HASHING, CIFRADO Y CERTIFICADOS

Práctica 3

Informe de prácticas

Parte 1

1. A partir de la página de manual sobre la forma de uso de md5sum y sha1sum (y documentación extra que pueda obtener sobre MD5 y SHA-*), elabora un resumen de cómo se emplean estas aplicaciones para generar hashcodes a partir de documentos. Discuta brevemente las debilidades conocidas y reportadas.

MD5

El algoritmo de hashing MD5 usa una **compleja fórmula matemática** para crear un hash. Convierte los datos en **bloques de tamaños específicos** y los manipula varias veces. Mientras esto ocurre, el algoritmo añade un **valor único** en el cálculo y convierte el resultado en una pequeña firma o hash.

Los pasos del algoritmo MD5 son complejos por una razón: no se puede invertir este proceso y generar el archivo original a partir del hash. Pero la **misma entrada siempre producirá la misma salida**. Eso es lo que los hace tan útiles para verificar datos.

Hashing MD5 convierte los datos en una cadena de **32 caracteres**. Del mismo modo, un archivo de 1,2 GB también genera un hash con el mismo número de caracteres. **Si cambia un solo bit** en un archivo, independientemente de lo grande que sea el archivo, **la información del hash cambiará completa e irreversiblemente**. Nada, salvo una copia exacta, pasará la prueba MD5.

Debilidades
El MD5 se usaba antes para la seguridad y el cifrado de datos, pero hoy en día su uso principal es la autenticación. Dado que un hacker con un ordenador muy potente puede hacer que un archivo malicioso genere el mismo hash que uno inofensivo, generando una **colisión MD5**, el MD5 no es seguro en caso de que alguien manipule un archivo. Pero si tan solo está copiando un archivo de un lugar a otro, el MD5 servirá.

SHA1
Una de las características principales de los algoritmos SHA es la **no reflexividad**. Es decir, no es posible determinar la cadena de bits de origen del hash. Solamente se puede conseguir el hash a partir del fichero a encriptar. Otra característica que ha cambiado según sus versiones es **la extensión de bits** de cada hash. En las primeras versiones (SHA-0 y SHA-1), las funciones producirían hashes de 160 bits (20 bytes), pero surgieron después actualizaciones con una mayor longitud.

El **algoritmo SHA-1** produce como resultado **un hash de 160 bits** y sus mejoras con respecto al SHA-0 fueron desarrolladas por la Agencia de Seguridad Nacional de Estados Unidos (NSA). Con el tiempo, se descubrieron sus debilidades y se desarrollaron versiones para producir funciones hash más extensas y seguras.

Debilidades

La vulnerabilidad matemática del SHA-1 se descubrió en el año 2005. A partir del año 2010, comenzó a dejar de estar aprobada para todos sus usos en el área de la criptografía. De hecho, en el año 2017, se descubrió un ataque de colisión en contra de este algoritmo.

2. Use ambas herramientas para construir el hash asociado a diferentes tiras de caracteres que difieran en pocos caracteres. Analice y documente las diferencias entre ambas.

Desde la máquina *mallet* comprobamos si podemos lanzar md5sum y sha1sum. Una vez comprobado, utilizamos estas funciones hash mediante el uso de **tuberías**.

md5sum

Para generar un resumen utilizando la función hash md5sum empleamos el siguiente comando: **echo Saludos | md5sum** el cual nos genera el siguiente resumen:

```
root@mallet:/home/mallet# echo Saludos | md5sum
748a90289dc05063379050bce16cb1a4  -
```

Si realizamos unas pocas modificaciones sobre la cadena introducida, obtenemos lo siguiente: **echo Saludo | md5sum**

```
root@mallet:/home/mallet# echo Saludo | md5sum
b609d70da553e618615b847fdaecb557  -
```

Como se puede observar, aunque cambiamos un solo carácter, el código cifrado obtenido es totalmente distinto al que tuvimos antes, obteniendo en ambos casos una salida de **32 caracteres hexadecimales o 128 bits**.

sha1sum

Procedemos a utilizar la misma cadena anterior, pero ahora hacemos uso de la función hash sha1sum, donde obtenemos la siguiente codificación:

```
root@mallet:/home/mallet# echo Saludos | sha1sum
4acf68ff1f0b773a7166012a86e6b682d30f6a4d  -
```

Mientras que, si modificamos de nuevo la cadena, obtenemos una codificación distinta al igual que ocurría con md5sum.

```
root@mallet:/home/mallet# echo Saludo | sha1sum
cfe3ae8ce421764fa64bb8b4b2416b370993b9e5  -
```

En este caso, la longitud de los mensaje cifrados es mayor ya que el tamaño de la salida es de **40 caracteres hexadecimales o 160 bits**.

3. Elija cualquier fichero presente en tu ordenador (puedes crearlo) y obtenga el hashcode usando ambas herramientas, describiendo la salida que se obtiene.

En este caso, nos encontramos con el archivo *testttttt.pdf* en la máquina y aparte, creamos un archivo más sencillo y pequeño para comparar el cifrado realizado.

- Comenzamos con **md5sum**. Escribimos el siguientes comando: md5sum testttttt.pdf y obtenemos el siguiente cifrado:

```
root@mallet:/home/mallet# md5sum testttttt.pdf
76379ef9ddf935d80b397646ece7f0bc  testttttt.pdf
```

Como se puede observar, obtenemos un resumen cifrado de 32 cifras, sin importar cual sea el tamaño del fichero que le hayamos pasado.
Para verificar esto, realizamos lo mismo con un fichero de menor tamaño llamado *claves.txt* cuyo contenido es el siguiente:

```
root@mallet:/home/mallet# cat claves.txt
username : admin
password : 1234
phone : +1 445 3469
```

Y si pasamos a codificar el fichero obtenemos:

```
root@mallet:/home/mallet# md5sum claves.txt
08325b0a233f1e29c6d8a2c402855b66  claves.txt
```

Como podemos observar, obtenemos siempre un mensaje cifrado de 32 caracteres, por lo que el tamaño del fichero no influye en el tamaño de la salida.

- Pasamos a utilizar la función hash **sha1sum**: Escribimos sha1sum testttttt.pdf y obtenemos lo siguiente:

```
root@mallet:/home/mallet# sha1sum testttttt.pdf
bcf3202e577284717465cbe7d6ebbd76e7223dc0  testttttt.pdf
```

Y si ahora codificamos el fichero claves.txt obtenemos:

```
root@mallet:/home/mallet# sha1sum claves.txt
07eeeae5dfb80b81a8f2690e793b61cb563cb4f3  claves.txt
```

En este caso, obtenemos un mensaje de 40 caracteres sea cual sea el tamaño del fichero.

4. Sobre los hashes generados en el punto anterior, utiliza la herramienta hashid para tratar de obtener cuál ha sido la función de hash utilizada para crear los resúmenes anteriores.

hashID es una herramienta de identificación de algoritmos hash. Por lo tanto, lo que vamos a hacer a continuación es lo siguiente: dado el mensaje cifrado que hemos obtenido anteriormente trataremos de obtener el algoritmo hash correspondiente.

- Usaremos los mensajes encriptados obtenidos a partir de cifrar el fichero claves.txt

1) Si escribimos el siguiente comando: **hashid 08325b0a233f1e29c6d8a2c402855b66** obtenemos la siguiente lista:

```
jhon@LAPTOP-91HTN3Q5:~/GSI/Prac_3$ hashid 08325b0a233f1e29c6d8a2c402855b66
Analyzing '08325b0a233f1e29c6d8a2c402855b66'
[+] 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
```

Donde se muestran las posibles funciones hash que se han utilizado. Como se puede ver, la función hash MD5 aparece la segunda en la lista, lo que quiere decir que se trata de un algoritmo muy probable.

2) Si ahora escribimos **hashid 07eeeae5dfb80b81a8f2690e793b61cb563cb4f3** obtenemos esta nueva lista:

```
jhon@LAPTOP-91HTN3Q5:~/GSI/Prac_3$ hashid 07eeeae5dfb80b81a8f2690e793b61cb563cb4f3
Analyzing '07eeeae5dfb80b81a8f2690e793b61cb563cb4f3'
[+] SHA-1
[+] Double SHA-1
[+] RIPEMD-160
[+] Haval-160
[+] Tiger-160
[+] HAS-160
[+] LinkedIn
[+] Skein-256(160)
[+] Skein-512(160)
```

Donde se puede ver que, efectivamente el algoritmo hash que se ha utilizado es el algoritmo SHA-1.

5. A partir de la información sobre el fichero (propiedades) que se pueden obtener, por ejemplo, con la orden ls -l, genere un hashcode. Modifique alguna propiedad del fichero y vuelva a obtener el hashcode. ¿qué conclusiones obtienes?

El comando ls -l nos permite listar los ficheros en formato de una sola columna, listando un fichero por cada línea con la siguiente información: permisos del fichero, el número de enlace, nombre del propietario, nombre del grupo al que pertenece, tamaño en bytes, una marca de tiempo y nombre del fichero.

Si ejecutamos el comando **ls -l claves.txt** obtenemos lo siguiente:

```
-rw-r--r-- 1 jhon jhon 53 Oct 14  2022 claves.txt
```

A continuación, procedemos a generar un hashcode a partir de las propiedades vistas del fichero mediante el siguiente comando: ls -l claves.txt | md5sum

```
jhon@LAPTOP-91HTN3Q5:~/GSI/Prac_3$ ls -l claves.txt | md5sum
2dc719b896c63ec8674a0bf8750e2966  -
jhon@LAPTOP-91HTN3Q5:~/GSI/Prac_3$
```

Ahora nos disponemos a cambiar alguna propiedad del fichero como, por ejemplo, los permisos de este. Para ello, utilizamos el comando **chmod 666** mediante el cual permitimos a todos los usuarios leer y escribir en el archivo.

```
jhon@LAPTOP-91HTN3Q5:~/GSI/Prac_3$ chmod 666 claves.txt
jhon@LAPTOP-91HTN3Q5:~/GSI/Prac_3$ ls -l
total 0
-rw-rw-rw- 1 jhon jhon 57 Oct 14 16:38 claves.txt
jhon@LAPTOP-91HTN3Q5:~/GSI/Prac_3$
```

Volvemos a obtener un hashcode a partir de las propiedades del fichero y obtenemos lo siguiente:

```
jhon@LAPTOP-91HTN3Q5:~/GSI/Prac_3$ ls -l claves.txt | md5sum
5f08aa041ba3fbfa256a06c7a75fd59a  -
```

De esta manera, obtenemos el hashcode y **comprobamos que efectivamente no coincide con el anterior**, debido a que cualquier mínimo cambio genera un hashcode totalmente distinto.

6. Sobre un sistema Ubuntu20.04.2 LTS, crea dos usuarios (root/toor), (admin/123456). Determina cuál es la función de hash utilizada para que las claves de los usuarios no se almacenen en texto plano y realiza un ataque de fuerza bruta sobre las contraseñas de los usuarios almacenadas por el sistema operativo en el fichero /etc/shadow. ¿Para qué puede ser útil la herramienta John The Ripper?

Debido a que en la máquina en la que se realiza la práctica ya existen tanto el usuario root como el admin, se procederá a crear 2 nuevos usuarios con nombres distintos: (**alice/toor**) y (**bob/123456**).

El fichero /etc/passwd contiene los usuarios del sistema y por su parte, el fichero /etc/shadow contiene las contraseñas cifradas de cada uno de los usuarios.

cat /etc/passwd

```
jhon:x:1000:1000:,,,:/home/jhon:/bin/bash
alice:x:1001:1001:,,,:/home/alice:/bin/bash
bob:x:1002:1002:,,,:/home/bob:/bin/bash
```

sudo cat /etc/shadow

```
jhon:$6$C8CGRdRk2lVfHQj9$T4uM7KXz7r6bsgh6WFdLxC8i7AyHP0jcD3HvrjaTpnQL65Gmb08hnmXAcAW9KfkfiZxYE1wu5kzuPxDAErLCC1:19279:0:99999:7:::
alice:$6$BXbAI869E5KdJzeh$dwBBGc98MNWKgIIz7CFjkGkPBKCZG3SPszuzkje7G7g9S/0LyoMORLPNmEEhAqUthmXe/p40eLJV0lOyMzgVb1:19279:0:99999:7:::
bob:$6$CWSJyscqg0Kdf/En$u0nT//3SX3c7bt/IPugjr5Qu9ullEMqtbMALANIQUaGPAC3LcRKi3qV8vA7TMOFfM07VrPPA7i8RimYqrVboM1:19279:0:99999:7:::
```

Como se puede observar al comienzo de cada contraseña aparece: 6 que se corresponde con el identificador de la función hash utilizada para cifrar el texto plano y que en este caso se corresponde con la función SHA-512, la cual genera una salida de 512 bits.

Para realizar el ataque de fuerza bruta utilizamos la herramienta **John The Ripper** la cual sirve para recuperar contraseñas. Esta herramienta combina varios descifradores de contraseñas en un único paquete, auto detecta los tipos de hash de las contraseñas e incluye un descifrador personalizable.

Antes de ejecutar la herramienta **John,** pasamos a un nuevo fichero las contraseñas de *alice* y *bob* que vamos a intentar descifrar.

```
jhon@LAPTOP-91HTN3Q5:~/GSI/Prac_3$ cat passwords.txt
alice:$6$BXbAI869E5KdJzeh$dwBBGc98MNWKgIIz7CFjkGkPBKCZG3SPszuzkje7G7g9S/0LyoMORLPNmEEhAqUthmXe/p40eLJV0lOyMzgVb1:19279:0:99999:7:::
bob:$6$CWSJyscqg0Kdf/En$u0nT//3SX3c7bt/IPugjr5Qu9ullEMqtbMALANIQUaGPAC3LcRKi3qV8vA7TMOFfM07VrPPA7i8RimYqrVboM1:19279:0:99999:7:::
```

A continuación, ejecutamos el siguiente comando: **sudo john passwords.txt** y obtenemos lo siguiente:

```
jhon@LAPTOP-91HTN3Q5:~/GSI/Prac_3$ sudo john passwords.txt --show
[sudo] password for jhon:
alice:toor:19279:0:99999:7:::
bob:123456:19279:0:99999:7:::

2 password hashes cracked, 0 left
```

Como se puede observar se han descifrado las contraseñas '*toor'* de alice y '*123456'* de bob de forma exitosa.

7. Construya un programa [buildhash] (shell script o como prefiera), que obtenga para todos y cada uno de los ficheros cuyos nombres aparecen (uno por línea) en un fichero de texto de entrada dos hashcodes: uno asociado al propio fichero y otro a sus propiedades. Para cada fichero, se generará una línea que contenga el nombre de fichero, el hashcode del fichero y el hashcode de sus propiedades, separados por ';'.

Para este ejercicio he creado un script en Python con una seria de módulos que he ido comentando. Para realizar el cifrado utilizo la función hash **SHA256**, que como se comentó en el laboratorio, es una de las principales funciones hash que no produce colisiones.

En cuanto al contenido del fichero de entrada, he incluido el nombre de cuatro ficheros de la siguiente manera:
fichero1.text
fichero2.text
fichero3.text
fichero4.text

El script creado es el siguiente:

```python
# Imports
import sys
import hashlib
import os

def main():
    # Nombre del fichero del que vamos a leer
    filename = sys.argv[1]

    fich = open(filename, "r") # Lo abrimos en modo lectura

    # Obtenemos las lineas del fichero
    lines = fich.readlines()

    # Creamos el fichero donde vamos a escribir los hashcodes
    salida = open("ficheroSalida.txt", "w")

    for line in lines:
        # Formateamos cada linea
        line = line.rstrip()

        # Para cifrar el fichero con la funcion sha256 tenemos que utilizar
        # el modulo de python 'hashlib' el cual incluye el algoritmo hash
        # que queremos utilizar
        hashCode = hashlib.sha256(str(line).encode('utf-8')).hexdigest()

        # Ahora codificamos las propiedades del fichero
        # Para ello utilizamos el modulo 'os' de pyton y mediante la sentencia
        # os.stat() obtenemos toda la informacion del fichero en cuestion
        properties = os.stat(line)
        hashProperties = hashlib.sha256(str(properties).encode('utf-8')).hexdigest()

        # Finalmente le damos el formato a cada linea de salida
        output = line + ";" + hashCode + ";" + hashProperties + "\n"

        # Escribimos en el fichero
        salida.write(output)

    salida.close()

if __name__ == "__main__":
    main()
```

Salida obtenida:

```
jhon@LAPTOP-91HTN3Q5:~/GSI/Prac_3$ cat ficheroSalida.txt
fichero1.txt;55442bcdce0a1488588a13cdd4472edb6ebfffb59677e99ba69b81691723b384;2a28f400887705de0ea8e1dcaeba004b7c79d3ef5339df2b59f4389093e89e27
fichero2.txt;19d4ac566ff7116fe9d334f3705f6953a8951f1fe439da79fd0d94b7ff09c931;52bb6a4d95ab4c2218670f826ed45123999900539ff4d4bd17c9ecbb08fe2b9b
fichero3.txt;c35ee60b140e432a95dd2a222b3249a153a70fad94761cf32a72e99377e5aa72;045e9b1de3ee881ba736b9e74f4c0448fe979e61c92d9db8fe92fab328130b32
fichero4.txt;7990b49e0fe0f9d49d9910800f16e36c80dfdf0a64d204c29501a222a8d73cd0;5ce02f5bd2d902c82c958bcb1c3ffe5f10ef4d270a1e51863cf96f33907dc6c2
jhon@LAPTOP-91HTN3Q5:~/GSI/Prac_3$
```

8. Construya un programa [checkhash] (shell script o como prefiera), que tome como entrada el fichero generado por el anterior y compruebe, para cada fichero, si se han producido cambios en el mismo o en sus propiedades, generando una salida en que se muestre cada fichero y se indique si se ha modificado o no.

```python
# Imports
import sys
import hashlib
import os

def main():
    # La entrada es el fichero generado por 'buildhash'
    filename = sys.argv[1]
    fich = open(filename, "r")

    # Obtenemos las lineas del fichero
    lines = fich.readlines()

    for line in lines:
        # Formateamos cada linea
        line = line.rstrip()

        # Procesamos los datos de cada linea
        data = line.split(';')
        name = data[0]           # Obtenemos el nombre de cada fichero
        hashCode = data[1]       # Obtenemos el hash de cada fichero
        hashProperties = data[2] # Obtenemos el hash de las propiedades de cada fichero

        # Ciframos de nuevo el fichero para luego comprobar con el obtenido
        hash1 = hashlib.sha256(str(name).encode('utf-8')).hexdigest()

        proper = os.stat(name)
        hash2 = hashlib.sha256(str(proper).encode('utf-8')).hexdigest()

        # Comprobamos finalmente si se han producido cambios en el fichero o en sus propiedades
        if hash1 != hashCode or hash2 != hashProperties:
            print("El fichero: " + name + " ha sido modificado")
        else:
            print("El fichero: " + name + " NO ha sido modificado")

if __name__ == '__main__':
    main()
```

Si lo lanzamos sin modificar ningún fichero dándole como parámetro el fichero obtenido con *buildhash* obtenemos lo siguiente:

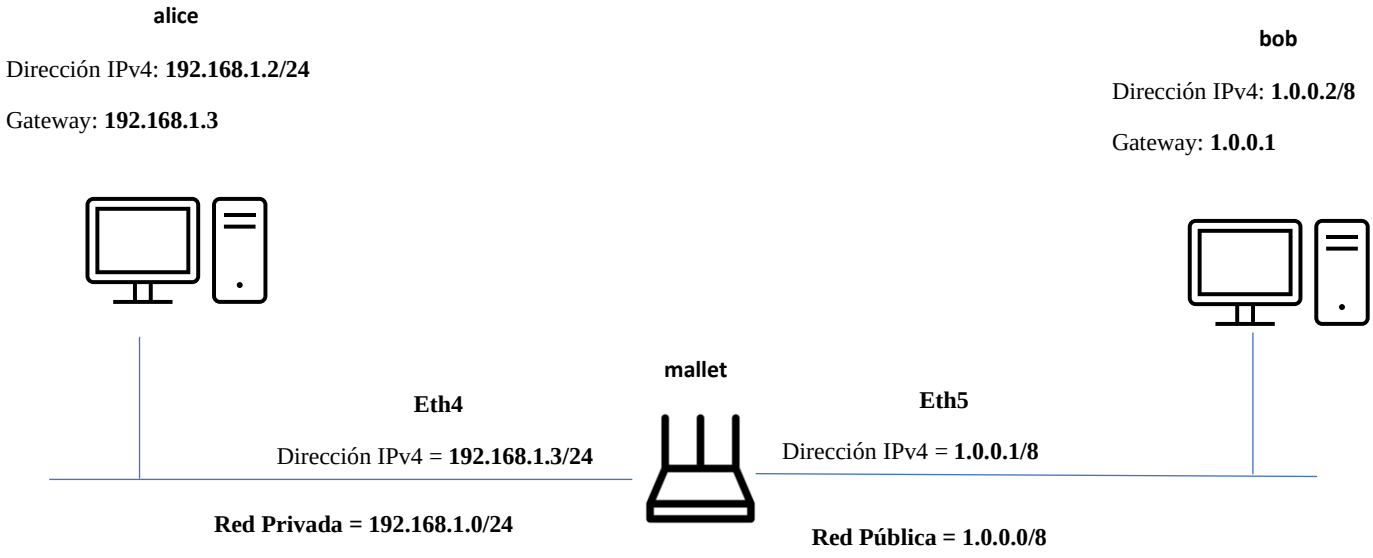
```
jhon@LAPTOP-91HTN3Q5:~/GSI/Prac_3$ python3 checkhash.py ficheroSalida.txt
El fichero:  fichero1.txt NO ha sido modificado
El fichero:  fichero2.txt NO ha sido modificado
El fichero:  fichero3.txt NO ha sido modificado
El fichero:  fichero4.txt NO ha sido modificado
```

Por el contrario, si modificamos, por ejemplo, el fichero 3 obtenemos lo siguiente:

```
jhon@LAPTOP-91HTN3Q5:~/GSI/Prac_3$ echo hola > fichero3.txt
jhon@LAPTOP-91HTN3Q5:~/GSI/Prac_3$ cat fichero3.txt
hola
jhon@LAPTOP-91HTN3Q5:~/GSI/Prac_3$ python3 checkhash.py ficheroSalida.txt
El fichero:  fichero1.txt NO ha sido modificado
El fichero:  fichero2.txt NO ha sido modificado
El fichero:  fichero3.txt ha sido modificado
El fichero:  fichero4.txt NO ha sido modificado
```

Parte 2

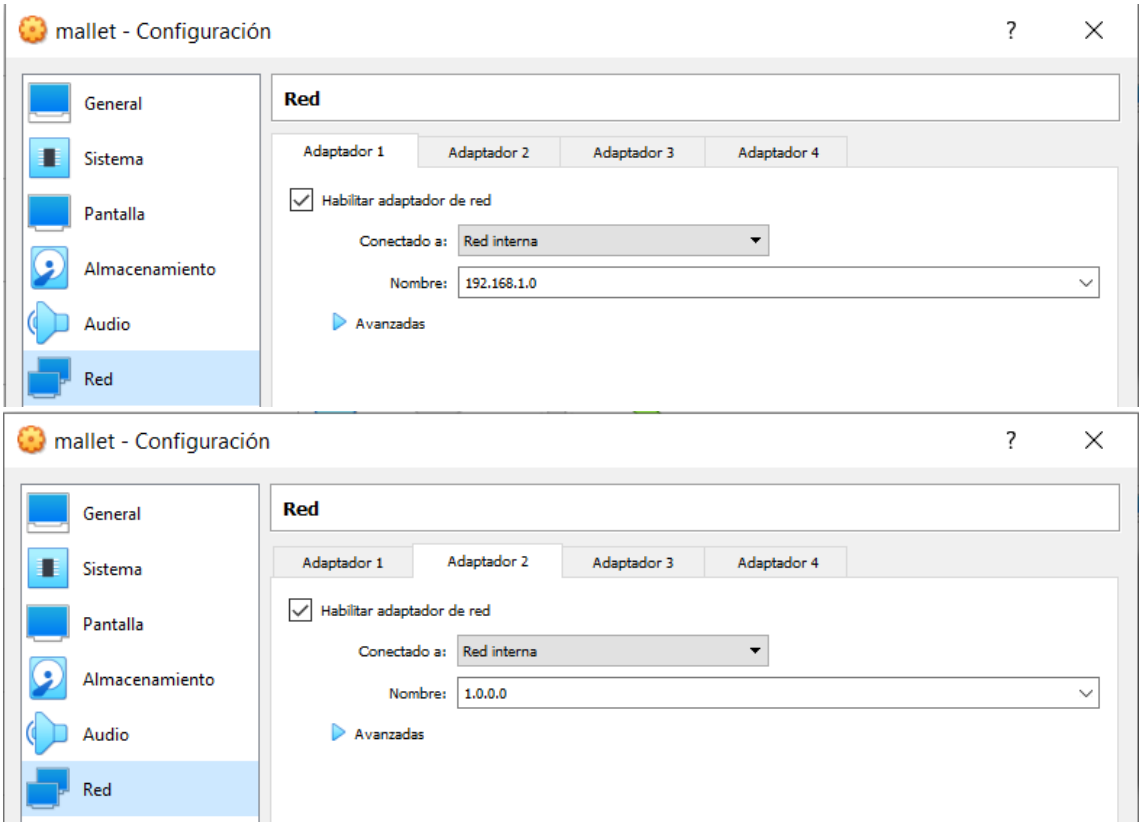
1. Dibujar el diagrama de red lo más detallado posible.



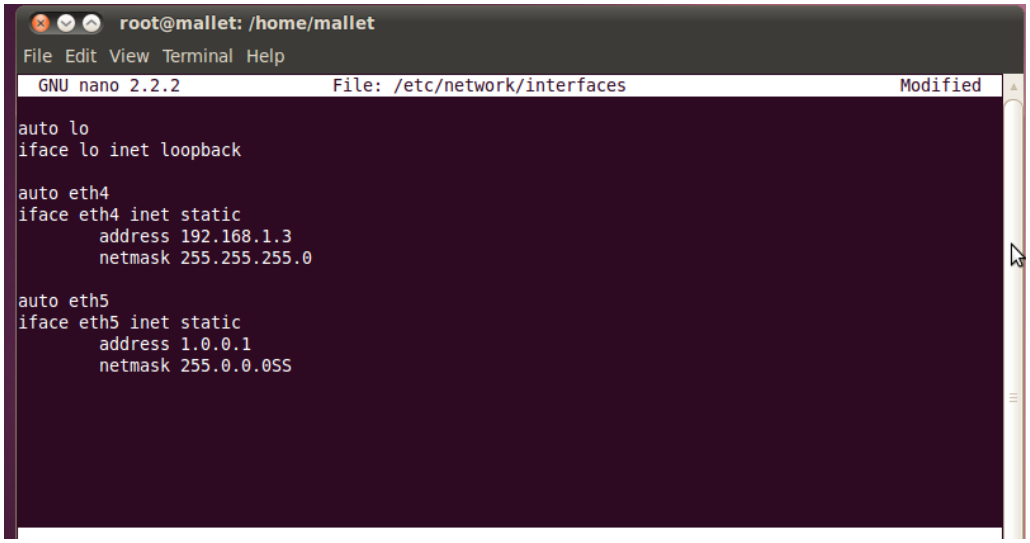
2. Configura la máquina mallet para que actúe como router. Tendrá dos interfaces de red, la eth4 con dirección 192.168.1.3/24 para eth4 en la red interna "Privada", y la eth5 con dirección 1.0.0.1/8 en la red interna "Pública". Deberá tener habilitado el bit de enrutamiento.

Red interna en VirtualBox no despliega ningún tipo de servicio por lo que les podemos dar las direcciones IP que queramos a las distintas interfaces.

Para ello, nos dirigimos a Configuración de la máquina mallet y establecemos las 2 interfaces. Seleccionamos redes internas y les asignamos las direcciones IP indicadas.



Ahora, tenemos que configurar las direcciones IP de las interfaces de red creadas. Para ello debemos modificar el archivo **/etc/network/interfaces**, el cual es un archivo crítico por lo que tenemos que ser superusuarios.



Reiniciamos los servicios de red para que los cambios se apliquen:

```
root@mallet:/home/mallet# sudo /etc/init.d/networking restart
 * Reconfiguring network interfaces...
ssh stop/waiting
ssh start/running, process 1677
255.0.0.0SS: Unknown host
Failed to bring up eth5.
                                                                    [ OK ]
root@mallet:/home/mallet#
```

Comprobamos que se han realizado los cambios

```
root@mallet:/home/mallet# 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 qlen 1000
    link/ether 08:00:27:62:45:e6 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.3/24 brd 192.168.1.255 scope global eth4
    inet6 fe80::a00:27ff:fe62:45e6/64 scope link
       valid_lft forever preferred_lft forever
3: eth5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 08:00:27:1f:2f:f9 brd ff:ff:ff:ff:ff:ff
    inet 1.0.0.1/8 brd 1.255.255.255 scope global eth5
    inet6 fe80::a00:27ff:fe1f:2ff9/64 scope link
       valid_lft forever preferred_lft forever
root@mallet:/home/mallet#
```

Finalmente, realizamos un ping a las interfaces para comprobar que realmente funcionan.

```
root@mallet:/home/mallet# ping 192.168.1.3 -c 5
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=0.012 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=64 time=0.017 ms
64 bytes from 192.168.1.3: icmp_seq=3 ttl=64 time=0.040 ms
64 bytes from 192.168.1.3: icmp_seq=4 ttl=64 time=0.018 ms
64 bytes from 192.168.1.3: icmp_seq=5 ttl=64 time=0.020 ms

--- 192.168.1.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3997ms
rtt min/avg/max/mdev = 0.012/0.021/0.040/0.010 ms
root@mallet:/home/mallet# ping 1.0.0.1 -c 5
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=0.020 ms
64 bytes from 1.0.0.1: icmp_seq=2 ttl=64 time=0.042 ms
64 bytes from 1.0.0.1: icmp_seq=3 ttl=64 time=0.042 ms
64 bytes from 1.0.0.1: icmp_seq=4 ttl=64 time=0.041 ms
64 bytes from 1.0.0.1: icmp_seq=5 ttl=64 time=0.043 ms

--- 1.0.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.020/0.037/0.043/0.011 ms
root@mallet:/home/mallet#
```

Como se puede observar en la imagen, el ping se ha realizado de forma exitosa.

Como último paso, nos queda habilitar el bit de enrutamiento el cual permite a los paquetes pasar a través del router.

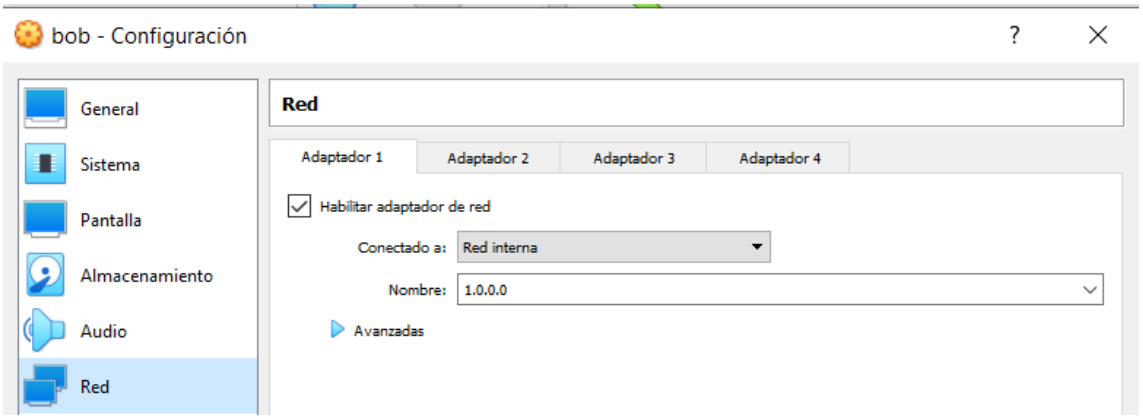
```
root@mallet:/home/mallet# cat /proc/sys/net/ipv4/ip_forward
0
root@mallet:/home/mallet#
```

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

Estos cambios son temporales, cuando reiniciemos la maquina volverá el valor 0.

3. Configura la máquina bob para que esté dentro de la red interna "Pública" y tenga como dirección IP la 1.0.0.2/8, y como puerta de enlace la 1.0.0.1/8.

Configuramos bob estableciendo su dirección IP que tiene que ser la misma que la interfaz eth5 de mallet.



Procedemos a modificar el fichero de configuración de las interfaces de red.

```
  GNU nano 2.0.7         File: /etc/network/interfaces                Modified

# 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
auto eth0
iface eth0 inet static
        address 1.0.0.2
        netmask 255.0.0.0
        gateway 1.0.0.1


allow-hotplug eth1
#iface eth1 inet dhcp
```

Reiniciamos los servicios de red para que los cambios se apliquen y comprobamos que se han efectuado los cambios.

Sin embargo, al intentar reiniciar los servicios de red obtengo el siguiente error, por lo que procedemos a configurar bob utilizando una dirección IP temporal.

```
bob:/home/bob# /etc/init.d/networking restart
Internet Systems Consortium DHCP Client V3.1.1
Copyright 2004-2008 Internet Systems Consortium.
All rights reserved.
For info, please visit http://www.isc.org/sw/dhcp/

SIOCSIFADDR: No such device
eth1: ERROR while getting interface flags: No such device
eth1: ERROR while getting interface flags: No such device
Bind socket to interface: No such device
bob:/home/bob#
```

Para asignar la IP de forma temporal utilizo el siguiente comando: **ifconfig eth0 1.0.0.2**

```
bob:/home/bob# ifconfig eth0 1.0.0.2
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 08:00:27:63:40:98 brd ff:ff:ff:ff:ff:ff
    inet 1.0.0.2/8 brd 1.255.255.255 scope global eth0
    inet6 fe80::a00:27ff:fe63:4098/64 scope link
       valid_lft forever preferred_lft forever
3: sit0: <NOARP> mtu 1480 qdisc noop state DOWN
    link/sit 0.0.0.0 brd 0.0.0.0
bob:/home/bob# S
```

Ahora debemos asignarle el Gateway utilizando: **route add default gw 1.0.0.1 eth0**. Para comprobar las rutas que existen usamos **route -n**.

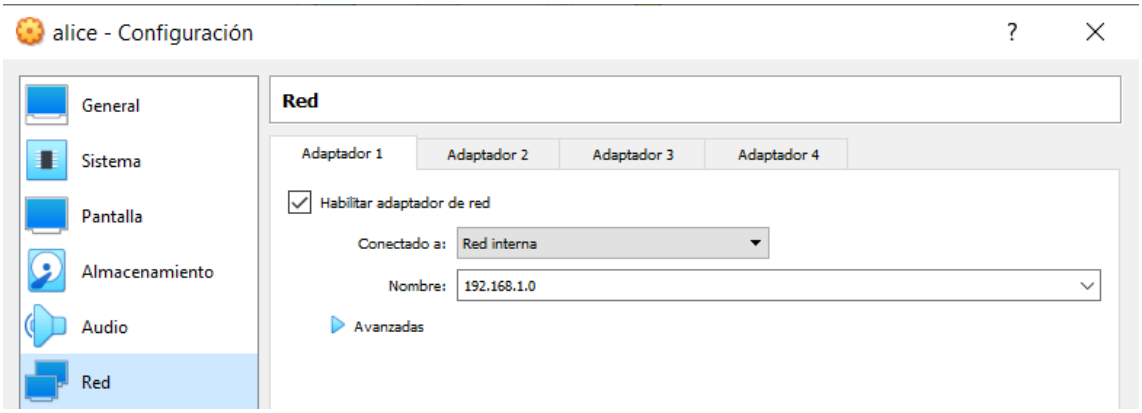
```
bob:/home/bob# route add default gw 1.0.0.1 eth0
bob:/home/bob# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
1.0.0.0         0.0.0.0         255.0.0.0       U     0      0        0 eth0
0.0.0.0         1.0.0.1         0.0.0.0         UG    0      0        0 eth0
bob:/home/bob#
```

Finalmente, realizamos un ping a la dirección IP para comprobar que realmente funciona:

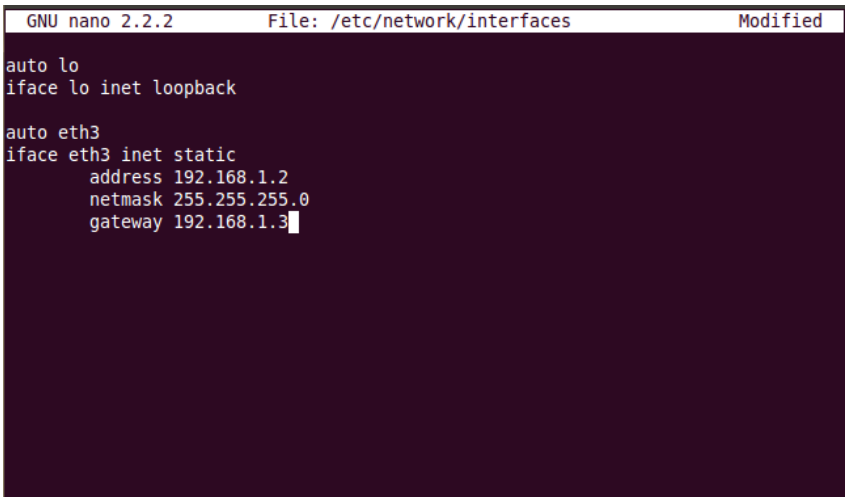
```
bob:/home/bob# ping 1.0.0.2 -c 5
PING 1.0.0.2 (1.0.0.2) 56(84) bytes of data.
64 bytes from 1.0.0.2: icmp_seq=1 ttl=64 time=0.011 ms
64 bytes from 1.0.0.2: icmp_seq=2 ttl=64 time=0.191 ms
64 bytes from 1.0.0.2: icmp_seq=3 ttl=64 time=0.194 ms
64 bytes from 1.0.0.2: icmp_seq=4 ttl=64 time=0.253 ms
64 bytes from 1.0.0.2: icmp_seq=5 ttl=64 time=0.077 ms

--- 1.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4004ms
rtt min/avg/max/mdev = 0.011/0.145/0.253/0.088 ms
bob:/home/bob#
```

4. Configura la máquina Alice para que esté dentro de la red interna "Privada" y tenga como dirección IP la 192.168.1.2, y como puerta de enlace la 192.168.1.3/24.

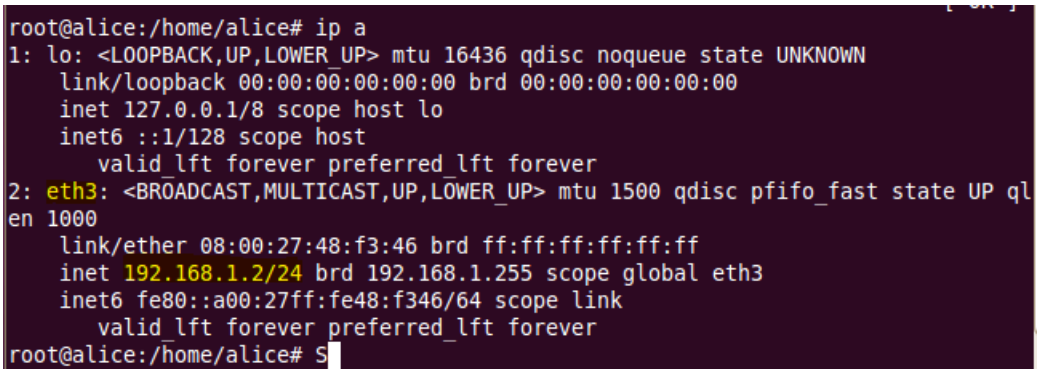
Configuramos alice estableciendo su dirección IP que tiene que ser la misma que la interfaz eth4 de mallet.



Modificamos el fichero de configuración de las interfaces de red.



Reiniciamos los servicios de red para que los cambios se apliquen y comprobamos que se han efectuado los cambios.



-Realizamos un ping a la interfaz para comprobar que realmente funciona.

```
root@alice:/home/alice# ping 192.168.1.2 -c 10
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=64 time=0.060 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.042 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=64 time=0.042 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=64 time=0.022 ms
64 bytes from 192.168.1.2: icmp_seq=5 ttl=64 time=0.042 ms
64 bytes from 192.168.1.2: icmp_seq=6 ttl=64 time=0.015 ms
64 bytes from 192.168.1.2: icmp_seq=7 ttl=64 time=0.011 ms
64 bytes from 192.168.1.2: icmp_seq=8 ttl=64 time=0.049 ms
64 bytes from 192.168.1.2: icmp_seq=9 ttl=64 time=0.014 ms
64 bytes from 192.168.1.2: icmp_seq=10 ttl=64 time=0.027 ms

--- 192.168.1.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8999ms
rtt min/avg/max/mdev = 0.011/0.032/0.060/0.016 ms
You have new mail in /var/mail/root
root@alice:/home/alice#
```

5. Realiza pruebas a nivel de red para comprobar que las máquinas se comunican. ¿Qué ocurre con el valor del *ttl* cunado un paquete atraviesa un router?

Para comprobar que las máquinas realmente se comunican, procedemos a realizar *pings* entre ellas. Comenzamos realizando pings desde bob al resto de máquinas.

1. Ping de bob a la interfaz eth5 (1.0.0.1) y la interfaz eth4 (192.168.1.3) de mallet.

```
bob:/home/bob# ping 1.0.0.1 -c 5
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=7.25 ms
64 bytes from 1.0.0.1: icmp_seq=2 ttl=64 time=1.17 ms
64 bytes from 1.0.0.1: icmp_seq=3 ttl=64 time=0.722 ms
64 bytes from 1.0.0.1: icmp_seq=4 ttl=64 time=1.20 ms
64 bytes from 1.0.0.1: icmp_seq=5 ttl=64 time=1.01 ms

--- 1.0.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4011ms
rtt min/avg/max/mdev = 0.722/2.274/7.252/2.494 ms
bob:/home/bob# ping 192.168.1.3 -c 5
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=0.443 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=64 time=0.964 ms
64 bytes from 192.168.1.3: icmp_seq=3 ttl=64 time=1.31 ms
64 bytes from 192.168.1.3: icmp_seq=4 ttl=64 time=0.667 ms
64 bytes from 192.168.1.3: icmp_seq=5 ttl=64 time=1.07 ms

--- 192.168.1.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4010ms
rtt min/avg/max/mdev = 0.443/0.894/1.318/0.307 ms
bob:/home/bob# _
```

Como se puede ver en la imagen, bob se comunica correctamente con mallet. Podemos ver también que el valor del **ttl**, es decir, el tiempo de vida del paquete tiene un valor de **64**.

2. Ahora realizamos un ping desde bob hacia alice.

```
bob:/home/bob# ping 192.168.1.2 -c 5
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=0.927 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=63 time=1.48 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=63 time=0.942 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=63 time=1.21 ms
64 bytes from 192.168.1.2: icmp_seq=5 ttl=63 time=0.824 ms

--- 192.168.1.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4011ms
rtt min/avg/max/mdev = 0.824/1.077/1.480/0.240 ms
bob:/home/bob#
```

Como podemos ver, el ping es de nuevo exitoso. Pero, si nos fijamos en el valor del ttl, vemos cómo este ha reducido su valor en una unidad. Esto es debido al **routing** de la máquina mallet.

Ahora realizamos los ping desde alice al resto.

1. Ping de alice a la interfaz eth4 (192.168.1.3) y la interfaz eth5 (1.0.0.1) de mallet.

```
root@alice:/home/alice# ping 192.168.1.3 -c 5
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=0.376 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=64 time=0.418 ms
64 bytes from 192.168.1.3: icmp_seq=3 ttl=64 time=0.621 ms
64 bytes from 192.168.1.3: icmp_seq=4 ttl=64 time=0.514 ms
64 bytes from 192.168.1.3: icmp_seq=5 ttl=64 time=0.460 ms

--- 192.168.1.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.376/0.477/0.621/0.089 ms
root@alice:/home/alice# ping 1.0.0.1 -c 5
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=0.557 ms
64 bytes from 1.0.0.1: icmp_seq=2 ttl=64 time=0.485 ms
64 bytes from 1.0.0.1: icmp_seq=3 ttl=64 time=0.478 ms
64 bytes from 1.0.0.1: icmp_seq=4 ttl=64 time=0.493 ms
64 bytes from 1.0.0.1: icmp_seq=5 ttl=64 time=0.448 ms

--- 1.0.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.448/0.492/0.557/0.038 ms
root@alice:/home/alice#
```

Como se puede ver en la imagen, alice se comunica correctamente con mallet. Podemos ver también que el valor del **ttl** tiene un valor de **64**.

2. Ahora realizamos un ping desde alice hacia bob.

```
root@alice:/home/alice# ping 1.0.0.2 -c 5
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=0.939 ms
64 bytes from 1.0.0.2: icmp_seq=2 ttl=63 time=1.00 ms
64 bytes from 1.0.0.2: icmp_seq=3 ttl=63 time=0.864 ms
64 bytes from 1.0.0.2: icmp_seq=4 ttl=63 time=0.762 ms
64 bytes from 1.0.0.2: icmp_seq=5 ttl=63 time=0.891 ms

--- 1.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 0.762/0.893/1.009/0.081 ms
root@alice:/home/alice#
```

Como ocurría previamente, el valor del ttl vuelve a disminuir por el **routing** de mallet.

En conclusión, cuando el paquete tiene que atravesar mallet para llegar a la máquina destino, su valor de ttl se verá disminuido en una unidad debido al routing de mallet. Por otra parte, si el paquete no necesita atravesar mallet, es decir, el destino del paquete es la propia máquina mallet, el valor del ttl permanece igual.

6. Indica las diferencias entre las reglas de tipo INPUT, OUTPUT y FORWARD en iptables.

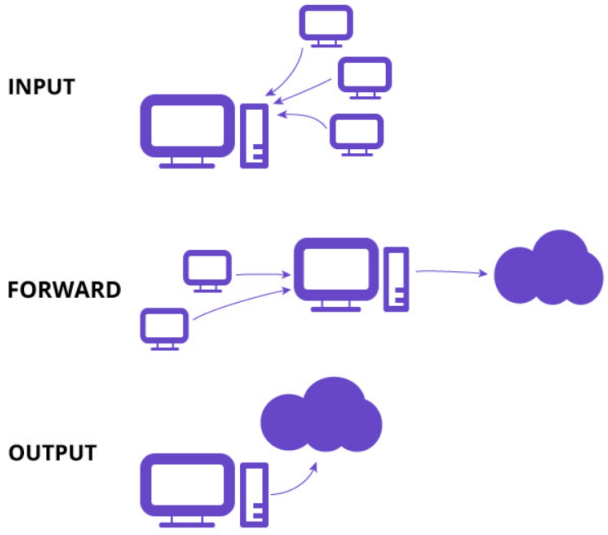
IPtables es un sistema de firewall vinculado al kernel de Linux que se ha extendido enormemente a partir del kernel 2.4 de este sistema operativo.

INPUT,OUTPUT y FORWARD son los tres tipos de reglas de filtrado. Para los paquetes que van a la propia máquina se aplican las reglas INPUT y OUTPUT, y para filtrar paquetes que van a otras redes o máquinas se aplican simplemente reglas FORWARD.

- **INPUT**: se utiliza para filtrar aquellos paquetes que vienen hacia nuestra máquina (solo entra, pero no sale).

- **OUTPUT**: se utiliza para filtrar paquetes generados por nuestra máquina. Ejemplo, estamos en mallet y realizamos un ping hacia alice.

- **FORWARD**: se utiliza para filtrar aquellos paquetes que llegan a nuestra máquina, pero no son para nosotros, es decir, llegan a nuestra máquina para ser reencaminados (input + output).



7. Configura una regla en el firewall del router para que sólo se puedan realizar peticiones de tipo ICMP(8) desde la red interna.

Para configurar la regla pedida debemos ejecutar el siguiente comando:
iptables -A FORWARD -s 1.0.0.1/8 -d 192.168.1.0/24 -p icmp –icmp-type 8 -j DROP.

- *-A FORWARD*: añade la regla a la cadena de reglas.
- *-s 1.0.0.1/8*: origen de la especificación.
- *-d 192.168.1.0/24*: destino de la especificación.
- *-p icmp*: protocolo de conexión utilizado.
- *--icmp-type 8*: indicación de que sólo se pueden realizar peticiones ICMP(8).
- *-j DROP*: salto al objetivo especificado: ignorar silenciosamente el paquete y dejar de procesar las reglas de esta cadena.

Procedemos a comprobar que la regla ha sido creada correctamente. Para ellos listamos las reglas de filtrado actuales mediante: **iptables -L**.

```
root@mallet:/home/mallet# iptables -A FORWARD -s 1.0.0.1/8 -d 192.168.1.0/24 -p
icmp --icmp-type 8 -j DROP
root@mallet:/home/mallet# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
DROP       icmp --  1.0.0.0/8           192.168.1.0/24      icmp echo-request

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@mallet:/home/mallet#
```

Como consecuencia de la regla creada, no podemos realizar una conexión de bob con alice ya que solamente se permiten peticiones ICMP(8) desde la red interna. Si realizamos un ping desde bob hacia alice, este no se debería completar dado que el **firewall** denegará el paso de paquetes.

```
bob:/home/bob# ping 192.168.1.2 -c 5
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.

--- 192.168.1.2 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4011ms

bob:/home/bob#
```

Efectivamente, como se puede comprobar en la imagen, no se ha recibido ningún paquete en alice, todos se han perdido.

Para borrar la regla debemos ejecutar el siguiente comando:
iptables -D FORWARD -s 1.0.0.1/8 -d 192.168.1.0/24 -p icmp –icmp-type 8 -j DROP.

```
root@mallet:/home/mallet# iptables -D FORWARD -s 1.0.0.1/8 -d 192.168.1.0/24 -p
icmp --icmp-type 8 -j DROP
root@mallet:/home/mallet# iptables -L
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
root@mallet:/home/mallet#
```

Finalmente, una vez borrada la regla, realizamos un ping de confirmación desde bob hacia alice.

```
bob:/home/bob# ping 192.168.1.2 -c 5
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=7.17 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=63 time=1.02 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=63 time=1.70 ms
64 bytes from 192.168.1.2: icmp_seq=4 ttl=63 time=1.09 ms
64 bytes from 192.168.1.2: icmp_seq=5 ttl=63 time=2.16 ms

--- 192.168.1.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4011ms
rtt min/avg/max/mdev = 1.029/2.635/7.179/2.310 ms
bob:/home/bob#
```

8. Configura una regla en el firewall para que sólo la máquina con IP 192.168.1.3 sea quién pueda establecer una conexión por *ssh* a la máquina 192.168.1.2.

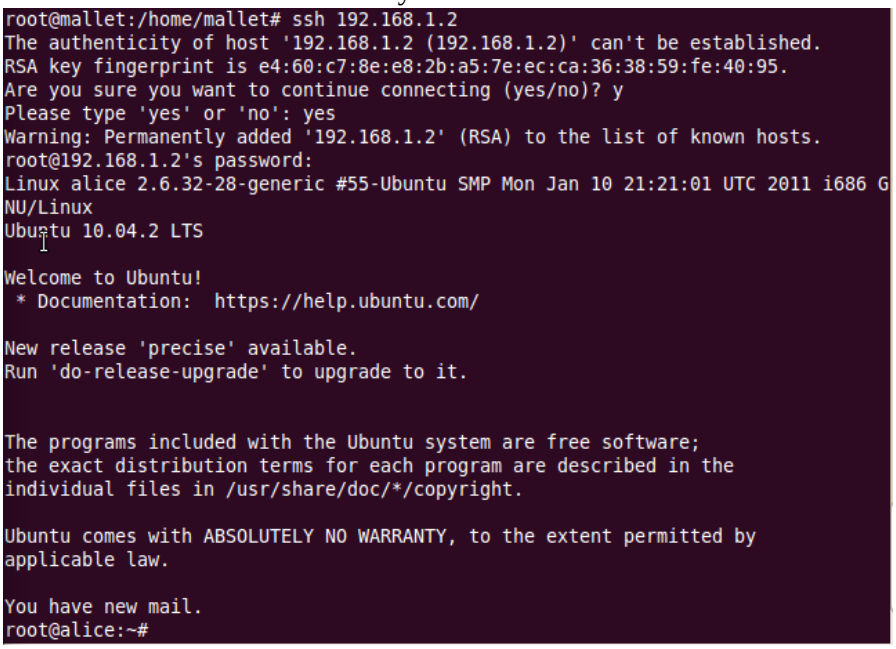
Esta nueva regla solamente permite una conexión ssh desde mallet (192.168.1.3) hacia alice (192.168.1.2). Para ello, ejecutamos el siguiente comando:
iptables -A FORWARD -s 0.0.0.0/0 -p tcp –dport 22 -j DROP.

- *-A FORWARD*: añade la regla a la cadena de reglas.
- *-s 0.0.0.0/0*: origen de la especificación.
- *-p tcp*: protocolo de conexión utilizado.
- *--dport 22*: indica el puerto(s) de destino requerido(s) para esta regla. Utilizamos el puerto 22 debido a que es el puerto TCP asignado al protocolo SSH.
- *-j DROP*: salto al objetivo especificado: ignorar silenciosamente el paquete y dejar de procesar las reglas de esta cadena.

```
root@mallet:/home/mallet# iptables -A FORWARD -s 0.0.0.0/0 -p tcp --dport 22 -j
DROP
root@mallet:/home/mallet# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  anywhere             anywhere             tcp dpt:ssh

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@mallet:/home/mallet#
```

Mediante esta nueva regla, podemos realizar conexiones ssh desde mallet hacia alice y hacia bob, pero no desde bob hacia alice o viceversa. Esto es debido a que para realizar dicha conexión deberíamos atravesar el router, pero esta nueva regla añadida al firewall lo impedirá.

1. Verificamos la conexión ssh entre mallet y alice.

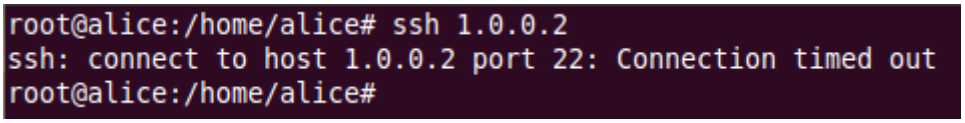


Conexión establecida con éxito.

2. Verificamos la conexión ssh entre mallet y bob.

De nuevo, se establece la conexión de forma exitosa.

Sin embargo, si intentamos realizar una conexión ssh entre alice y bob como comentamos anteriormente, obtendremos lo siguiente:



```
bob:/home/bob# ssh 192.168.1.2
ssh: connect to host 192.168.1.2 port 22: Connection timed out
bob:/home/bob#
```

Para borrar la regla debemos ejecutar el siguiente comando:
iptables -D FORWARD -s 0.0.0.0/0 -p tcp –dport 22 -j DROP.

```
root@mallet:/home/mallet# iptables -D FORWARD -s 0.0.0.0/0 -p tcp --dport 22 -j
DROP
root@mallet:/home/mallet# iptables -L
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
root@mallet:/home/mallet#
```

Finalmente, una vez borrada la regla, realizamos una conexión shh de confirmación desde alice hacia bob.

```
root@alice:/home/alice# ssh 1.0.0.2
ssh: connect to host 1.0.0.2 port 22: Connection timed out
root@alice:/home/alice# ssh 1.0.0.2
The authenticity of host '1.0.0.2 (1.0.0.2)' can't be established.
RSA key fingerprint is fb:9a:ce:41:95:d0:60:40:35:62:35:ae:e2:99:58:c1.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '1.0.0.2' (RSA) to the list of known hosts.
root@1.0.0.2'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.
Last login: Tue Nov  1 20:12:29 2022 from 1.0.0.1
bob:~#
```

Y una conexión ssh desde bob hacia alice.

```
bob:/home/bob# ssh 192.168.1.2
The authenticity of host '192.168.1.2 (192.168.1.2)' can't be established.
RSA key fingerprint is e4:60:c7:8e:e8:2b:a5:7e:ec:ca:36:38:59:fe:40:95.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.2' (RSA) to the list of known hosts.
root@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 G
NU/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.

You have new mail.
Last login: Tue Nov  1 20:09:53 2022 from mallet
root@alice:~# _
```