Daniel Garcia Salinas

HASHING Y CIFRADO

PARTE 1

1. PROPIEDADES DE LAS FUNCIONES HASH POR LAS CUALES SON INTERESANTES DESDE EL PUNTO DE VISTA DE LA SEGURIDAD DE LA INFORMACIÓN
   a) **DETERMINISMO:** Una función hash siempre debe producir la misma salida para la misma entrada. Esto nos asegura que podamos descifrar el mensaje inicial tras haberlo encriptado
   b) **UNIFORME:** Una función hash debe distribuir su salida uniformemente en el rango posible de valores sin ningún sesgo. Esto asegura que al ser todos los valores equiprobables, sea igual de difícil descifrar el mensaje para todas las salidas posibles
   c) **UNIDIRECCIONAL:** En una función hash debe ser sencillo obtener la salida a partir de la entrada pero complicado obtener la entrada a partir de la salida. Esto nos asegura que solo aquél dispositivo confiable que sepa la forma de proceder para obtener la entrada, sea el único que pueda descifrarlo
   d) **RESISTENTE A COLISIONES:** Es complicado o imposible encontrar una entrada que de lugar a 2 salida. Al no ocurrir esto, sabemos que cada salida se asocia a una entrada, sin dar lugar a equivocaciones tras el descifrado.
   e) **EFECTO AVALANCHA:** Un pequeño cambio en la entrada, provoca que la salida sea completamente distinta. Esto ayuda a despistar a los atacantes, ya que si en otros casos se encuentran 2 mensajes similares, podrían descifrar el patrón a partir de estos. Con las funcione hash esto no es posible

Información: https://fastercapital.com/es/tema/propiedades-de-las-buenas-funciones-hash.html

2. INDICA LA UTILIDAD DE LAS FUNCIONES HASH EN EL ÁMBITO DE LA SEGURIDAD DE LA INFORMACIÓN

- Como ya hemos visto, las funciones hash nos permiten tener una comunicación segura en un medio no privado. Esto tiene muchas utilidades:
  - **VERIFICACIÓN DE CONTRASEÑAS**:
    - En vez de enviar la contraseña en texto plano, esta se manda como un mensaje cifrado mediante una función hash. Esta se compara con la contraseña almacenada, que también

está almacenada tras haber sido cifrada con dicha función hash y si son iguales se acepta

- **INTEGRIDAD DE LOS DATOS**:
  - En la transferencia de archivos, se forma un hash junto con los datos originales enviados. Tras recalcular el hash y compararlo con el valor original en destino, podemos ver si se ha modificado el contenido de los datos
- **FIRMAS MALWARE**:
  - Existen bases de datos de "malware", donde cada firma malware es un valor hash generado a partir del código malicioso. Cuando un sistema de seguridad escanea archivos en busca de malware, este calcula los hashes y si son coincidentes, se etiquetan como malware

Información: https://www.bbva.com/es/innovacion/de-la-verificacion-de-contrasena-a-la-firma-electronica-el-secreto-esta-en-el-hash/

3. COMPLETA LA SIGUIENTE TABLA PERTENECIENTE A CARACTERÍSTICAS DE LAS FUNCIONES HASH

FUNCIÓN	Nº CARACTERES EN HEXADECIMAL	Nº DE BITS	¿VULNERABLE?
MD5	32	128	SI
SHA1	40	160	SI
SHA256	64	256	NO
SHA512	128	512	NO

4. CONSTRUCCIÓN DE HASHES CON DISTINTAS HERRAMIENTAS

- PARA CADENAS DE CARACTERES:
  - **MD5SUM**

```
mallet@mallet:~$ echo "hola mundo" | md5sum
1e04bb3f9f396d3b71d93d326ebfc42d  -
mallet@mallet:~$ echo "hola mund0" | md5sum
f9068a8a9ea3a5977e7e0cd31f225379  -
```

  - **SHA1SUM**

```
mallet@mallet:~$ echo "hola mundo" | sha1sum
f66b87fef5bf79850e957497ef2d529ce607b7ad  -
mallet@mallet:~$ echo "hola mund0" | sha1sum
ac00b5a1e6a80f3e60154bc408a70d3e99f00bf3  -
```

- **SHA256SUM**

```
mallet@mallet:~$ echo "hola mundo" | sha256sum
41d85e0b52944ee2917adfd73a2b7ce3d3c8368533a75e54db881fac6c9ad176  -
mallet@mallet:~$ echo "hola mund0" | sha256sum
f5c2b8c04ffc90f6494c434e5740582028eadda1fabfffe8ea3fc3accf9ae0d1  -
```

- **SHA512SUM**

```
mallet@mallet:~$ echo "hola mundo" | sha512sum
8d03191067cbd1988994a5c23dc524d8bea05471bf81a60191652b0dbc67e58ae893e5157fda32c6
f7135d7d26c875613f85a7dfb14e88455b3596c6fce025c9  -
mallet@mallet:~$ echo "hola mund0" | sha512sum
b86272c4904ba9798083045a90424e51b957b30072c0e6c7ba4387dbb13ce5a9bf6bfd08333940d9
f9635070503ea81652b9b5191ce030a721aaa575d15e07ec  -
```

- En estos ejemplos, podemos ver distintas características de las funciones hash y de las herramientas que hemos usado para modificar el mensaje en texto plano:
  - Se puede observar como una simple modificación en un carácter provoca que se cambie completamente la cadena resultado. Esto no quiere decir que la cadena sea aleatoria ya que si volvemos a repetir la entrada, la misma salida saldrá:

```
mallet@mallet:~$ echo "hola mund0" | sha512sum
b86272c4904ba9798083045a90424e51b957b30072c0e6c7ba4387dbb13ce5a9bf6bfd08333940d9
f9635070503ea81652b9b5191ce030a721aaa575d15e07ec  -
mallet@mallet:~$ echo "hola mund0" | sha512sum
b86272c4904ba9798083045a90424e51b957b30072c0e6c7ba4387dbb13ce5a9bf6bfd08333940d9
f9635070503ea81652b9b5191ce030a721aaa575d15e07ec  -
```

    Tan solo significa que como ya hemos visto, las funciones hash siguen la propiedad de avalancha.

  - Podemos observar también la longitud de las cadenas resultado, usando cada una de las herramientas. Si contamos los caracteres, saldrá exactamente el mismo número que sale en la tabla del apartado 3

- PARA FICHEROS:
  - Antes de todo:

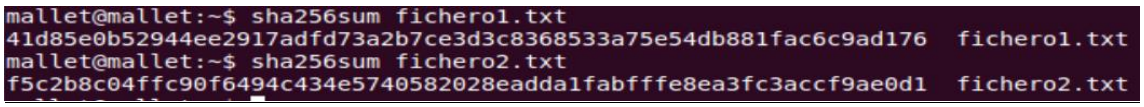
```
mallet@mallet:~$ echo "hola mundo" > fichero1.txt
mallet@mallet:~$ echo "hola mund0" > fichero2.txt
```

  - **MD5SUM**

```
mallet@mallet:~$ md5sum fichero1.txt
1e04bb3f9f396d3b71d93d326ebfc42d  fichero1.txt
mallet@mallet:~$ md5sum fichero2.txt
f9068a8a9ea3a5977e7e0cd31f225379  fichero2.txt
```

- **SHA1SUM**

- **SHA256SUM**



- **SHA512SUM**

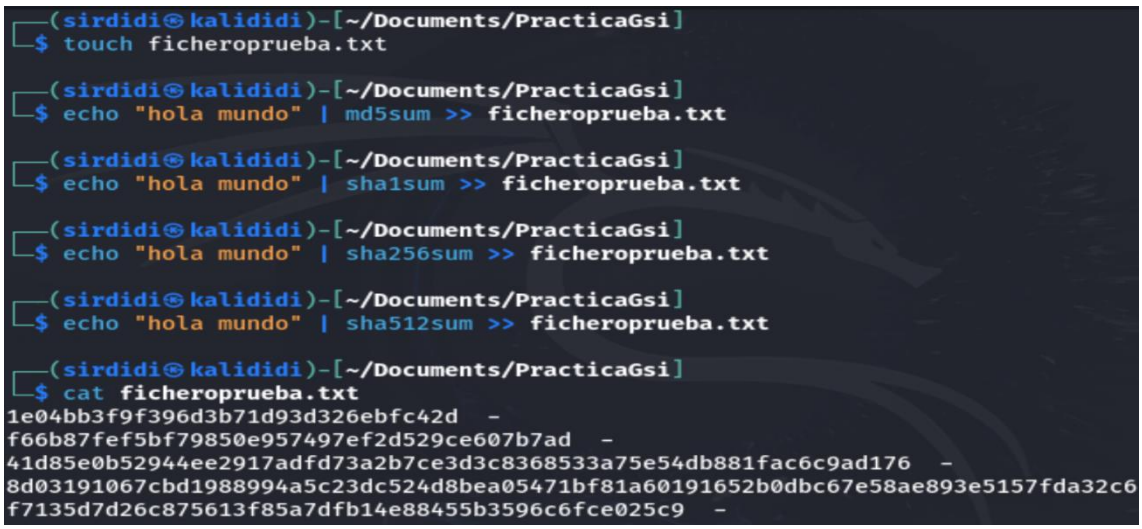
- Podemos ver que el hash creado para el contenido del fichero y el hash creado para el texto plano es el mismo. Esto es correcto, ahora bien debemos tener cuidado, ya que si al crear el fichero, añadimos un salto de línea al final de la frase que queremos modificar con la función hash, esta será completamente distinta.

5. USO DE HASHID PARA DESCUBRIR HERRAMIENTA USADA PARA EL CIFRADO

- Como hashid no se encuentra instalado en las máquinas virtuales y no podemos instalarlo (probablemente por la versión de Ubuntu, que no nos lo permite al ser esta antigua), vamos a realizar esta parte en nuestra máquina personal.

- Como podemos ver por la captura anterior, hashid se encuentra instalado en mi sistema personal por lo que podemos proceder con el ejercicio.

- Antes de todo vamos a añadir en el mismo fichero de texto todos los mensajes cifrados para así poder usar la herramienta hashid sobre cada una de las líneas.



- Para poder ver que hash es posible que hayamos estado usando en cada caso, usamos el siguiente comando:

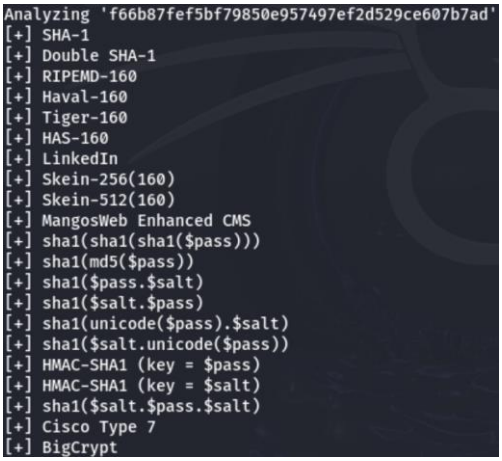
- Explicación:

    - Cat: leemos el fichero al completo
    - Sed: Eliminamos de cada línea caracteres no deseados que se han formado al escribir el hash en el fichero (saltos de línea por ejemplo)
    - While…do: Leemos cada línea del fichero
    - Hashid: Aplicamos sobre cada línea el comando hashid para obtener el nombre de las funciones hash que podrían ser válidas para la cadena formateada

- Veamos ahora las distintas salidas:

MD5SUM

SHA1SUM



SHA256SUM

SHA512SUM

- Vemos que el comando nos muestra diferentes funciones hash con las cuales puede coincidir el texto cifrado. Entre ellas se encuentran las funciones hash originales con las cuales creamos la cadena de texto cifrado.

6. HASH SOBRE LAS PROPIEDADES DE UN FICHERO

- Sobre las propiedades de un fichero también se puede formar una cadena de texto cifrada. Esto es muy útil, sobre todo para ver fácilmente si un archivo ha sido modificado o no (por la propiedad del efecto avalancha de las funciones hash). Veámoslo con el siguiente ejemplo:

- En un directorio, creamos un fichero y vemos los permisos con los que se crea. Este fichero con estos permisos tiene una función hash asociada:

```
root@mallet:/home/mallet/PracticaHash# ls -l ficheroprueba.txt
-rw-r--r-- 1 root root 0 2024-12-02 05:06 ficheroprueba.txt
root@mallet:/home/mallet/PracticaHash# ls -l ficheroprueba.txt | md5sum
7cdc0891d91f332e86b2646eec68b338  -
```

- Ahora cambiemos los permisos asociados al fichero, añadiendo por ejemplo permisos de escritura para los usuarios del mismo grupo. Generamos tras esto el hash asociado al fichero con los nuevos permisos:

```
root@mallet:/home/mallet/PracticaHash# chmod 664 ficheroprueba.txt
root@mallet:/home/mallet/PracticaHash# ls -l ficheroprueba.txt | md5sum
647ef6f067fbb7964e6d488ad2a25bda  -
```

- Como podemos ver, con solo cambiar uno de los permisos, cambia completamente el hash resultado, causando que se pueda ver perfectamente que ha habido un cambio entre el fichero original y el modificado.
- Como hemos dicho antes, una utilidad de obtener el hash de los archivos, puede ser ver fácilmente si el archivo ha sido modificado. Si enviamos a alguien el fichero inicial, esperando que no modifique nada, y recibimos el segundo con los permisos modificados, podremos observar fácilmente si el usuario al que enviamos el archivo intento, de forma probablemente malintencionada, modificar las propiedades de este.

7. ATAQUE DE FUERZA BRUTA CONTRA CONTRASEÑAS CIFRADAS

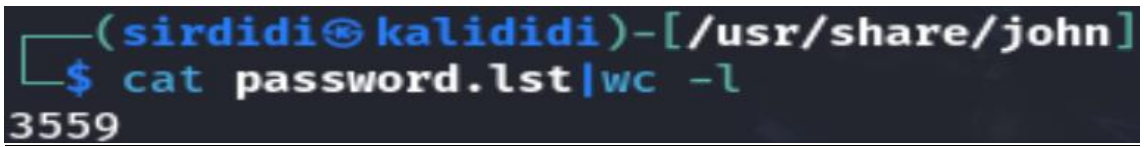
- Para poder probar a hacer un ataque de fuerza bruta sobre las contraseñas cifradas, debemos crear 2 usuarios (root y toor) con contraseña (admin y 123456):

```
┌──(sirdidi㉿kalididi)-[~/Documents/PracticaGsi]
└─$ sudo useradd root -m
[sudo] password for sirdidi:
useradd: user 'root' already exists

┌──(sirdidi㉿kalididi)-[~/Documents/PracticaGsi]
└─$ sudo useradd toor -m

┌──(sirdidi㉿kalididi)-[~/Documents/PracticaGsi]
└─$ sudo passwd root
New password:
Retype new password:
passwd: password updated successfully

┌──(sirdidi㉿kalididi)-[~/Documents/PracticaGsi]
└─$ sudo passwd toor
New password:
Retype new password:
passwd: password updated successfully
```

- Sabemos que aunque en nuestro sistema exista el fichero /etc/passwd, las contraseñas cifradas no se guardan en este fichero, sino que se guardan en /etc/shadow. Ahora necesitamos obtener las líneas del fichero shadow del usuario root y toor y ver las contraseñas cifradas:

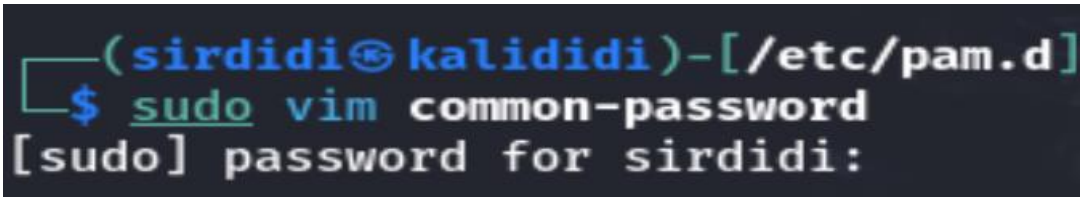
- Ahora, conociendo el formato de las cadenas de texto formadas por /etc/shadow, sabemos que la contraseña cifrada va desde los ":" detrás del nombre de usuario hasta los siguientes ":". Dentro de la propia contraseña cifrada, los 3 primeros caracteres (o los caracteres que estén entre "$") corresponden al algoritmo de encriptación que se está usando para codificar la contraseña. En nuestro caso, el algoritmo que se esta usando es el y que no se corresponde con ningún algoritmo conocido. Esto probablemente significa que esta distribución de Ubuntu, usa un algoritmo personalizado que no corresponde con ninguno de los normalmente conocidos (md5sum,shaXsum...)
- En cambio, vamos a comprobarlo con la máquina Mallet:

- En Mallet, si que se usa un algoritmo conocido, que sería el que corresponde a 6 que es SHA-512
- Ahora, sobre nuestra máquina personal, vamos a intentar a realizar un ataque de fuerza bruta usando la herramienta John The Ripper.
- Para ello, vamos a usar un wordlist que viene instalado con John The Ripper, que consiste en las contraseñas más comúnes. La herramienta irá probando hashes y si la contraseña se encuentra entre las de la librería, la adivinará:
- El wordlist se encuentra en el siguiente directorio y contiene las contraseñas que pone en la captura (una contraseña por línea), que corresponden a las contraseñas más usadas:

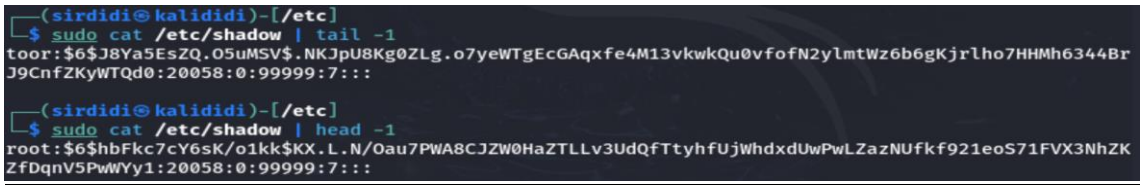


- El problema es que como hemos visto antes, con la codificación de las contraseñas con y, que corresponde a la función hash YESCRIPT, no podemos usar john, ya que no está contemplado. Vamos a cambiar para que la codificación de las contraseñas se realice en SHA512 (6):
    - En el siguiente fichero, hay que añadir la línea que viene en la captura (la línea ya viene preescrita pero pone YESCRIPT en vez de SHA512):

    - También debemos cambiar la configuración del siguiente archivo (al igual que antes, la línea ya viene preescrita pero debemos cambiar YESCRIPT por sha512):



- Vamos a comprobar ahora, creando el usuario de nuevo, si la codificación de las contraseñas ya es sha512:



- Vemos que ya sale 6, luego la codificación de las contraseñas se ha cambiado correctamente. Ahora ya podemos usar el john nos permita adivinar las contraseñas de root y toor a través de la contraseña cifrada:

```
┌──(sirdidi㉿kalididi)-[~/Documents/PracticaGsi]
└─$ sudo john --format=sha512crypt ./contrasenashash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 128/128 SSE2 2x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 4 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
123456           (toor)
1g 0:00:00:02 DONE 2/3 (2024-12-02 15:13) 0.4065g/s 1320p/s 1320c/s 1320C/s 123456..franklin
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

┌──(sirdidi㉿kalididi)-[~/Documents/PracticaGsi]
└─$ cat contrasenashash.txt
toor:$6$J8Ya5EsZQ.O5uMSV$.NKJpU8Kg0ZLg.o7yeWTgEcGAqxfe4M13vkwkQu0vfofN2ylmtWz6b6gKjrlho7HHMh6344Br
J9CnfZKyWTQd0:20058:0:99999:7:::
```

```
┌──(sirdidi㉿kalididi)-[~/Documents/PracticaGsi]
└─$ cat contrasenashash.txt
root:$6$hbFkc7cY6sK/o1kk$KX.L.N/Oau7PWA8CJZW0HaZTLLv3UdQfTtyhfUjWhdxdUwPwLZazNUfkf921eoS71FVX3NhZK
ZfDqnV5PwWYy1:20058:0:99999:7:::

┌──(sirdidi㉿kalididi)-[~/Documents/PracticaGsi]
└─$ sudo john --format=sha512crypt ./contrasenashash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 128/128 SSE2 2x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 4 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
admin            (root)
1g 0:00:00:03 DONE 2/3 (2024-12-02 15:18) 0.2666g/s 1617p/s 1617c/s 1617C/s Winnie..mobydick
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

- Podemos observar que al estar las contraseñas en el wordlist de john, este es capaz de adivinar las contraseñas mediante prueba y error, codificando las contraseñas con la herramienta de hashing sha-512, para después compararla con la cadena pasada.

8. CONSTRUCCIÓN PROGRAMA PARA CODIFICAR NOMBRE FICHEROS Y PROPIEDADES DADOS COMO ENTRADAS

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

int main(int argc, char *argv[]){
        //Declaración de variables
        char linea2[256];       //obtener lineas del fichero temporal
        char linea[256];        //obtener lineas del fichero pasado como argumento
        FILE *fichero;          //descriptor de fichero pasado como argumento
        char cadena1 [1024];    //guarda cadena que redirecciona al fichero temporal la salida del md5sum del contenido del fichero
        char cadena2 [1024];    //guarda cadena que redirecciona al fichero donde se guardaran las propiedades del fichero pasado como arg
        char cadena3 [1024];    //guarda cadena que redirecciona al fichero temporal la salida del md5sum sobre las propiedades del fichero
        FILE *fichero2;         //descriptor de fichero que guarda el archivo temporal
        char *cadenatemp = "/tmp/ficheropruebahash";    //cadena con fichero temporal que guardara las propiedades del fichero
        char *cadenatemp2 = "/tmp/ficheropruebahash2";  //cadena con fichero temporal que guardara los hashes
        int iteracion=0;        //numero de iteraciones al sacar por pantalla
        //comprobacion de parametros
        if(argc!=2){
                fprintf(stderr,"Error de parametros");
                exit(-1);
        }
        else{
                //abrimos fichero pasado como argumento para la lectura de los ficheros
                fichero = fopen(argv[1],"r");
                if(fichero==NULL){
                        fprintf(stderr,"Error al abrir el archivo");
                        exit(-1);
                }
                //leemos fichero linea a linea
                while(fgets(linea,sizeof(linea),fichero)!=NULL){
                        //creamos el primer hash y lo guardamos en el archivo temporal
                        sprintf(cadena1,"(md5sum %s)>%s",linea,cadenatemp2);
                        system(cadena1);
                        //creamos el segundo hash y lo guardamos en el archivo temporal
                        sprintf(cadena2,"(ls -l %s)>%s",linea,cadenatemp);
                        system(cadena2);
                        sprintf(cadena3,"(md5sum %s)>>%s",cadenatemp,cadenatemp2);
                        system(cadena3);
                        //abrimos el fichero temporal para imprimir los 2 hashes correctamente por pantalla
                        fichero2=fopen(cadenatemp2,"r");
                        if(fichero2 == NULL){
                                fprintf(stderr,"Error al abrir el fichero");
                                exit(-1);
                        }
                        linea[strlen(linea)-1]='\0';
                        //imprimimos el nombre del fichero
                        fprintf(stdout,"%s",linea);
                        iteracion=0;
                        //bucle de lectura de lineas del fichero temporal
                        while(fgets(linea2,sizeof(linea),fichero2)!=NULL){
                                linea2[strlen(linea2)-1]='\0';
                                //mostramos por pantalla el hash de el contenido del archivo
                                if(iteracion==0){
                                        linea2[strlen(linea2)-strlen(linea)-2]='\0';
                                        iteracion++;
                                }
                                //mostramos por pantalla el hash de las propiedades del archivo
                                else{
                                        linea2[strlen(linea2)-strlen(cadenatemp2)-1]='\0';
                                }
                                fprintf(stdout,";%s",linea2);
                        }
                        fprintf(stdout,"\n");
                        //cerramos fichero temporal
                        fclose(fichero2);
                        fflush(0);
                }
                //cerramos fichero pasado como argumento
                fclose(fichero);

        }
        //salimos
        return 0;
}
```

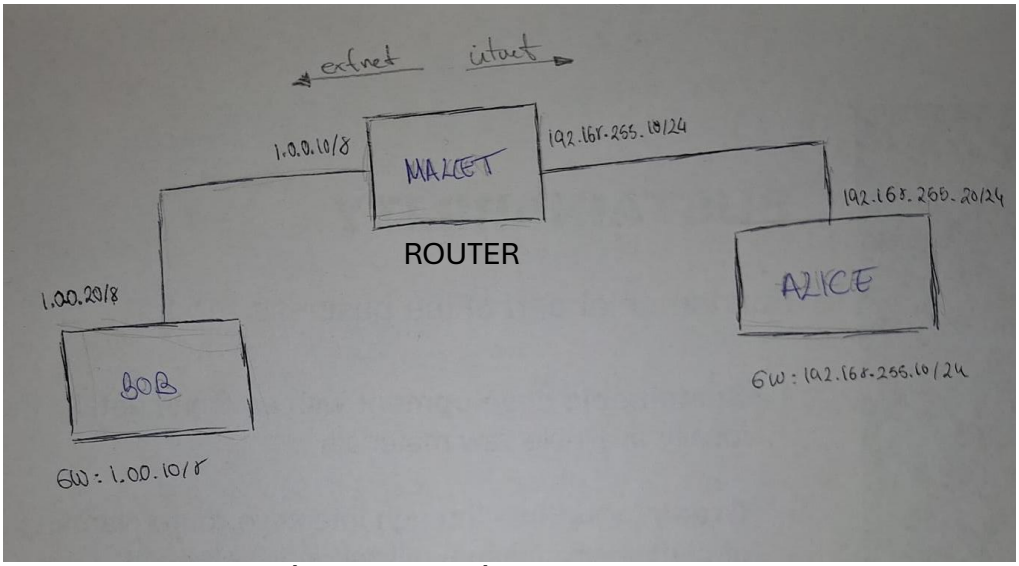
EJEMPLO DE SALIDA:

```
root@mallet:/home/mallet/PracticaHash# cat ficheroprueba.txt
ficheroprueba.txt
buildhash.c
root@mallet:/home/mallet/PracticaHash# ./buildhash ficheroprueba.txt
ficheroprueba.txt;4353adc5e3cee48b65c1b5a2c90e30b0;63a656de8675064736c590fc17c8ccdb
buildhash.c;ce865ff2360c06661880cd8edcc70f3d;ec295f2e7027f2f995dce3595b722eb1
root@mallet:/home/mallet/PracticaHash# md5sum ficheroprueba.txt
4353adc5e3cee48b65c1b5a2c90e30b0  ficheroprueba.txt
root@mallet:/home/mallet/PracticaHash# ls -l ficheroprueba.txt | md5sum
63a656de8675064736c590fc17c8ccdb  -
root@mallet:/home/mallet/PracticaHash# md5sum buildhash.c
ce865ff2360c06661880cd8edcc70f3d  buildhash.c
root@mallet:/home/mallet/PracticaHash# ls -l buildhash.c | md5sum
ec295f2e7027f2f995dce3595b722eb1  -
```

- Podemos observar perfectamente, como coinciden los hashes de la salida del programa escrito en C con los hashes generados por las funciones directamente.
- Muy útil para automatizar la creación de hashes desde un fichero dado

PARTE 2

1. DIBUJO DEL DIAGRAMA DE RED



2. CONFIGURACIÓN DE LA MÁQUINA MALLET COMO ROUTER

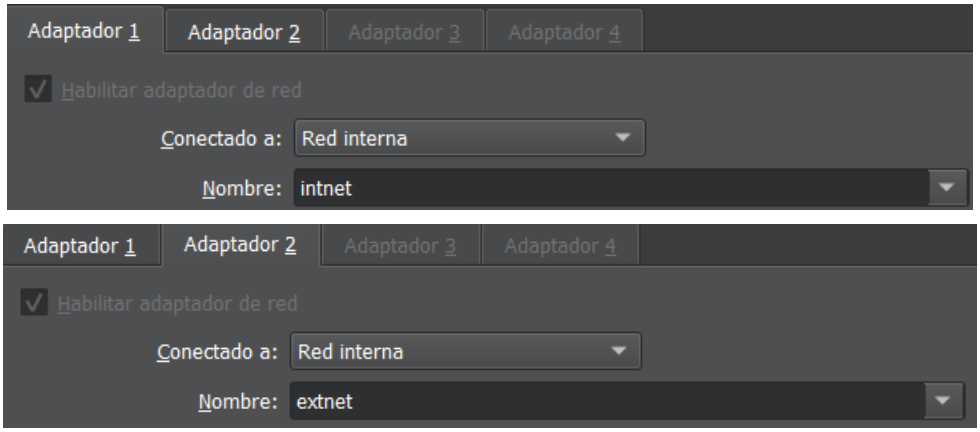
- Para conseguir que Mallet funcione como router, este tiene que tener al menos 2 interfaces de red. La eth4 funcionará como interfaz de red para la subred interna, mientras que la eth5 funcionará para la red externa.
- Para configurar las interfaces, debemos modificar el archivo /etc/network/interfaces (y hacer Sudo /etc/init.d/networking restart):

```
auto lo eth4 eth5
iface lo inet loopback

iface eth4 inet static
        address 192.168.255.10
        netmask 255.255.255.0

iface eth5 inet static
        address 1.0.0.10
        netmask 255.0.0.0
```

```
mallet@mallet:/etc$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP q
en 1000
    link/ether 08:00:27:2e:fa:eb brd ff:ff:ff:ff:ff:ff
    inet 192.168.255.10/24 brd 192.168.255.255 scope global eth4
    inet6 fe80::a00:27ff:fe2e:faeb/64 scope link
       valid_lft forever preferred_lft forever
3: eth5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP q
en 1000
    link/ether 08:00:27:b7:a5:59 brd ff:ff:ff:ff:ff:ff
    inet 1.0.0.10/8 brd 1.255.255.255 scope global eth5
    inet6 fe80::a00:27ff:feb7:a559/64 scope link
       valid_lft forever preferred_lft forever
```

- Ahora Mallet ya tiene 2 interfaces para enrutar, aunque antes de que pueda hacerlo, debemos realizar unas configuraciones extras:

    - Debemos indicar a virtualbox, que las 2 interfaces de Mallet están en redes distintas (redes en las que luego estarán alice y bob). Aunque la eth5 corresponda con una red externa, la supondremos como una red interna por sencillez:



    - Debemos de cambiar el bit de enrutamiento en Mallet para que así pueda funcionar como router. Esto se puede hacer de 2 formas, de forma temporal o persistente. En cualquiera de las 2 formas debemos ser super usuario (no basta con hacer sudo):

        - Temporal: /proc/sys/net/ipv4/ip_forward (simplemente cambiar el 0 que está por defecto en este archivo por un “1”

```
mallet@mallet:/proc/sys/net/ipv4$ cat ip_forward
1
```

        - Persistente: /etc/sysctl.conf . Hay que descomentar una línea del fichero:

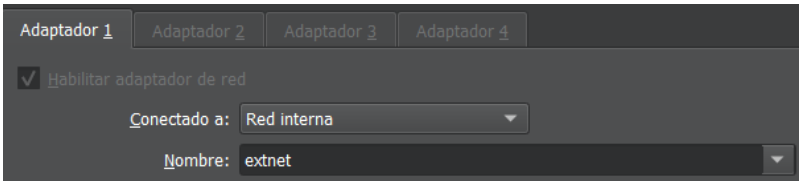
```
# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1
```

- Tras esto, cuando el resto de máquinas estén configuradas, Mallet ya podrá enrutar entre los distintos dispositivos de la red

3. CONFIGURACIÓN DE BOB

- Ahora Bob ya no se encuentra dentro de una red interna como en las prácticas anteriores, ahora queremos simular que esta se encuentra en el exterior de la subred en la que ahora tan solo se encuentra Alice.

- Debemos indicar que Bob se encuentra en la red extnet, desde virtualbox:



- Y reconfigurar la interfaz de Bob para que este se encuentre ahora en la subred 1.0.0.0/8

```
bob:/home/bob# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether 08:00:27:fe:c6:42 brd ff:ff:ff:ff:ff:ff
    inet 1.0.0.20/8 brd 1.255.255.255 scope global eth0
    inet6 fe80::a00:27ff:fefe:c642/64 scope link
       valid_lft forever preferred_lft forever
```

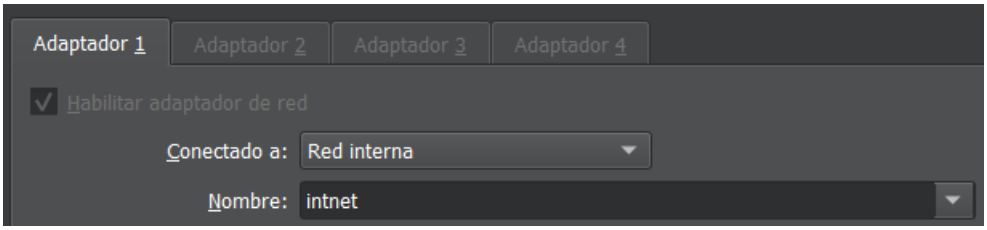
```
bob:/etc/network# cat interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug eth0
#iface eth0 inet dhcp

allow-hotplug eth1
#iface eth1 inet dhcp

auto eth0
iface eth0 inet static
        address 1.0.0.20
        netmask 255.0.0.0
        gateway 1.0.0.10
```

- Como podemos ver, bob ya esta configurado para que este en la red externa, con IP 1.0.0.20 y con el Gateway de la interfaz de Mallet que da a la red externa, para conseguir el enrutamiento entre paquetes

4. CONFIGURACIÓN DE ALICE

- Similar a la configuración de bob, pero alice sigue estando en la subred interna y tiene de Gateway la interfaz de red de Mallet que da a dicha subred:

```
alice@alice:/etc/network$ cat interfaces
auto lo
iface lo inet loopback

auto eth3
iface eth3 inet static
                address 192.168.255.20
        netmask 255.255.255.0
                gateway 192.168.255.10
alice@alice:/etc/network$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether 08:00:27:39:14:8d brd ff:ff:ff:ff:ff:ff
    inet 192.168.255.20/24 brd 192.168.255.255 scope global eth3
    inet6 fe80::a00:27ff:fe39:148d/64 scope link
       valid_lft forever preferred_lft forever
```



5. PRUEBAS A NIVEL DE RED

DESDE↓\HASTA→	ALICE	MALLET(INTERIOR)

ALICE	-	`alice@alice:/etc/network$ ping -c 1 192.168.255.10`
		`PING 192.168.255.10 (192.168.255.10) 56(84) bytes of data.`
		`64 bytes from 192.168.255.10: icmp_seq=1 ttl=64 time=4.84 ms`
		
		`--- 192.168.255.10 ping statistics ---`
		`1 packets transmitted, 1 received, 0% packet loss, time 0ms`
		`rtt min/avg/max/mdev = 4.840/4.840/4.840/0.000 ms`

MALLET(INT)	`root@mallet:/usr# ping -I eth4 192.168.255.20 -c 2`
	`PING 192.168.255.20 (192.168.255.20) from 192.168.255.10 eth4: 56(84) bytes of data.`
	`64 bytes from 192.168.255.20: icmp_seq=1 ttl=64 time=1.02 ms`
	`64 bytes from 192.168.255.20: icmp_seq=2 ttl=64 time=3.50 ms`
	
	`--- 192.168.255.20 ping statistics ---`
	`2 packets transmitted, 2 received, 0% packet loss, time 1006ms`
	`rtt min/avg/max/mdev = 1.029/2.266/3.503/1.237 ms`
		-

MALLET(EXT)	`root@mallet:/usr# ping -I eth5 192.168.255.20 -c 2`
	`PING 192.168.255.20 (192.168.255.20) from 1.0.0.10 eth5: 56(84) bytes of data.`
	`From 1.0.0.10 icmp_seq=1 Destination Host Unreachable`
	`From 1.0.0.10 icmp_seq=2 Destination Host Unreachable`
	
	`--- 192.168.255.20 ping statistics ---`
	`2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 999ms`
	`, pipe 2`
		`root@mallet:/usr# ping -I eth5 192.168.255.10 -c 2`
		`PING 192.168.255.10 (192.168.255.10) from 1.0.0.10 eth5: 56(84) bytes of data.`
		`From 1.0.0.10 icmp_seq=1 Destination Host Unreachable`
		`From 1.0.0.10 icmp_seq=2 Destination Host Unreachable`
		
		`--- 192.168.255.10 ping statistics ---`
		`2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1003ms`
		`, pipe 2`

BOB	`bob:/etc/network# ping 192.168.255.20 -c 2`
	`PING 192.168.255.20 (192.168.255.20) 56(84) bytes of data.`
	`64 bytes from 192.168.255.20: icmp_seq=1 ttl=63 time=3.06 ms`
	`64 bytes from 192.168.255.20: icmp_seq=2 ttl=63 time=8.34 ms`
	
	`--- 192.168.255.20 ping statistics ---`
	`2 packets transmitted, 2 received, 0% packet loss, time 1008ms`
	`rtt min/avg/max/mdev = 3.069/5.709/8.349/2.640 ms`
		`bob:/etc/network# ping 192.168.255.10 -c 2`
		`PING 192.168.255.10 (192.168.255.10) 56(84) bytes of data.`
		`64 bytes from 192.168.255.10: icmp_seq=1 ttl=64 time=1.35 ms`
		`64 bytes from 192.168.255.10: icmp_seq=2 ttl=64 time=2.79 ms`
		
		`--- 192.168.255.10 ping statistics ---`
		`2 packets transmitted, 2 received, 0% packet loss, time 1000ms`
		`rtt min/avg/max/mdev = 1.350/2.070/2.799/0.721 ms`

DESDE↓\HASTA→	MALLET(EXTERIOR)	BOB

ALICE	`alice@alice:/etc/network$ ping -c 1 1.0.0.10`
	`PING 1.0.0.10 (1.0.0.10) 56(84) bytes of data.`
	`64 bytes from 1.0.0.10: icmp_seq=1 ttl=64 time=1.16 ms`
	
	`--- 1.0.0.10 ping statistics ---`
	`1 packets transmitted, 1 received, 0% packet loss, time 0ms`
	`rtt min/avg/max/mdev = 1.166/1.166/1.166/0.000 ms`
		`alice@alice:/etc/network$ ping -c 1 1.0.0.20`
		`PING 1.0.0.20 (1.0.0.20) 56(84) bytes of data.`
		`64 bytes from 1.0.0.20: icmp_seq=1 ttl=63 time=8.24 ms`
		
		`--- 1.0.0.20 ping statistics ---`
		`1 packets transmitted, 1 received, 0% packet loss, time 0ms`
		`rtt min/avg/max/mdev = 8.247/8.247/8.247/0.000 ms`

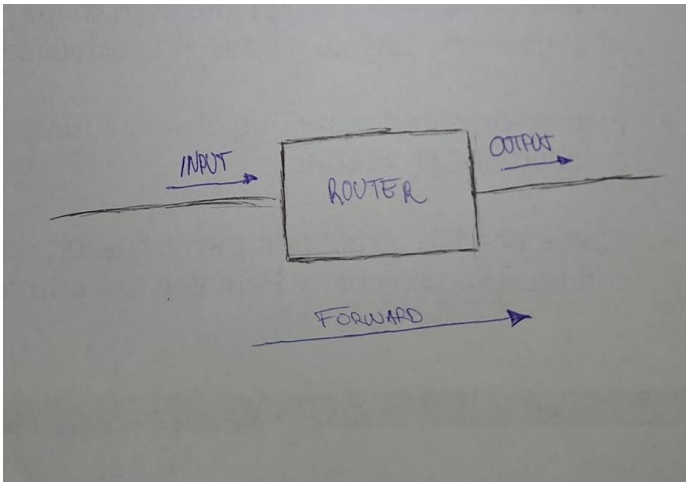
MALLET(INT)	`root@mallet:/usr# ping -I eth4 1.0.0.10 -c 2`
	`PING 1.0.0.10 (1.0.0.10) from 192.168.255.10 eth4: 56(84) bytes of data.`
	`From 192.168.255.10 icmp_seq=1 Destination Host Unreachable`
	`From 192.168.255.10 icmp_seq=2 Destination Host Unreachable`
	
	`--- 1.0.0.10 ping statistics ---`
	`2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1004ms`
	`, pipe 2`
		`root@mallet:/usr# ping -I eth4 1.0.0.20 -c 2`
		`PING 1.0.0.20 (1.0.0.20) from 192.168.255.10 eth4: 56(84) bytes of data.`
		`From 192.168.255.10 icmp_seq=1 Destination Host Unreachable`
		`From 192.168.255.10 icmp_seq=2 Destination Host Unreachable`
		
		`--- 1.0.0.20 ping statistics ---`
		`2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1003ms`
		`, pipe 2`

MALLET(EXT)	-	`root@mallet:/usr# ping -I eth5 1.0.0.20 -c 2`
		`PING 1.0.0.20 (1.0.0.20) from 1.0.0.10 eth5: 56(84) bytes of data.`
		`64 bytes from 1.0.0.20: icmp_seq=1 ttl=64 time=4.57 ms`
		`64 bytes from 1.0.0.20: icmp_seq=2 ttl=64 time=1.71 ms`
		
		`--- 1.0.0.20 ping statistics ---`
		`2 packets transmitted, 2 received, 0% packet loss, time 1004ms`
		`rtt min/avg/max/mdev = 1.711/3.142/4.573/1.431 ms`

BOB	`bob:/etc/network# ping 1.0.0.10 -c 2`
	`PING 1.0.0.10 (1.0.0.10) 56(84) bytes of data.`
	`64 bytes from 1.0.0.10: icmp_seq=1 ttl=64 time=1.45 ms`
	`64 bytes from 1.0.0.10: icmp_seq=2 ttl=64 time=4.02 ms`
	
	`--- 1.0.0.10 ping statistics ---`
	`2 packets transmitted, 2 received, 0% packet loss, time 1002ms`
	`rtt min/avg/max/mdev = 1.451/2.739/4.027/1.288 ms`
		-

- Cuestiones a considerar:
    - Conexiones no posibles:
        - Mallet(interior) > Mallet(exterior): Los sistemas de routing no permiten que un paquete se envíe directamente de una interfaz interna a la interfaz externa del mismo dispositivo sin pasar por un mecanismo de routing adicional.
        - Mallet(interior) > Bob: Si Mallet (interior) no tiene una ruta configurada para que el tráfico interno se dirija hacia la red externa a través de Mallet (exterior), entonces el tráfico no podrá llegar a Bob.
        - Mallet(exterior) > Alice: Igual que desde Mallet(interior) > Bob
        - Mallet(exterior) > Mallet(interior): misma explicación que antes
    - Decremento del TTL:
        - Podemos observar, que cuando se pasa de una red a otra mediante el router, el TTL disminuye ya que atraviesa el router. El TTL indica el número de saltos que le faltan a un paquete antes de ser descartado. Al pasar con la regla FORWARD por el router, el TTL se decrementa.

6. DIFERENCIAS ENTRE LAS REGLAS INPUT, OUTPUT y FORWARD



INPUT: se aplican a los paquetes entrantes que están destinados al propio router o dispositivo.

OUTPUT: se aplican a los paquetes salientes que se originan en el propio router/dispositivo.

FORWARD: se aplican a los paquetes que no están destinados al propio router, sino que simplemente están siendo encaminados (ruteados) a través de él de una red a otra.

7. CONFIGURACIÓN DE REGLAS EN EL FIREWALL PARA QUE SOLO SE PUEDAN HACER PETICIONES ICMP DESDE RED INTERNA

- Vamos a configurar las reglas de iptables para solo permitir paquetes ICMP desde la red interna hasta la externa:

```
root@mallet:/usr# sudo iptables -A FORWARD -d 192.168.255.0/24 -p icmp --icmp-type echo-request -j DROP
root@mallet:/usr# sudo iptables -A INPUT -p icmp --icmp-type echo-request -d 192.168.255.10 -j DROP
root@mallet:/usr# sudo iptables -A INPUT -p icmp --icmp-type echo-request -d 1.0.0.10 -j DROP
root@mallet:/usr# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       icmp --  anywhere             mallet.local         icmp echo-request
DROP       icmp --  anywhere             mallet.local         icmp echo-request

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

- Comprobemos ahora que pasa si queremos acceder desde Alice a Bob:

```
alice@alice:/etc/network$ ping 1.0.0.20 -c 2
PING 1.0.0.20 (1.0.0.20) 56(84) bytes of data.
64 bytes from 1.0.0.20: icmp_seq=1 ttl=63 time=7.55 ms
64 bytes from 1.0.0.20: icmp_seq=2 ttl=63 time=2.45 ms

--- 1.0.0.20 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 2.458/5.006/7.555/2.549 ms
```

- Podemos observar que, ya que no hemos configurado ninguna regla que impida a Alice llegar a Bob, es posible que esta se comunique con Bob.
- Veamos ahora al revés:

```
bob:/etc/network# ping 192.168.255.20 -c 2
PING 192.168.255.20 (192.168.255.20) 56(84) bytes of data.

--- 192.168.255.20 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1001ms
```

- Podemos ver que como está configurada la regla para FORWARD, que no deja pasar paquetes desde la red externa a la red interna, este paquete se dropea en el router y no llega a su destino, dándonos en el ping un 100% de packetloss.
- Vamos a ver también si nos podemos comunicar desde el exterior con las interfaces del router:

```
bob:/etc/network# ping 192.168.255.10 -c 2
PING 192.168.255.10 (192.168.255.10) 56(84) bytes of data.

--- 192.168.255.10 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1001ms

bob:/etc/network# ping 1.0.0.10 -c 2
PING 1.0.0.10 (1.0.0.10) 56(84) bytes of data.

--- 1.0.0.10 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 999ms
```

- Vemos que aquí tampoco podemos pinguear a las interfaces de Mallet, por culpa de las reglas antes establecidas de INPUT. Como medida de seguridad esto es muy conveniente.

8. CONFIGURAR REGLAS SSH

- SSH usa, por defecto, el puerto 22 para sus conexiones. Si implementamos reglas que capen desde el exterior de la red dicho puerto, solo te podrás conectar con SSH desde el interior al exterior y no viceversa
- Como en el enunciado viene distinto, aclaramos: entendemos que debe ser Alice al estar en la red interna, con IP 192.168.255.20, la que debe poder conectarse con Bob, con IP 1.0.0.20, mediante SSH, pero no al revés.
- También debemos capar a Bob para que no pueda acceder a las interfaces de Mallet, ya que sino este podría desde Mallet, acceder a Alice ya sea mediante un SSH u otro protocolo (como en el punto 7).
- Las reglas son las siguientes:

```
mallet@mallet:/usr/share$ sudo iptables -A INPUT -d 1.0.0.10 -p tcp --dport 22 -j DROP
mallet@mallet:/usr/share$ sudo iptables -A INPUT -d 192.168.255.10 -p tcp --dport 22 -j DROP
mallet@mallet:/usr/share$ sudo iptables -A FORWARD -d 192.168.255.0/24 -p tcp --dport 22 -j DROP
```

- Vemos todas las reglas, tanto las de ssh como las de icmp anteriores en la siguiente captura:

```
mallet@mallet:/usr/share$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       icmp --  anywhere             mallet.local         icmp echo-request
DROP       icmp --  anywhere             mallet.local         icmp echo-request
DROP       tcp  --  anywhere             mallet.local         tcp dpt:ssh
DROP       tcp  --  anywhere             mallet.local         tcp dpt:ssh

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
DROP       icmp --  anywhere             192.168.255.0/24     icmp echo-request
DROP       tcp  --  anywhere             192.168.255.0/24     tcp dpt:ssh
```

- Y ahora hagamos las siguiente comprobaciones:
  - ALICE PUEDE HACER SSH A BOB:

```
alice@alice:/etc/network$ ssh bob@1.0.0.20
bob@1.0.0.20's password:
Linux bob 2.6.17.1 #1 SMP Mon Jul 5 18:50:04 CEST 2010 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.
You have mail.
Last login: Mon Dec  2 05:10:57 2024 from 192.168.255.20
bob@bob:~$
```

  - BOB NO PUEDE HACER SSH A MALLET/ALICE:

```
bob:/etc/network# ssh alice@192.168.255.20

bob:/etc/network# ssh mallet@192.168.255.10

bob:/etc/network# ssh mallet@1.0.0.10
```

    - Por desgracia, al dropear los paquetes, no se nos muestra por pantalla nada y debemos cancelar el comando ssh nosotros mismos, con el afán de asegurarnos de que esto ocurra

porque el router deniega los paquetes, vamos a cambiar ligeramente las reglas de IpTables.

- Vamos a quitar las reglas anteriores y añadir nuevas que rechacen el paquete en vez de dropearlo.

```
mallet@mallet:/usr/share$ sudo iptables -D FORWARD -d 192.168.255.0/24 -p tcp --dport 22 -j DROP
mallet@mallet:/usr/share$ sudo iptables -D INPUT -d 192.168.255.10 -p tcp --dport 22 -j DROP
mallet@mallet:/usr/share$ sudo iptables -D INPUT -d 1.0.0.10 -p tcp --dport 22 -j DROP
mallet@mallet:/usr/share$ sudo iptables -A INPUT -d 1.0.0.10 -p tcp --dport 22 -j REJECT
mallet@mallet:/usr/share$ sudo iptables -A INPUT -d 192.168.255.10 -p tcp --dport 22 -j REJECT
mallet@mallet:/usr/share$ sudo iptables -A FORWARD -d 192.168.255.0/24 -p tcp --dport 22 -j REJECT
```

- Veamos que ocurre ahora:

```
bob:/etc/network# ssh alice@192.168.255.20
ssh: connect to host 192.168.255.20 port 22: Connection refused
bob:/etc/network# ssh mallet@192.168.255.10
ssh: connect to host 192.168.255.10 port 22: Connection refused
bob:/etc/network# ssh mallet@1.0.0.10
ssh: connect to host 1.0.0.10 port 22: Connection refused
```

- Ahora Mallet si rechaza los paquetes y podemos observar que la conexión ssh no se llega a establecer, consiguiendo limitar ssh desde el exterior de la subred.