienta es posibles algoritmos de *hash* que podrían haber generado el resumen que se le ha pasado como argumento. Cuando se dice posibles es porque existen más de un algoritmo de *hash* que ha utilizado el mismo alfabeto y la misma longitud de salida.

5. Modificar propiedades de un fichero y volver a obtener el hashcode

Lo primero que se va a hacer es utilizar el comando *touch* para crear un nuevo fichero vacío que se utilizará más adelante como ejemplo en esta sección. Una vez creado se va a comprobar qué permisos le ha asignado el sistema operativo en el momento de su creación.

```
mallet@mallet:~$ touch prueba
mallet@mallet:~$
mallet@mallet:~$ ls -l prueba
-rw-r--r-- 1 mallet mallet 0 2022-10-29 18:13 prueba
```

Figura 9. Creación de un fichero y visualización de sus permisos

A continuación, se van a calcular los resúmenes *md5sum* y *sha1sum* para las propiedades de ese fichero como se puede observar en la siguiente imagen.

```
mallet@mallet:~$ ls -l prueba | md5sum
d0712605e27982fc5f2f2a16654a5ae5 -
mallet@mallet:~$
mallet@mallet:~$ ls -l prueba | sha1sum
9e2629857d6915c6e4ff5f195978b8edff64e953 -
```

Figura 10. Cálculo del hash de las propiedades del fichero

En este punto se va a modificar las propiedades del fichero para que el usuario *mallet* tenga permisos de ejecución sobre el fichero creado anteriormente.

```
mallet@mallet:~$ chmod 700 prueba
mallet@mallet:~$
mallet@mallet:~$ ls -l prueba
-rwx----- 1 mallet mallet 0 2022-10-29 18:13 prueba
```

Figura 11. Nuevos permisos del fichero

Finalmente, se calculan otra vez los *hashes* asociados a las nuevas propiedades del fichero.

```
mallet@mallet:~$ ls -l prueba | md5sum
cbacfafff4fae1d6633025ecadcfd334 -
mallet@mallet:~$
mallet@mallet:~$ ls -l prueba | sha1sum
e8ec4327be853b9eb222b116e147765b4e4c45cf -
```

Figura 12. Cálculo de los nuevos hashes

En conclusión, se puede afirmar que cualquier cambio que sufran los datos de entrada, es decir sobre los que se aplican las funciones de *hash*, produciría un resultado de salida muy diferente. Esto se conoce como el efecto cascada. Ese cambio en el resultado del *hash* podría indicar que el fichero ha sido corrompido.

6. Herramienta John The Ripper

En el mundo informático cuando se tiene una máquina comprometida, el objetivo más buscado es la persistencia. Para poder lograr eso hay que saber dónde se guarda la información sobre los usuarios y sus contraseñas. Por defecto, Linux almacena dichos datos en el fichero llamado */etc/passwd*.

Por motivos de seguridad, en ese fichero los *passwords* no aparecen visibles. Esto se debe a que las contraseñas están almacenadas en el fichero */etc/shadow* de forma cifrada. Para poder ver el contenido de este fichero hay que tener privilegios de administrador.

En una situación real, si se está realizando un test de intrusión y se consigue el contenido del fichero *shadow* habría que intentar obtener las contraseñas en texto claro. Para poder obtener el texto en claro de una contraseña es necesario saber la función *hash* que lo ha generado.

```
hplip:!:14728:0:99999:7:::
gdm:!:14728:0:99999:7:::
mallet:$6$59dzXltF$m4G9nbTcb/UP1R5jY5V5uQK43hulvMm7nTY40oQwwIFcQ2b64gV0bXegmQjbDj3Ah3Ig
NkNgcTpz9osP94aCG0:14792:0:99999:7:::
```

Figura 13. Contenido fichero */etc/shadow* de *mallet*

Como se puede observar en la *Figura 13* la contraseña cifrada del usuario *mallet* empieza por \$6\$. Eso es el identificador de la función de *hash* que se ha utilizado para obtener el resumen de su contraseña. Haciendo una búsqueda se puede saber que la función *hash* que se identifica con \$6\$ es SHA-512 en UNIX.

Como pentester si se quiere realizar un ataque de fuerza bruta para poder obtener el texto en claro vinculado al *hash* de la contraseña del usuario *mallet*, que se ha obtenido con SHA-512, se utiliza la herramienta *John The Ripper*.

En el caso de que se haya comprometido un sistema con conexión a Internet y se haya descargado esta aplicación en la misma máquina, se podrían descifrar todas las contraseñas de los usuarios del sistema. En la siguiente imagen se muestra un ejemplo del uso de esta herramienta con los datos de los usuarios que proporciona el enunciado.

```
mallet@mallet:~$ sudo john /etc/shadow
Loaded 3 password hashes with 3 different salts (generic crypt(3) [?/32])
mallet          (mallet)
123456          (toor)
admin           (root)
guesses: 3  time: 0:00:01:19 100.00% (2) (ETA: Sat Oct 29 18:43:55 2022)  c/s: 91.97
```

Figura 14. Uso de la herramienta *John The Ripper*

Se puede dar el caso en el que en la *Figura 14* no aparezca la contraseña del usuario *root*. Esto se debe a que si se consultara el fichero donde están almacenadas las claves se podría comprobar como el usuario *root* no tendría un *password* asignado como se muestra a continuación.

```
mallet@mallet:~$ sudo cat /etc/shadow | grep -i root
root:!:14792:0:99999:7:::
```

Figura 15. Buscar usuario *root* en el fichero de las contraseñas

En conclusión, uno de los usos útiles de la herramienta *John The Ripper* es para que los administradores de sistemas comprueben cuales son las políticas de las contraseñas de los usuarios.

7. Programa buildhash para calcular los hashcode del fichero y sus propiedades

En esta sección se pide crear un programa *buildhash* con el objetivo de obtener para cada uno de los ficheros, cuyos nombres aparecen en un fichero de entrada auxiliar, dos *hashcodes*: uno asociado al propio fichero y otro a sus propiedades.

El contenido del fichero que se le pasa al *shell script* se muestra a continuación. Contiene un nombre de un fichero por cada línea.

```
mallet@mallet:~$ cat ficheros.txt
prueba
prueba2
mallet@mallet:~$
mallet@mallet:~$ ls
buildhash_aux.md5  checkhash.sh  Downloads  iptables-ejer7  prueba
buildhash.md5      claves.txt    Exploits    iptables-ejer8  prueba2
buildhash.sh       Desktop      ficheros.txt  iptables.fw     testttttt.pdf
```

Figura 16. Contenido del fichero de entrada

En la *Figura 17* se presenta el código del *shell script* encargado de hacer la función descrita anteriormente.

```
#!/bin/bash

while IFS= read -r linea          # Leemos el contenido del fichero con los nombres linea a linea
do
    md5sum $linea >> buildhash.md5 # Calculamos el hashcode del fichero y el resultado lo almacenamos en un fichero
    ls -l $linea | md5sum >> buildhash.md5 # Calculamos el hashcode de las propiedades para almacenarlo en el mismo fichero
    echo -e "\n" >> buildhash.md5 # Imprimimos un salto de linea para diferenciar el resultado de cada fichero
done < /home/mallet/ficheros.txt # Le pasamos el fichero que contiene el nombre de los ficheros para calcular su hashcode
```

Figura 17. Código del programa buildhash

Finalmente, se muestra el resultado por pantalla cuando el fichero de entrada tiene como contenido el nombre de dos ficheros de prueba.

```
mallet@mallet:~$ ./buildhash.sh
mallet@mallet:~$
mallet@mallet:~$ cat buildhash.txt
1e04bb3f9f396d3b71d93d326ebfc42d  prueba
b05400f088e2ad8807ebc2afa77a4d75  -

23325878538f0347ff8307b26d996cc0  prueba2
e7bab38ec036a74f041c833278700663  -
```

Figura 18. Resultado por pantalla del programa buildhash

8. Programa checkhash para comprobar que los ficheros no se han modificado

En esta sección se pedía construir un programa *checkhash* cuya entrada sería el fichero generado en la sección anterior. El propósito de este *shell script* es comprobar si se han producido cambios tanto en los ficheros como en sus propiedades. Al finalizar la comprobación, generará una salida indicando si se han producido cambios o no.

En la *Figura19* se presenta el código del *shell script* encargado de hacer la función descrita anteriormente.

```
#! /bin/bash

while IFS= read -r linea # Leemos el contenido del fichero con los nombres linea a linea
do
    md5sum $linea >> buildhash_aux.md5 # Calculamos el hashcode del fichero y el resultado lo almacenamos en otro fichero
    ls -l $linea | md5sum >> buildhash_aux.md5 # Calculamos el hashcode de sus propiedades para almacenarlo en el mismo fichero
    echo -e "\n" >> buildhash_aux.md5 # Imprimimos un salto de linea para diferenciar
done < /home/mallet/ficheros.txt # Le pasamos el fichero que contiene los nombres de los ficheros para volver a calcular
                                # su hashcode

if cmp -s "buildhash.md5" "buildhash_aux.md5" # Comprobamos si el contenido de ambos ficheros es igual
then
    echo "Todo OK --> Los ficheros no se han modificado" # Si es igual imprime este mensaje
    echo -e "\n"
else
    echo "ERROR --> Los ficheros se han modificado" # Si son diferentes, mostrara el contenido de ambos ficheros más este mensaje
    echo -e "\n"
fi
```

Figura 19. Código del programa checkbash

A continuación, se muestra el resultado por pantalla en el caso de que no se haya producido ningún cambio en los ficheros.

```
mallet@mallet:~$ ./buildhash.sh
mallet@mallet:~$
mallet@mallet:~$ ./checkhash.sh
Todo OK --> Los ficheros no se han modificado
```

Figura 20. Resultado todo correcto

Sin embargo, si se ha producido un cambio en cualquiera de los dos ficheros, ya sea en su contenido o en sus propiedades, el programa informará del error en la comprobación como se muestra a continuación.

```
mallet@mallet:~$ chmod 600 prueba2
mallet@mallet:~$
mallet@mallet:~$ ./checkhash.sh
ERROR --> Los ficheros se han modificado
```

Figura 21. Resultado cuando ha habido un cambio

PARTE 2

1. Dibuja el diagrama de red lo más detallado posible

El router es un dispositivo clave para una red informática compleja, ya que tiene múltiples funciones como, por ejemplo, encaminar los paquetes en la red permitiendo así poder interconectar equipos que pertenecen a diferentes redes con diferentes direcciones IP.

Dentro de una empresa es imprescindible tener un esquema de la red completo, por muy pequeña que sea la empresa, para poder saber exactamente cuántos dispositivos están conectados, que direcciones IP tienen asignadas, etc. De esta forma se puede tener un control total sobre la red.

Un router como mínimo debería tener dos interfaces de red. Las direcciones IP de dichas interfaces deberían ser siempre fijas, porque si esa dirección IP cambia dicho router dejaría de prestar servicio.

La puerta de enlace, también llamada *gateway*, es una máquina, que está en el mismo segmento de red, que permite conectarse a máquinas que pertenecen a otra red.

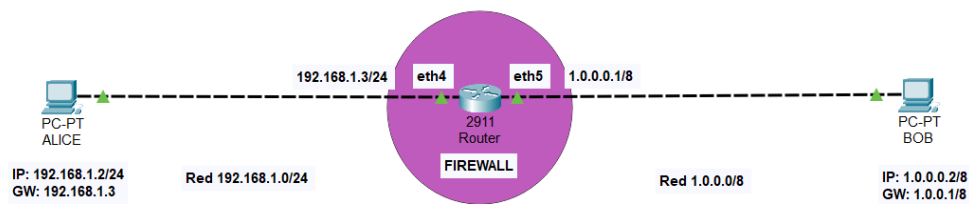


Figura 22. Diagrama de la red

Un *firewall* sirve para aplicar una serie de políticas con el objetivo de decidir que paquetes están permitidos a circular por la red. ¿Dónde se colocaría un firewall en una red? Normalmente se pondría en el router, como se puede ver en la imagen, ya que por él pasa todo el tráfico de la red.

2. Configurar la máquina de *mallet* para que actúe como router

La máquina de *mallet* inicialmente solo tiene configurada una interfaz de red. Sin embargo, el propósito de esta sección es configurarla para que pueda actuar como un router.

Una red interna, en VirtualBox, es una red que no tiene el servicio de enrutado ni el servicio de DHCP ni DNS. Es decir las direcciones IP del rango 192.168.1.1-3 están libres.

Si se configurara la dirección IP 192.168.1.3 en una red NAT habría problemas porque dicha dirección pertenece al servicio DHCP de VirtualBox. No obstante, utilizando una red interna no habría ningún tipo de problemas de solapamiento de direcciones IP con los servicios de VirtualBox.

En primer lugar, para poder hacer cualquier cambio en la configuración de la red hay que asegurarse de que la máquina de *mallet* esté apagada.

Por un lado, se va a configurar el adaptador de red 1, con el alias eth4, para que esté dentro de una red interna cuyo nombre es 192.168.1.0.



Figura 23. Configuración de eth4 en mallet

Por otro lado, la tarjeta de red, cuyo alias es eth5 que equivale al adaptador de red 2, pertenecerá a la red interna con el nombre 1.0.0.0 que es la dirección de red donde va a estar ese adaptador.

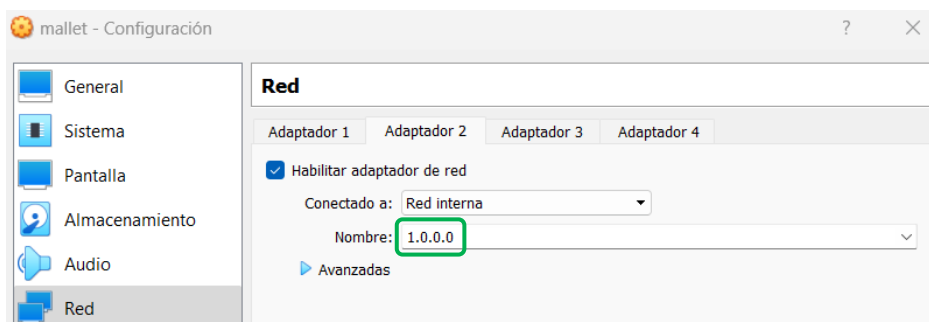


Figura 24. Configuración de eth5 en mallet

Finalmente, el resultado que se obtiene gracias a esta configuración es la máquina virtual de *mallet* con dos interfaces de red. El siguiente paso sería configurar las direcciones IP de dichas interfaces con las direcciones que aparecen en el diagrama de red de la Figura 22.

Como se va a modificar un fichero crítico del sistema es necesario ser un super usuario o, en su defecto, poder tener privilegios de super usuario. Antes de empezar se va a comprobar a que grupo o grupos pertenece el usuario de *mallet* como se muestra en la siguiente imagen.

```
mallet@mallet:~$ groups
mallet adm dialout cdrom plugdev lpadmin admin sambashare
mallet@mallet:~$ id
uid=1000(mallet) gid=1000(mallet) groups=4(adm),20(dialout),24(cdrom),46(plugdev),105(lpadmin),119(admin),122(sambashare),1000(mallet)
```

Figura 25. Grupos a los que pertenece el usuario de *mallet*

Para poder configurar las interfaces de red de una máquina es necesario saber cuál es el fichero del sistema que se encarga de esta función.

```
GNU nano 2.2.2                               File: /etc/network/interfaces
auto lo eth4 eth5
iface lo inet loopback

iface eth4 inet static
    address 192.168.1.3
    netmask 255.255.255.0

iface eth5 inet static
    address 1.0.0.1
    netmask 255.0.0.0
```

Figura 26. Modificaciones en el fichero de configuración de las interfaces de red

Después de la palabra *auto* aparecen los alias de las interfaces de red que se quieren activar en el momento en que se encienda la máquina. Se configuran ambas interfaces con una dirección IP estática para que no se pierda su servicio. Primero se configura la dirección y posteriormente su máscara de red. Finalmente, se guardan los cambios que se han efectuado en este fichero.

Para que dichos cambios en la configuración tengan efecto habría que reiniciar los servicios de red con privilegios como se muestra en la siguiente imagen.

```
mallet@mallet:~$ sudo /etc/init.d/networking restart
[sudo] password for mallet:
* Reconfiguring network interfaces...
ssh stop/waiting
ssh start/running, process 1638
ssh stop/waiting
ssh start/running, process 1696
```

Figura 27. Reiniciando los servicios de red de *mallet*

Finalmente, gracias al comando *ifconfig* se hace una comprobación del estado de las interfaces de red que tiene activas en este momento la máquina de *mallet*.

```
mallet@mallet:~$ ifconfig
eth4      Link encap:Ethernet  HWaddr 08:00:27:bf:6a:bf
          inet addr:192.168.1.3  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:febf:6abf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:41 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:5967 (5.9 KB)

eth5      Link encap:Ethernet  HWaddr 08:00:27:05:34:ee
          inet addr:1.0.0.1  Bcast:1.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::a00:27ff:fe05:34ee/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:49 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:7510 (7.5 KB)
```

Figura 28. Estado de las interfaces de red de *mallet*

Para finalizar esta sección y que el router pueda funcionar correctamente se debería activar el servicio de *routing*. Es decir activar el bit de enrutamiento para que los paquetes puedan atravesar la máquina de *mallet* y llegar a su destino sin ningún problema.

Actualmente el contenido del fichero de esa configuración es un cero, lo que significa que el *forwarding* está deshabilitado en esa máquina. Sin embargo, para que los paquetes puedan entrar y salir de la máquina de *mallet* habría que cambiar el cero por un uno con el objetivo de activar este servicio.

```
root@mallet:/home/mallet# cat /proc/sys/net/ipv4/ip_forward
0
root@mallet:/home/mallet#
root@mallet:/home/mallet# echo 1 > /proc/sys/net/ipv4/ip_forward
root@mallet:/home/mallet#
root@mallet:/home/mallet# cat /proc/sys/net/ipv4/ip_forward
1
```

Figura 29. Configuración del bit de enrutamiento

Desafortunadamente, esta configuración es temporal. Si la máquina se reinicia por cualquier motivo la configuración del bit de enrutamiento se perderá.

3. Configuración de la máquina de *bob*

La máquina de *bob*, a diferencia de la de *alice*, tiene que pertenecer a la red interna 1.0.0.0 como se indica en la siguiente imagen.

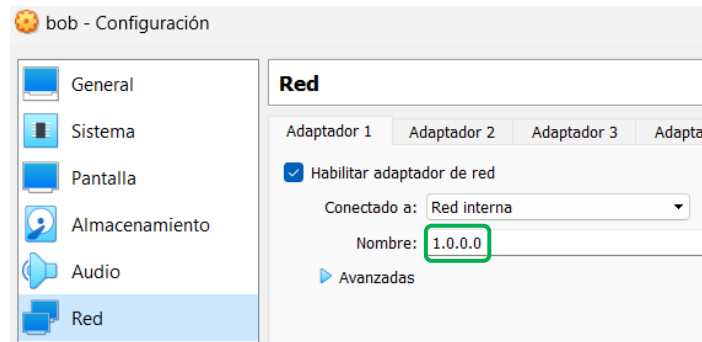


Figura 30. Configuración del adaptador de red de bob

Para esta máquina se va a asignar una dirección IP estática de un modo diferente a lo que se ha explicado anteriormente. Se va a hacer una configuración de la interfaz `eth0` de manera temporal como se muestra a continuación.

```
bob:~# ifconfig eth0 1.0.0.2
bob:~#
bob:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:ce:a3:62
          inet addr:1.0.0.2  Bcast:1.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::a00:27ff:fece:a362/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:468 (468.0 B)
          Base address:0xd010  Memory:f0000000-f0020000
```

Figura 31. Configuración temporal de la dirección IP de bob

Para que la máquina de *bob* se pueda comunicar con la de *alice* falta configurar un último parámetro. A continuación, se va a comprobar si existe una puerta de enlace para *bob*.

```
bob:~# route -n
Kernel IP routing table
Destination    Gateway         Genmask         Flags Metric Ref    Use Iface
1.0.0.0        0.0.0.0        255.0.0.0      U        0      0        0 eth0
```

Figura 32. Comprobar si existe un gateway para bob

Finalmente, se va a configurar manualmente el *gateway* para que se pueda comunicar con *alice*.

```

bob:~# route add -net 192.168.1.0 netmask 255.255.255.0 gw 1.0.0.1
bob:~#
bob:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     1.0.0.1        255.255.255.0   UG    0      0      0 eth0
1.0.0.0         0.0.0.0        255.0.0.0       U     0      0      0 eth0

```

Figura 33. Configuración del gateway de bob

El significado de esta configuración es que para poder llegar a la red donde se encuentra la máquina de *alice*, los paquetes tendrán que entrar al router por la interfaz de red 1.0.0.1 con alias eth5.

4. Técnica de *pivoting* de post-explotación

Cuando se tiene comprometida una máquina es probable que se descubra que tiene acceso a otras máquinas o incluso redes, a las cuales un pentester no podría tener acceso directamente.

Esto quiere decir que se tendría que usar la máquina comprometida como “máquina de salto” para poder tener acceso a redes internas o otros quipos. Este proceso se puede repetir de forma recursiva.

Sin entrar mucho en detalle para no alargar el informe, la metodología completa sería la siguiente:

- *Pivoting*
- Enumeración post-explotación de redes y máquinas
- Explotación
- Volver al paso uno

5. Configuración de la máquina de *alice*

La máquina de *alice* va a pertenecer a la red interna 192.168.1.0, que es la misma que la de la interfaz de red *eth4* de la máquina de *mallet*. En la configuración de red de *alice* debería aparecer disponible dicha red interna ya que previamente se había configurado correctamente en la máquina de *mallet*.

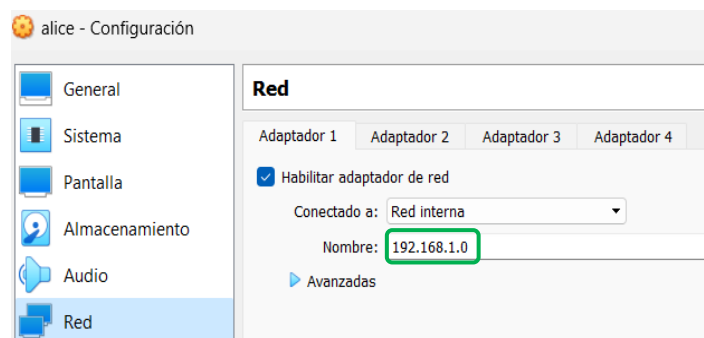


Figura 34. Configurar el adaptador de red de alice

Con la siguiente configuración se le está diciendo al sistema operativo de la máquina de *alice* que la interfaz de red *eth3* se active en el momento que la máquina arranque. Posteriormente, se le va a asignar una dirección IPv4 y además se le va a proporcionar la dirección de la puerta de enlace para que se pueda comunicar con la máquina de *bob*.

```
GNU nano 2.2.2                               File: /etc/network/interfaces
auto lo
iface lo inet loopback

auto eth3
iface eth3 inet static
    address 192.168.1.2
    netmask 255.255.255.0
    gateway 192.168.1.3
```

Figura 35. Configuración de la interfaz de red de *alice*

El siguiente paso sería reiniciar los servicios de red para que la configuración anterior tenga efecto en el sistema.

```
alice@alice:~$ sudo /etc/init.d/networking restart
* Reconfiguring network interfaces...
ssh stop/waiting
ssh start/running, process 2327
```

Figura 36. Reiniciando los servicios de red de *alice*

Finalmente, se hace una comprobación de que tanto la dirección IP como la máscara de red se han configurado correctamente para evitar problemas de conexión en un futuro.

```
alice@alice:~$ ifconfig
eth3      Link encap:Ethernet  HWaddr 08:00:27:06:ae:f6
          inet addr:192.168.1.2  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe06:aef6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:125 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:7663 (7.6 KB)
```

Figura 37. Comprobación de la dirección IP de *alice*

6. Pruebas a nivel de red para comprobar que las máquinas se pueden comunicar

En primer lugar, se van a realizar pruebas en la máquina de *mallet* ya que es la primera que se ha configurado. Esto se debe a que cuando se monta una red es mejor ir haciendo pequeñas pruebas cada poco tiempo que hacerlas todas cuando la red está montada, ya que eso podría dificultar tanto la detección de posibles fallos como la identificación de la raíz del problema

Como se muestra en la *Figura 38*, se ha realizado un *ping* desde la máquina de *mallet* a sus dos interfaces de red con el objetivo de comprobar si están activas y bien configuradas. En los siguientes ejemplos se puede observar como el valor del TTL es 64 debido a que es una máquina con un sistema operativo Linux.

```
mallet@mallet:~$ ping 192.168.1.3 -c 3 ← eth4
PING 192.168.1.3 (192.168.1.3) 56(84) bytes of data.
64 bytes from 192.168.1.3: icmp_seq=1 ttl=64 time=0.021 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=64 time=0.022 ms
64 bytes from 192.168.1.3: icmp_seq=3 ttl=64 time=0.017 ms

--- 192.168.1.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.017/0.020/0.022/0.002 ms
mallet@mallet:~$
mallet@mallet:~$
mallet@mallet:~$ ping 1.0.0.1 -c 3 ← eth5
PING 1.0.0.1 (1.0.0.1) 56(84) bytes of data.
64 bytes from 1.0.0.1: icmp_seq=1 ttl=64 time=0.021 ms
64 bytes from 1.0.0.1: icmp_seq=2 ttl=64 time=0.055 ms
64 bytes from 1.0.0.1: icmp_seq=3 ttl=64 time=0.018 ms

--- 1.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.018/0.031/0.055/0.017 ms
```

Figura 38. Pruebas a nivel de red en la máquina de mallet

A continuación, se harán pruebas a nivel de red desde la máquina de *alice*. En este momento se puede hacer un *ping* a ambas interfaces del router. Esto se debe a que todavía no se ha utilizado el enrutamiento ya que el paquete no ha salido de la máquina de *mallet*. En este punto, como no se está aplicando el *routing* da igual el estado del bit de enrutamiento.

```
alice@alice:~$ ping 192.168.1.3 -c 3 ← eth4
PING 192.168.1.3 (192.168.1.3) 56(84) bytes of data.
64 bytes from 192.168.1.3: icmp_seq=1 ttl=64 time=0.745 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=64 time=1.15 ms
64 bytes from 192.168.1.3: icmp_seq=3 ttl=64 time=1.06 ms

--- 192.168.1.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.745/0.989/1.158/0.176 ms
alice@alice:~$
alice@alice:~$
alice@alice:~$ ping 1.0.0.1 -c 3 ← eth5
PING 1.0.0.1 (1.0.0.1) 56(84) bytes of data.
64 bytes from 1.0.0.1: icmp_seq=1 ttl=64 time=0.762 ms
64 bytes from 1.0.0.1: icmp_seq=2 ttl=64 time=1.50 ms
64 bytes from 1.0.0.1: icmp_seq=3 ttl=64 time=1.25 ms

--- 1.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.762/1.172/1.502/0.310 ms
```

Figura 39. Pruebas a nivel de red en la máquina de alice

Finalmente, se va a comprobar si desde la máquina de *bob* se puede conectar a las interfaces de red de la máquina de *mallet* de la misma manera que se ha hecho en el ejemplo anterior.

```

bob:~# ping 192.168.1.3 -c 3 ← eth4
PING 192.168.1.3 (192.168.1.3) 56(84) bytes of data.
64 bytes from 192.168.1.3: icmp_seq=1 ttl=64 time=5.47 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=64 time=2.02 ms
64 bytes from 192.168.1.3: icmp_seq=3 ttl=64 time=1.24 ms

--- 192.168.1.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 1.244/2.914/5.476/1.839 ms
bob:~#
bob:~#
bob:~# ping 1.0.0.1 -c 3 ← eth5
PING 1.0.0.1 (1.0.0.1) 56(84) bytes of data.
64 bytes from 1.0.0.1: icmp_seq=1 ttl=64 time=0.586 ms
64 bytes from 1.0.0.1: icmp_seq=2 ttl=64 time=0.980 ms
64 bytes from 1.0.0.1: icmp_seq=3 ttl=64 time=1.91 ms

--- 1.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.586/1.160/1.915/0.558 ms

```

Figura 40. Pruebas a nivel de red en la máquina de bob

Una vez se ha comprobado que no existen fallos de conectividad desde ambas máquinas al router, es momento de comprobar si la máquina de *alice* se puede comunicar con *bob* que era realmente el objetivo de esta parte.

```

bob:~# ping 192.168.1.2 -c 3 ← alice
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=63 time=8.70 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=63 time=2.85 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=63 time=2.97 ms

--- 192.168.1.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 2.854/4.842/8.703/2.730 ms

```

Figura 41. Ping entre la máquina de bob y la de alice

En este último ejemplo se puede comprobar como el valor del TTL no es 64 como en los casos anteriores, sino que su valor es 63. Esto se debe a que el paquete ha tenido que atravesar el router para poder llegar hasta *alice*, lo que provoca un decremento en una unidad del valor del TTL.

7. Diferencias entre las reglas de tipo INPUT, OUTPUT y FORWARD en *iptables*

Iptables es una herramienta avanzada de filtrado de paquetes en Linux que se encarga de actuar como un firewall o cortafuegos en la red. Su principal función es analizar cada uno de los paquetes del tráfico de red que entra en una máquina y decidir, en función de un conjunto de reglas, qué hacer con ese paquete.

Esta aplicación utiliza tres reglas de filtrado para indicar qué paquetes serán aceptados, cuáles serán rechazados o cuáles serán omitidos:

- **INPUT:** Hace referencia a los paquetes que solo entran al router. Ejemplo: cuando desde *alice* se hace un *ping* a la interfaz de red *eth4* o *eth5*. El paquete ICMP entra al router pero no sale.
- **OUTPUT:** Hace referencia a los paquetes que se generan dentro del router y salen hacia al exterior de la red. Ejemplo: desde *mallet* se hace un *ping* a la máquina de *alice*. El paquete se genera dentro de *mallet* y sale hacia *alice*.
- **FORDWARD:** Referencia aquellos paquetes que entran y salen del router. Es decir, el tráfico que el router reenvía a otros equipos. Ejemplo: un *ping* desde *alice* a *bob*. En este caso habría un decremento en una unidad del valor TTL, el tiempo de vida de los paquetes. Esto se repetirá cada vez que haya un salto en un router.

El orden en el que se configuran las reglas del firewall es determinante. Normalmente cuando hay que decidir que se hace con un paquete se va comparando con cada regla del firewall hasta que se encuentra una que le afecta, y se hace lo que dicte esta regla. Después de eso no se mirarán más reglas para ese paquete. ¿Cuál es el peligro? Si se ponen reglas muy permisivas entre las primeras del firewall, puede que las siguientes no se apliquen y el cortafuegos no funcione como se espera.

8. Configurar el firewall para que solo se puedan realizar peticiones de tipo ICMP

El primer paso siempre es comprobar que reglas existen actualmente en el sistema para saber cuáles habría que añadir o, en su defecto, modificar.

```
root@mallet:~# iptables -nL
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Figura 42. Reglas iniciales del firewall

Como se puede comprobar en la *Figura 42* en este momento se permite todo tipo de tráfico entrante, saliente y de reenvío. Esto en una situación real sería muy peligroso ya que la red estaría totalmente desprotegida ante ciberataques.

En primer lugar, se configuran tres reglas para bloquear todo tipo de tráfico INPUT, OUTPUT o FORWARD. Solo se permitirá pasar por el firewall aquellos paquetes que explícitamente tengan permiso. Por defecto, si no se indica lo contrario con la opción *-t*, se trabaja siempre con la tabla *filter*.

```
root@mallet:~# iptables -P INPUT DROP
root@mallet:~# iptables -P OUTPUT DROP
root@mallet:~# iptables -P FORWARD DROP
```

Figura 43. Bloquear todo tipo de tráfico

El siguiente paso es configurar primero cada regla de la cadena INPUT para que, tanto la máquina de *alice* como la de *bob*, puedan hacer un *ping* a las interfaces de red de la máquina de *mallet*.

```
alice@alice:~$ ping 192.168.1.3 -c 2 → eth4
PING 192.168.1.3 (192.168.1.3) 56(84) bytes of data.
64 bytes from 192.168.1.3: icmp_seq=1 ttl=64 time=0.688 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=64 time=1.54 ms

--- 192.168.1.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.688/1.115/1.542/0.427 ms
alice@alice:~$
alice@alice:~$
alice@alice:~$ ping 1.0.0.1 -c 2 → eth5
PING 1.0.0.1 (1.0.0.1) 56(84) bytes of data.
64 bytes from 1.0.0.1: icmp_seq=1 ttl=64 time=0.702 ms
64 bytes from 1.0.0.1: icmp_seq=2 ttl=64 time=0.742 ms

--- 1.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.702/0.722/0.742/0.020 ms
```

Figura 44. Comprobación de ping de la cadena INPUT con alice hacia *mallet*

```
bob:~# ping 192.168.1.3 -c 2 → eth4
PING 192.168.1.3 (192.168.1.3) 56(84) bytes of data.
64 bytes from 192.168.1.3: icmp_seq=1 ttl=64 time=0.811 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=64 time=1.81 ms

--- 192.168.1.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.811/1.312/1.813/0.501 ms
bob:~#
bob:~#
bob:~# ping 1.0.0.1 -c 2 → eth5
PING 1.0.0.1 (1.0.0.1) 56(84) bytes of data.
64 bytes from 1.0.0.1: icmp_seq=1 ttl=64 time=0.637 ms
64 bytes from 1.0.0.1: icmp_seq=2 ttl=64 time=1.08 ms

--- 1.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.637/0.860/1.084/0.225 ms
```

Figura 45. Comprobación de ping de la cadena INPUT con bob hacia *mallet*

De la misma manera, se configuró las reglas de la cadena FORWARD para que las máquinas de *alice* y *bob* puedan comunicarse entre ellas mediante un *ping*.

```
bob:~# ping 192.168.1.2 -c 2 → alice
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=63 time=6.90 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=63 time=2.45 ms

--- 192.168.1.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1005ms
rtt min/avg/max/mdev = 2.450/4.676/6.903/2.227 ms
```

Figura 46. Comprobación de ping de la cadena FORWARD con bob hacia alice

```
alice@alice:~$ ping 1.0.0.2 -c 2 → bob
PING 1.0.0.2 (1.0.0.2) 56(84) bytes of data.
64 bytes from 1.0.0.2: icmp_seq=1 ttl=63 time=0.947 ms
64 bytes from 1.0.0.2: icmp_seq=2 ttl=63 time=0.926 ms

--- 1.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.926/0.936/0.947/0.032 ms
```

Figura 47. Comprobación de ping de la cadena FORWARD con alice hacia bob

Finalmente, se configuró varias reglas en la cadena OUTPUT para que las interfaces de red de la máquina de *mallet* pudieran hacer un *ping* a las máquinas de *bob* y *alice* respectivamente. Para demostrar que esto funciona correctamente se va a utilizar la opción *-I*, que ofrece la herramienta *ping*, para poder indicar el nombre o dirección IP del origen del *ping*.

```
mallet@mallet:~$ ping 192.168.1.2 -I eth4 -c 2 → alice
PING 192.168.1.2 (192.168.1.2) from 192.168.1.3 eth4: 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=0.744 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=1.02 ms

--- 192.168.1.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.744/0.886/1.029/0.145 ms
mallet@mallet:~$
mallet@mallet:~$
mallet@mallet:~$ ping 1.0.0.2 -I eth5 -c 2 → bob
PING 1.0.0.2 (1.0.0.2) from 1.0.0.1 eth5: 56(84) bytes of data.
64 bytes from 1.0.0.2: icmp_seq=1 ttl=64 time=4.86 ms
64 bytes from 1.0.0.2: icmp_seq=2 ttl=64 time=0.889 ms

--- 1.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.889/2.878/4.868/1.990 ms
```

Figura 48. Comprobación de ping de la cadena OUTPUT con mallet

```

mallet@mallet:~$ ping 192.168.1.2 -I 1.0.0.1 -c 2 ➡ alice
PING 192.168.1.2 (192.168.1.2) from 1.0.0.1 : 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=64 time=0.739 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.616 ms

--- 192.168.1.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.616/0.677/0.739/0.066 ms
mallet@mallet:~$
mallet@mallet:~$
mallet@mallet:~$ ping 1.0.0.2 -I 192.168.1.3 -c 2 ➡ bob
PING 1.0.0.2 (1.0.0.2) from 192.168.1.3 : 56(84) bytes of data.
64 bytes from 1.0.0.2: icmp_seq=1 ttl=64 time=0.479 ms
64 bytes from 1.0.0.2: icmp_seq=2 ttl=64 time=1.08 ms

--- 1.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.479/0.779/1.080/0.301 ms

```

Figura 49. Comprobación de ping de la cadena OUTPUT con mallet

Sin embargo, cualquier paquete que venga de otra red que no sea la interna será descartado. Por ejemplo, si la máquina de *bob* vuelve a pertenecer a la red NAT GSI no podría hacer un *ping* a *alice*.

```

bob:~# ifconfig eth0 192.168.10.10
bob:~#
bob:~# ping 192.168.1.2 -c 2
connect: Network is unreachable

```

Figura 50. Comprobación de un ping fuera de la red interna

Toda esta configuración que se ha explicado y detallado anteriormente se muestra en la *Figura 51*. Para hacer un uso más detallado de esta herramienta, se ha usado el comando *iptables-restore* para cargar en el núcleo el conjunto de reglas previamente guardadas con *iptables-save*.

```

root@mallet:/home/mallet# iptables-restore < /home/mallet/iptables-ejer7
root@mallet:/home/mallet#
root@mallet:/home/mallet# iptables -nL --line-numbers
Chain INPUT (policy DROP)
num  target      prot opt source                destination
1    ACCEPT      icmp -- 192.168.1.2            192.168.1.3
2    ACCEPT      icmp -- 192.168.1.2            1.0.0.1
3    ACCEPT      icmp -- 1.0.0.2              192.168.1.3
4    ACCEPT      icmp -- 1.0.0.2              1.0.0.1

Chain FORWARD (policy DROP)
num  target      prot opt source                destination
1    ACCEPT      icmp -- 192.168.1.2            1.0.0.2
2    ACCEPT      icmp -- 1.0.0.2              192.168.1.2

Chain OUTPUT (policy DROP)
num  target      prot opt source                destination
1    ACCEPT      icmp -- 192.168.1.3            1.0.0.2
2    ACCEPT      icmp -- 192.168.1.3            192.168.1.2
3    ACCEPT      icmp -- 1.0.0.1              1.0.0.2
4    ACCEPT      icmp -- 1.0.0.1              192.168.1.2

```

Figura 51. Reglas configuradas en el firewall para permitir peticiones ICMP solo desde la red interna

Finalmente, para no perder esta configuración se va a hacer una copia de seguridad gracias a la opción que tiene la herramienta *iptables* como se muestra a continuación.

```
root@mallet:/home/mallet# iptables-save > /home/mallet/iptables-ejer7
```

Figura 52. Hacer copia de seguridad de las reglas

9. Configurar el firewall para que solo una máquina pueda conectarse por ssh con otra

Para poder explicar bien este apartado se han borrado todas las reglas configuradas en la sección anterior con el objetivo de poder visualizar mejor las reglas de este apartado.

```
root@mallet:/home/mallet# iptables -nL --line-numbers
Chain INPUT (policy ACCEPT)
num target prot opt source destination
1 REJECT tcp -- 1.0.0.2 192.168.1.3 tcp dpt:22 reject-with icmp-port-unreachable
2 REJECT tcp -- 1.0.0.2 1.0.0.1 tcp dpt:22 reject-with icmp-port-unreachable
3 REJECT tcp -- 192.168.1.2 1.0.0.1 tcp dpt:22 reject-with icmp-port-unreachable
4 REJECT tcp -- 192.168.1.2 192.168.1.3 tcp dpt:22 reject-with icmp-port-unreachable

Chain FORWARD (policy ACCEPT)
num target prot opt source destination
1 REJECT tcp -- 1.0.0.2 192.168.1.2 tcp dpt:22 reject-with icmp-port-unreachable
2 REJECT tcp -- 192.168.1.2 1.0.0.2 tcp dpt:22 reject-with icmp-port-unreachable

Chain OUTPUT (policy ACCEPT)
num target prot opt source destination
1 ACCEPT tcp -- 192.168.1.3 192.168.1.2 tcp dpt:22
2 REJECT tcp -- 1.0.0.1 192.168.1.2 tcp dpt:22 reject-with icmp-port-unreachable
3 REJECT tcp -- 1.0.0.1 1.0.0.2 tcp dpt:22 reject-with icmp-port-unreachable
4 REJECT tcp -- 192.168.1.3 1.0.0.2 tcp dpt:22 reject-with icmp-port-unreachable
```

Figura 53. Reglas para controlar el acceso por ssh

En este apartado se ha introducido un nuevo concepto en la configuración de un firewall con *iptables*. En el ejemplo anterior solo se había utilizado la política ACCEPT. Sin embargo, en este caso se ha hecho uso de la política REJECT. Su función es descartar el paquete, pero en esta ocasión se manda un error en un paquete ICMP a la máquina que hizo la petición para indicarle que no tiene permiso. En la siguiente imagen se muestran todos los comandos ejecutados para llegar a configurar el firewall como en la imagen anterior.

```
root@mallet:/home/mallet# cat iptables-ejer8
# Generated by iptables-save v1.4.4 on Mon Oct 31 20:46:19 2022
*filter
:INPUT ACCEPT [198:26482]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [223:25308]
-A INPUT -s 1.0.0.2/32 -d 192.168.1.3/32 -p tcp -m tcp --dport 22 -j REJECT --reject-with icmp-port-unreachable
-A INPUT -s 1.0.0.2/32 -d 1.0.0.1/32 -p tcp -m tcp --dport 22 -j REJECT --reject-with icmp-port-unreachable
-A INPUT -s 192.168.1.2/32 -d 1.0.0.1/32 -p tcp -m tcp --dport 22 -j REJECT --reject-with icmp-port-unreachable
-A INPUT -s 192.168.1.2/32 -d 192.168.1.3/32 -p tcp -m tcp --dport 22 -j REJECT --reject-with icmp-port-unreachable
-A FORWARD -s 1.0.0.2/32 -d 192.168.1.2/32 -p tcp -m tcp --dport 22 -j REJECT --reject-with icmp-port-unreachable
-A FORWARD -s 192.168.1.2/32 -d 1.0.0.2/32 -p tcp -m tcp --dport 22 -j REJECT --reject-with icmp-port-unreachable
-A OUTPUT -s 192.168.1.3/32 -d 192.168.1.2/32 -p tcp -m tcp --dport 22 -j ACCEPT
-A OUTPUT -s 1.0.0.1/32 -d 192.168.1.2/32 -p tcp -m tcp --dport 22 -j REJECT --reject-with icmp-port-unreachable
-A OUTPUT -s 1.0.0.1/32 -d 1.0.0.2/32 -p tcp -m tcp --dport 22 -j REJECT --reject-with icmp-port-unreachable
-A OUTPUT -s 192.168.1.3/32 -d 1.0.0.2/32 -p tcp -m tcp --dport 22 -j REJECT --reject-with icmp-port-unreachable
COMMIT
# Completed on Mon Oct 31 20:46:19 2022
```

Figura 54. Comandos ejecutados para configurar el firewall

Para comprobar si las reglas de la cadena INPUT funcionan se va a probar a establecer una conexión ssh a la máquina de *mallet* tanto desde *bob* como desde *alice*.

```
bob:~# sshallet@192.168.1.3
ssh: connect to host 192.168.1.3 port 22: Connection refused
bob:~#
bob:~# sshallet@1.0.0.1
ssh: connect to host 1.0.0.1 port 22: Connection refused
```

Figura 55. Prueba de conexión ssh a la máquina de *mallet* desde *bob*

```
alice@alice:~$ sshallet@192.168.1.3
ssh: connect to host 192.168.1.3 port 22: Connection refused
alice@alice:~$
alice@alice:~$ sshallet@1.0.0.1
ssh: connect to host 1.0.0.1 port 22: Connection refused
```

Figura 56. Prueba de conexión ssh a la máquina de *mallet* desde *alice*

El siguiente paso sería probar a establecer una conexión ssh entre las máquinas de *bob* y *alice* para verificar el correcto funcionamiento de la regla de la cadena FORWARD.

```
bob:~# sshalice@192.168.1.2
ssh: connect to host 192.168.1.2 port 22: Connection refused
```

Figura 57. Prueba de conexión ssh de *bob* a *alice*

```
alice@alice:~$ sshbob@1.0.0.2
ssh: connect to host 1.0.0.2 port 22: Connection refused
```

Figura 58. Prueba de conexión ssh de *alice* a *bob*

Finalmente, se mostrarán pruebas de que las reglas de la cadena OUTPUT funcionan como deberían que es lo que nos pedían en este apartado.

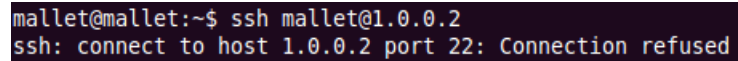
```
mallet@mallet:~$ sshalice@192.168.1.2
alice@192.168.1.2's password:
Linux alice 2.6.32-28-generic #55-Ubuntu SMP Mon Jan 10 21:21:01 UTC 2011 i686 GNU/Linux
Ubuntu 10.04.2 LTS

Welcome to Ubuntu!
 * Documentation:  https://help.ubuntu.com/

New release 'precise' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Mon Oct 31 20:35:10 2022 from mallet
alice@alice:~$
```

Figura 59. Conexión ssh correcta entre *mallet* y *alice*



```
mallet@mallet:~$ ssh mallet@1.0.0.2
ssh: connect to host 1.0.0.2 port 22: Connection refused
```

Figura 60. Prueba de conexión ssh fallida de mallet a bob

10. Conclusiones

Gracias a los contenidos de esta práctica se ha podido entender como funcionan algunas de las herramientas de creación de *hashes*. También, se ha podido usar una potente aplicación que usan frecuentemente los *pentesters* para poder hacer un ataque de fuerza bruta sobre las contraseñas de un sistema y poder conseguir así persistencia en la máquina comprometida.

Por otro lado, se ha aprendido a configurar correctamente una máquina para que actúe como router con un firewall incorporado.

Finalmente, se puede decir que se han podido aplicar técnicas usadas tanto por el *red team* como por el *blue team*.