

## Práctica 3. Hashing, Cifrado y Certificados

### 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.

Antes de comenzar con las propias preguntas del apartado voy a comentar brevemente información sobre las funciones de HASH.

→ Son funciones criptográficas

→ Características:

- Son funciones de una sola vía  $f(\text{texto}) \rightarrow \text{resumen/código hash}$ , es decir, desde el punto de vista computacional (tiempo y procesador) es muy fácil obtener dado el texto claro obtener la función de hash, pero dado un resumen/código hash es muy complicado obtener el texto que ha generado ese resumen.
- La longitud del resumen es la misma independientemente del tamaño de la entrada. Aunque la entrada sea muy grande la salida siempre van a ser 32 caracteres en hexadecimal (md5) o 40 caracteres en hexadecimal (sha1).
- ¿Qué garantizan? La integridad de la información, es decir, que la información no se pueda alterar.

La manera de utilizar md5sum y sha1sum es la misma, es decir el comando es exactamente igual para ambas formas de creación de hashcodes, únicamente cambia el nombre de la función que utilizas. Además, existen dos formas para codificar, la primera es pasando el texto plano que vamos a querer que md5sum o sha1sum codifique mediante una tubería.

```
mallet@mallet:~$ echo prueba | md5sum
e231d7bb3debf6497cac29db4f258295 -
mallet@mallet:~$ echo prueba | sha1sum
5c7b760a8c20028408fd2f48a6486f6d0b455804 -
```

La otra posible opción es utilizando un archivo ya creado previamente, pasándole a md5sum o sha1sum el nombre de este. En este caso voy a utilizar testttttt.pdf que ya estaba creado previamente en la máquina virtual.

```
mallet@mallet:~$ md5sum testttttt.pdf
76379ef9ddf935d80b397646ece7f0bc testttttt.pdf
mallet@mallet:~$ sha1sum testttttt.pdf
bcf3202e577284717465cbe7d6ebbd76e7223dc0 testttttt.pdf
```

En cuanto a las debilidades conocidas de estas dos formas de crear hashcodes, ambas presentan debilidades de colisión, lo cual implica que, para dos archivos claves, palabras etc. distintos, puede generar el mismo código hash. Como se ve en esta tabla, también md5 utiliza 128 bits mientras que sha1 utiliza 160 bits, lo que influye en el número de caracteres 32 y 40 respectivamente.

HASH	Tamaño salida	Colisiones
MD5	32 caracteres hexadecimal 128 bits	SI
SHA1	40 caracteres hexadecimal 160 bits	SI

**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.**

Para este apartado voy a tomar las cadenas: “cadena 1” y “cadena1” que únicamente difieren en un espacio.

Primero utilizando md5 para esas dos cadenas vemos que el código hash creado es diferente puesto que las palabras a codificar eran distintas.

```
mallet@mallet:~$ echo cadena1 | md5sum
b4631ac5b8562148d02738e662f25d81 -
mallet@mallet:~$ echo cadena 1 | md5sum
9ca8108cadf439f62c1c1d59eaba206c -
```

Luego igual, pero utilizando sha1, también podemos ver que la codificación es distinta.

```
mallet@mallet:~$ echo cadena1 | shasum
171956711e5d0e83138a9c88dba0d3c43506a1a0 -
mallet@mallet:~$ echo cadena 1 | shasum
9ba0cb02bfc3cde7ea66ce47107a86f1174ed20d -
```

Y comparando ambas podemos ver, como ya hemos comentado en la tabla del apartado anterior que md5 codifica devolviendo una cadena de 32 caracteres en hexadecimal, 128 bits. Mientras que sha1 devuelve una cadena de 40 caracteres en hexadecimal, 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.**

He creado un archivo de texto plano llamado claves.txt, cuyo contenido son 3 claves bastante comunes, sobre este fichero realizaremos ambas herramientas y después lo modificaremos levemente para ver si la salida de las herramientas sigue siendo la misma o ha cambiado.

Primero el contenido del archivo claves.txt:

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

Y ahora ejecuto md5 sobre el fichero claves.txt y después sobre claves.txt añadiendo un retorno de carro. En la flecha amarilla modifico el contenido del fichero, añadiendo el retorno de carro “\n”.

```
mallet@mallet:~$ md5sum claves.txt
840ddb0e48ace8be70745da81cda04c4  claves.txt
mallet@mallet:~$ nano claves.txt ←
mallet@mallet:~$ md5sum claves.txt
db406c38125295f9762a1c1e5c6c1b9b  claves.txt
```

Y ahora exactamente lo mismo, pero con sha1, primero la codificación con el fichero original y después levemente modificado.

```
mallet@mallet:~$ shasum claves.txt
b2042edf7fe9061da428add79a3246945e6ab6db  claves.txt
mallet@mallet:~$ nano claves.txt ←
mallet@mallet:~$ shasum claves.txt
26932c7d44b840313694a334abaefd79e69ece29  claves.txt
```

Como se puede ver tanto para sha1 como para md5 los códigos hash generados son distintos, eso quiere decir que el archivo ha sido modificado, cosa que hemos hecho.

La diferencia que apreciamos entre sha1 y md5 como ya hemos comentado es el número de caracteres en hexadecimal que obtenemos en la salida, 40 y 32, es decir 160 y 128 bits respectivamente.

#### **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.**

Como no tengo la herramienta hashid, he intentado instalar en mallet pero no he podido. Por tanto, como en el punto 6 nos pide instalar un sistema Ubuntu 20.04.2 LTS he decidido instalarlo ahora y realizar este apartado ya en este sistema.

Para instalar la herramienta en esta nueva máquina he introducido el comando “*sudo apt install hashid*”. Una vez instalada la herramienta voy a coger los códigos hash creados previamente y con el comando “*hashid código/resumen*”, va a analizar ese

código hash que hemos puesto y la herramienta nos va a decir con que posibles funciones de hash ha sido creado ese código.

Por ejemplo, para el punto anterior utilizamos hashid y le ponemos como argumento el código hash generado por md5 en el apartado anterior y vemos que comienza a analizar:

```
carlos@Ubuntu20:~$ hashid 840ddb0e48ace8be70745da81cda04c4
Analyzing '840ddb0e48ace8be70745da81cda04c4'
[+] 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
```

Como se puede ver, ha localizado que dicho código hash, fue creado con md5, pero también pudo ser creado con otras funciones hash.

Ahora lo mismo, pero cogiendo el código hash generado por sha1.

```
carlos@Ubuntu20:~$ hashid b2042edf7fe9061da428add79a3246945e6ad6db
Analyzing 'b2042edf7fe9061da428add79a3246945e6ad6db'
[+] SHA-1
[+] Double SHA-1
[+] RIPEMD-160
[+] Haval-160
[+] Tiger-160
[+] HAS-160
[+] LinkedIn
[+] Skein-256(160)
[+] Skein-512(160)
```

De nuevo identifica que una de las posibles funciones creadoras de dicho código hash es sha1.

**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?**

```
mallet@mallet:~$ ls -l claves.txt | md5sum
064fbf2790d4541af6bfea6de93954c1 -
mallet@mallet:~$ chmod 777 claves.txt
mallet@mallet:~$ ls -l claves.txt | md5sum
55e53098df569a1e5a26e914593b6274 -
mallet@mallet:~$ ls -l claves.txt | shasum
d19c9eachb76ae087c684f39a181e8bf740fc5d0f -
mallet@mallet:~$ chmod 644 claves.txt
mallet@mallet:~$ ls -l claves.txt | shasum
ee485d9ab511c6b26e5df66d2b36f0058c41725d -
```

Primero realizamos un hasheo con md5 de las propiedades del fichero `claves.txt` y vemos que obtenemos un código hash. Después modificamos los permisos del fichero con `chmod` y realizamos de nuevo el hasheo y ahora vemos que el código hash generado es distinto al anterior.

Y lo mismo podemos observar con la otra función hash (`sha1`). La conclusión a la que llego es que modificar cualquier propiedad del fichero hace que el código hash cambie, esto puede ser de gran ayuda para ver si alguien ha modificado cualquiera de las propiedades de este con intención de manipularlo.

**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?**

Ahora utilizo el mismo sistema que ya he montado en el apartado 4 para realizar el ejercicio con `hashid`.

Para crear un usuario en Ubuntu el comando es `"sudo useradd nombreusuario"` y para asignar a ese nuevo usuario una contraseña el comando es `"sudo passwd nombreusuario"`. Primero creo el usuario `root` (que ya está creado en el sistema) y le asigno la contraseña `"toor"`:

```
carlos@Ubuntu20:~$ sudo useradd root
useradd: el usuario «root» ya existe
carlos@Ubuntu20:~$ sudo passwd root
Nueva contraseña:
Vuelva a escribir la nueva contraseña:
passwd: contraseña actualizada correctamente
```

Y lo mismo para el segundo usuario `"admin"`, como este ya no estaba creado previamente el mensaje de aviso anterior `"useradd: el usuario <<root>> ya existe"` no lo muestra, y le asigno la contraseña `"123456"`:

```
carlos@Ubuntu20:~$ sudo useradd admin
carlos@Ubuntu20:~$ sudo passwd admin
Nueva contraseña:
Vuelva a escribir la nueva contraseña:
passwd: contraseña actualizada correctamente
```

Como hemos visto en clase, el fichero `/etc/passwd` contiene los usuarios del sistema y el fichero `/etc/shadow` las contraseñas de cada uno de los usuarios, pero a este fichero solo se puede acceder con permisos de superusuario o perteneciendo al grupo shadow.

John The Ripper es una herramienta que crackea por fuerza bruta las contraseñas, por tanto, para ver si John The Ripper consigue crackear las contraseñas de los usuarios que acabamos de crear, voy a pasarle un archivo que contenga las contraseñas hasheadas.

Para conocer cómo se almacenan las contraseñas en un sistema ejecuto el comando `"sudo cat /etc/shadow"` y elijo de ahí la de root y admin ya que son los dos usuarios que acabo de crear. Ambas líneas las introduzco en un fichero para luego pasárselo a John The Ripper para que las crackee.

Creo el fichero `contrasenas.txt` que contiene la salida obtenida para los usuarios root y admin después de hacer `"sudo cat /etc/shadow"`

```
carlos@Ubuntu20:~$ cat contrasenas.txt
root:$6$VrfXOTm1yXZOLIHx$CNHyI5bB/3ykL6E1RTpdIHotoqk4bYcab1KNFAiex0uILYmSzd/eGG
STIIIE9MP10MXgnXfsm7vUzwOQ0JbK1q/:19279:0:99999:7:::

admin:$6$YEd9WrkwBFoaqNlx$Q4.02h.vNGR7awE/59WMA6emLvH2dQRAAZZftKqlaGcRBd8idwXIj
sSeFjHRCJ9upFBVUmGJwqN0wSOSAZshK1:19279:0:99999:7:::
```

Antes de realizar el análisis con John The Ripper, vamos a ver que función de hash se utiliza para cifrar las contraseñas en el archivo `/etc/shadow`. En clase dijimos que `$6$` se correspondía al cifrado utilizando SHA-512 pero he buscado información sobre el cifrado de contraseñas (<https://blog.elhacker.net/2022/02/icheros-etc-passwd-shadow-y-group.html>), donde aquí podemos los distintos `$id` que se utilizan en el cifrado de estas, y corroborar que `$6$` se corresponde a SHA-512:

```
$1$ - MD5
$2a$ - Blowfish
$2y$ - Eksblowfish
$5$ - SHA-256
$6$ - SHA-512
```

También utilizando la herramienta `hashid` utilizada en el apartado 4, donde le pasas como argumento el código hash entre comillas simples, y obtienes que la función hash que lo ha codificado es sha512. Como se puede ver en esta captura.

```
carlos@Ubuntu20:~$ hashid '$6$VrfXOTm1yXZOLIHx$CNHyI5bB/3ykL6E1RTpdIHotoqk4bYcab1KNFAiex0uILYmSzd/eGGSTIIIE9MP10MXgnXfsm7vUzwOQ0JbK1q/'
Analyzing '$6$VrfXOTm1yXZOLIHx$CNHyI5bB/3ykL6E1RTpdIHotoqk4bYcab1KNFAiex0uILYmSzd/eGGSTIIIE9MP10MXgnXfsm7vUzwOQ0JbK1q/'
[+] SHA-512 Crypt
```

Después compruebo si está instalado John The Ripper y no lo está, por tanto, “*sudo apt install john*”. Ahora con el comando “john contraseña.txt” John The Ripper comenzará a crackear las contraseñas y la salida que obtenemos es:

```
carlos@Ubuntu20:~$ john contraseñas.txt
Created directory: /home/carlos/.john
Loaded 2 password hashes with 2 different salts (crypt, generic crypt(3) [?/64]
)
Press 'q' or Ctrl-C to abort, almost any other key for status
toor          (root)
123456        (admin)
2g 0:00:00:05 100% 2/3 0.3355g/s 523.6p/s 523.8c/s 523.8C/s 123456..pepper
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Como podemos observar la herramienta ha conseguido crackear ambas contraseñas y nos dice que la contraseña para el usuario root es toor y para el usuario admin es 123456

**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 ;'**

He decidido construir el código en Python, voy a dejar una captura del código tanto en este apartado como en el siguiente pero luego los adjunto en la entrega.

La justificación por la cual **utilizo sha256** en vez de sha1 o md5, es porque como ya comentamos en clase realizando la tabla de la parte inferior, sha256 no tiene problema de colisiones, mientras que md5 y sha1 si, y utilizo sha256 en vez de sha512 porque utiliza la mitad de los bits en el hasheo y también menos caracteres hexadecimales.

HASH	Tamaño salida	Colisiones
MD5	32 caracteres hexadecimal 128 bits	SI
SHA1	40 caracteres hexadecimal 160 bits	SI
SHA256	64 caracteres hexadecimal 256 bits	NO
SHA512	128 caracteres hexadecimal 512 bits	NO

Captura del código creado:

```
import sys, hashlib, os

# 1º Argumento --> Archivo del que vamos a leer las líneas a encriptar
# 2º Argumento --> Fichero de salida

entrada = sys.argv[1]
salida = sys.argv[2]

# Fichero de entrada en modo lectura
with open(entrada, 'r') as fich:

    # Fichero de salida en modo escritura
    ficheroSalida = open(salida, 'w')

    # Leo línea a línea del fichero y sustituyo los retornos de carro por ''
    for ruta in fich:
        ruta = ruta.replace('\n', '')

        # Como he indicado en el informe utilizo SHA-256 y el porque
        hash = hashlib.sha256()

        # Ahora leemos la ruta en binario
        with open(ruta, 'rb') as binfile:
            texto = 1
            while texto:
                # Vamos leyendo
                texto = binfile.read(1024)
                hash.update(texto)

                # Genero el código hash de la ruta
                textoHash = hash.hexdigest()

        # Obtengo las propiedades
        prop = os.stat(ruta)
        prop = repr(prop).encode()

        # Código hash generado para las propiedades
        propHash = hash.hexdigest()

        # Escribo en el fichero de salida la ruta, el hash de la ruta, y hash de las propiedades
        ficheroSalida.write(ruta+';'+textoHash+';'+propHash+'\n')

# Cierro el fichero
ficheroSalida.close()
```

**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.**

```
# 1º Argumento --> Archivo del que vamos a leer las líneas a encriptar
# Salida --> indica la modificación o no por consola

entrada = sys.argv[1]

# Fichero de entrada en modo lectura
with open(entrada, 'r') as fich:

    indice = 1
    # Leo línea a línea del fichero y sustituyo los retornos de carro por ''
    for ruta in fich:
        ruta = ruta.replace('\n', '')

        # En partes almaceno cada parte separada por ;
        partes = ruta.split(";")

        # Utilizo la misma función que he usado en BUILDHASH.py
        hash = hashlib.sha256()

        # Generar el código hash de la ruta que hay en el fichero
        with open(partes[0], 'rb') as binfile:
            texto = 1
            while texto:
                # Vamos leyendo
                texto = binfile.read(1024)
                hash.update(texto)

                # Genero el código hash de la ruta
                textoHash = hash.hexdigest()

        # Obtengo las propiedades
        prop = os.stat(partes[0])
        prop = repr(prop).encode()

        # Código hash generado para las propiedades
        propHash = hash.hexdigest()

        # Comparamos los nuevos códigos hash del fichero y propiedades
        # con los que ya había en el fichero para ver si han cambiado o no

        if(propHash != partes[2] and textoHash != partes[1]):
            print("Para el fichero ", indice, " con ruta ", partes[0], " se han modificado tanto el archivo como sus propiedades")
            indice = indice + 1
            continue
        else:
            print("Para el fichero ", indice, " con ruta ", partes[0], " no se ha modificado ni el archivo ni las propiedades")

        if(textoHash != partes[1]):
            print("Para el fichero ", indice, " con ruta ", partes[0], " se ha modificado el fichero")
        if(propHash != partes[2]):
            print("Para el fichero ", indice, " con ruta ", partes[0], " se ha modificado las propiedades del fichero")

        indice = indice + 1
```

Para comprobar que ambos programas funcionan voy a aportar capturas probando todas las posibilidades.

#### **BUILDHASH:**

```
PS C:\Users\carlo> python BUILDHASH2.py filesIguales.txt SalidaIguales
PS C:\Users\carlo> python BUILDHASH2.py files.txt SalidaDistintos
```

Ejecuto el programa dos veces en la primera filesIguales contiene las rutas a dos archivos que contienen exactamente el mismo contenido, por tanto, el fichero de salida SalidaIguales, debería tener el mismo código hash tanto para el fichero como para las propiedades de este.

```
C:\Users\carlo\filesIguales.txt;4b51c8f9b0c19f13d056348c3a4ee8bb7e11b7108a4b9f5004d1a5f42c83bc71;4b51c8f9b0c19f13d056348c3a4ee8bb7e11b7108a4b9f5004d1a5f42c83bc71
C:\Users\carlo\files3.txt;4b51c8f9b0c19f13d056348c3a4ee8bb7e11b7108a4b9f5004d1a5f42c83bc71;4b51c8f9b0c19f13d056348c3a4ee8bb7e11b7108a4b9f5004d1a5f42c83bc71
```

Tanto filesIguales como files3 tienen el mismo contenido.

Mientras que ahora en la segunda línea de ejecución files contiene las rutas a dos archivos que tienen distinto contenido por tanto el código hash para el fichero, así como el de las propiedades ha de ser distinto

```
C:\Users\carlo\files.txt;09093c473985a4976ab123355c44712eb7972a571058f10173f752a535a6e5d1;09093c473985a4976ab123355c44712eb7972a571058f10173f752a535a6e5d1
C:\Users\carlo\files2.txt;b4cf6600a50472e5ac2c9159aeb60e3741328c0b0a53e6c7b1720632a3bae97b;b4cf6600a50472e5ac2c9159aeb60e3741328c0b0a53e6c7b1720632a3bae97b
```

### CHECKHASH:

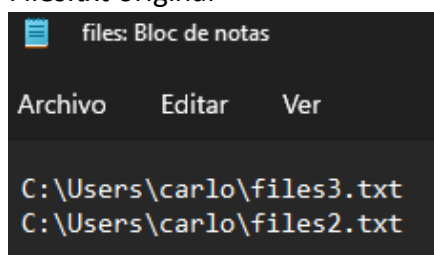
Este programa comprueba si se ha producido algún cambio en el archivo o sus propiedades, o en ambas. Para ello toma como entrada el fichero de salida del programa anterior, halla el código hash del fichero y de las propiedades de nuevo y compara con las que hay en el fichero de salida del programa anterior. Si alguna cambia el fichero ha sido modificado y si no pues no lo ha sido.

### Prueba ambos sin modificar:

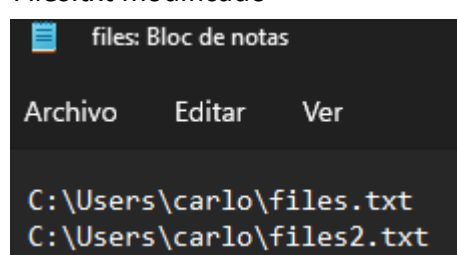
```
Para el fichero 1 con ruta C:\Users\carlo\files.txt no se ha modificado ni el archivo ni las propiedades
Para el fichero 2 con ruta C:\Users\carlo\files2.txt no se ha modificado ni el archivo ni las propiedades
```

### Modificando el primero de ellos:

Files.txt original



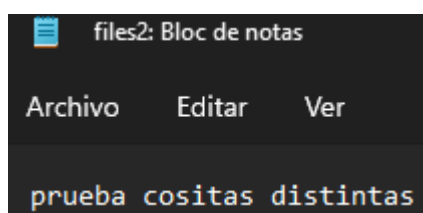
Files.txt modificado



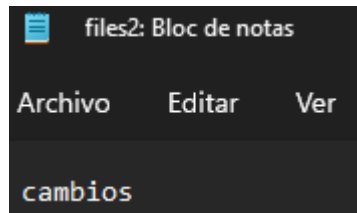
```
Para el fichero 1 con ruta C:\Users\carlo\files.txt se han modificado tanto el archivo como sus propiedades
Para el fichero 2 con ruta C:\Users\carlo\files2.txt no se ha modificado ni el archivo ni las propiedades
```

### Modificando ambos:

Files2.txt original



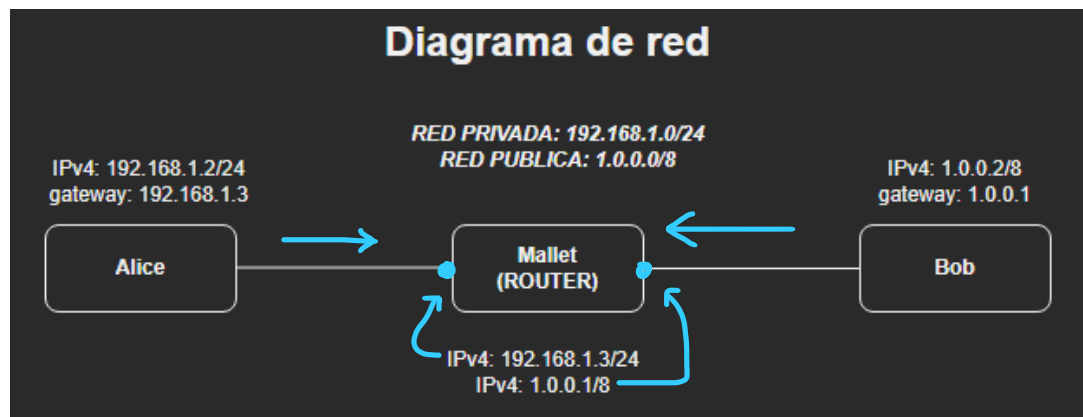
Files2.txt modificado



Para el fichero 1 con ruta C:\Users\carlo\files.txt se han modificado tanto el archivo como sus propiedades  
Para el fichero 2 con ruta C:\Users\carlo\files2.txt se han modificado tanto el archivo como sus propiedades

## PARTE 2

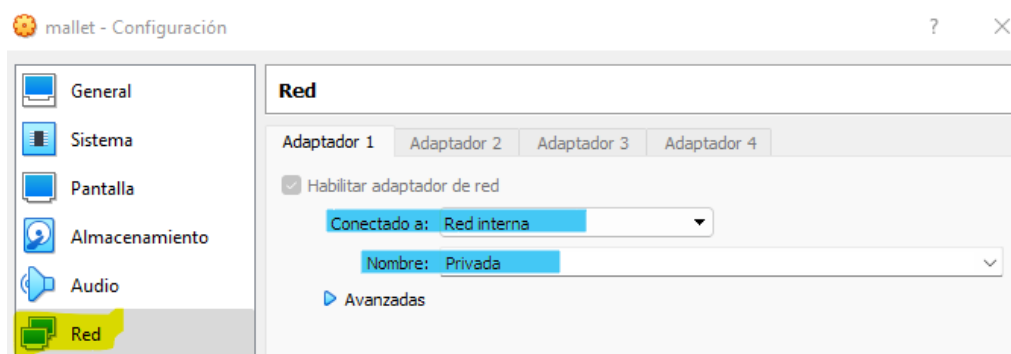
1. Dibuja el diagrama de red lo más detallado posible.

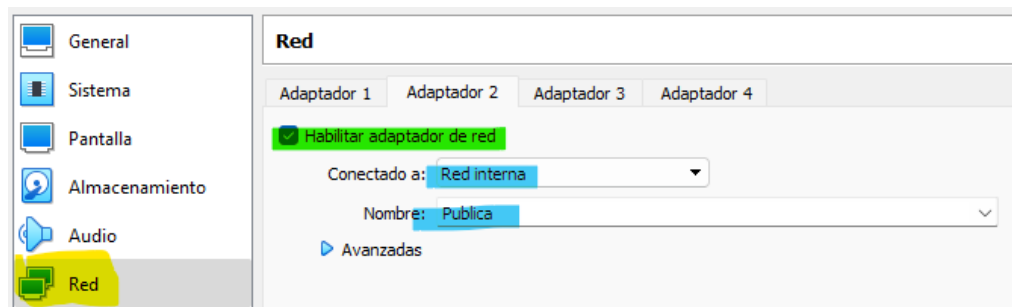


La puerta de enlace de Bob es 1.0.0.1/8 porque es la primera máquina que está en su mismo segmento de red, es decir, por la que tiene que pasar para comunicarse con Alice. Y la puerta de enlace para Alice es 192.168.1.3/24 ya que es la primera máquina por la que tiene que pasar que está en su mismo segmento de red para comunicarse con Bob.

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.

Primero desde el propio VirtualBox modifíco la configuración de red de la máquina mallet, que es la que en este caso va a actuar como router va a tener dos nuevas redes. Por tanto, en la parte de configuración de red de VirtualBox modificamos el nombre de la red del adaptador 1 a "Privada" y también de red NAT a red interna. Después habilitamos el conector 2 y hago lo mismo, modifíco el nombre a "Pública".





Ahora igual que hicimos en la práctica 1 toca modificar el fichero `/etc/network/interfaces`, modificando la red `eth4` y añadiendo la nueva `eth5` con la ip y máscara de red adecuada:

**`eth4` → ip: 192.168.1.3 mask: 255.255.255.0**

**`eth5` → ip: 1.0.0.1 mask: 255.255.255.0**

```
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
```

Ahora que ya hemos configurado las redes, tenemos que reiniciar los servicios de red con el comando `"sudo /etc/init.d/networking restart"`.

```
mallet@mallet:~$ sudo /etc/init.d/networking restart
* Reconfiguring network interfaces...
ssh stop/waiting
ssh start/running, process 2014
ssh stop/waiting
ssh start/running, process 2072
```

Ahora ejecuto el comando `ip` a para ver si he configurado bien las interfaces de red `eth4` y `eth5`. Y observamos en la foto que aparecen ambas interfaces de red.

```
mallet@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 ql
en 1000
    link/ether 08:00:27:c5:ad:a8 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:fec5:ada8/64 scope link
        valid_lft forever preferred_lft forever
3: eth5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP ql
en 1000
    link/ether 08:00:27:89:bc:bc 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:fe89:bcbc/64 scope link
        valid_lft forever preferred_lft forever
```

Para comprobar que ambas interfaces se han actualizado y creado de forma correcta, lanzo un ping a ambas ip's de 2 paquetes y vemos que los paquetes son transmitidos y recibidos.

```
mallet@mallet:~$ ping 1.0.0.1 -c 2
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=0.016 ms
64 bytes from 1.0.0.1: icmp_seq=2 ttl=64 time=0.017 ms

--- 1.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.016/0.016/0.017/0.004 ms
mallet@mallet:~$ ping 192.168.1.3 -c 2
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=0.014 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=64 time=0.020 ms

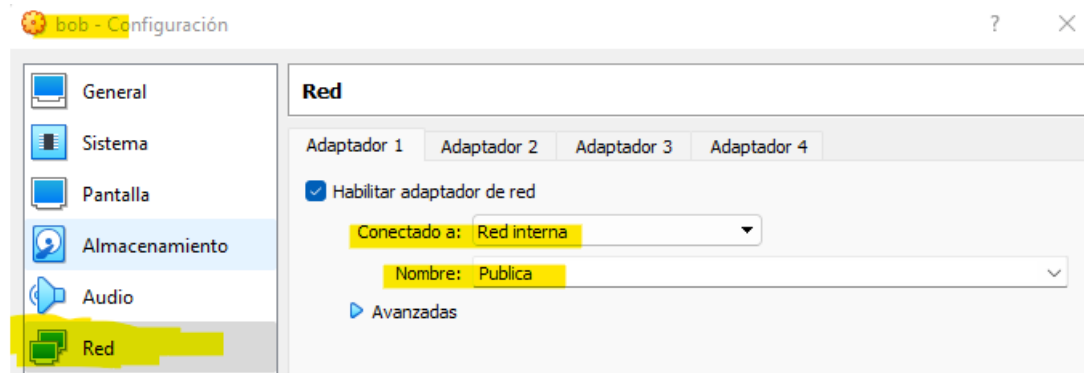
--- 192.168.1.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.014/0.017/0.020/0.003 ms
```

Para habilitar el bit de enrutamiento he consultado en internet como hacerlo y en esta página web nos indica como (<https://shelllavie.com/habilitar-enrutamiento-linux/>) por tanto ejecuto el comando "sudo su" y "echo 1 > /proc/sys/net/ipv4/ip\_forward" y con este comando ya habríamos modificado el bit de enrutamiento para que todo funcione de manera correcta. Pero este comando modifica el bit de enrutamiento de forma temporal, cuando reiniciemos la máquina perderemos esta configuración.

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

**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.**

Ahora para que bob esté dentro de la red Pública, desde VirtualBox, y en la máquina virtual de bob, ajustes de red, y modificamos el adaptador 1, poniendo red interna, y nombre “Publica”.



Ahora arranco la máquina para cambiar la ip de bob:

```
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

auto eth0
iface eth0 inet static
    address 1.0.0.2/8
    netmask 255.255.255.0

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

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

Ahora que ya he configurado la red, tenemos que reiniciar los servicios de red con el comando `“sudo /etc/init.d/networking restart”`. Pero al hacer esto me daba error y no me modificaba la ip de bob.

Para hacerlo he asignado la ip a bob de forma temporal con el comando `“ifconfig eth0 1.0.0.2”` y para asignar la del gateway o puerta de enlace he utilizado el comando `“route add default gw 1.0.0.1 eth0”`.

```
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 qlen 1000
    link/ether 08:00:27:f5:bb:62 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:fef5:bb62/64 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
```

Y ahora para modificar el gateway, comprobamos con “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
0.0.0.0        1.0.0.1         0.0.0.0         UG    0      0        0 eth0
```

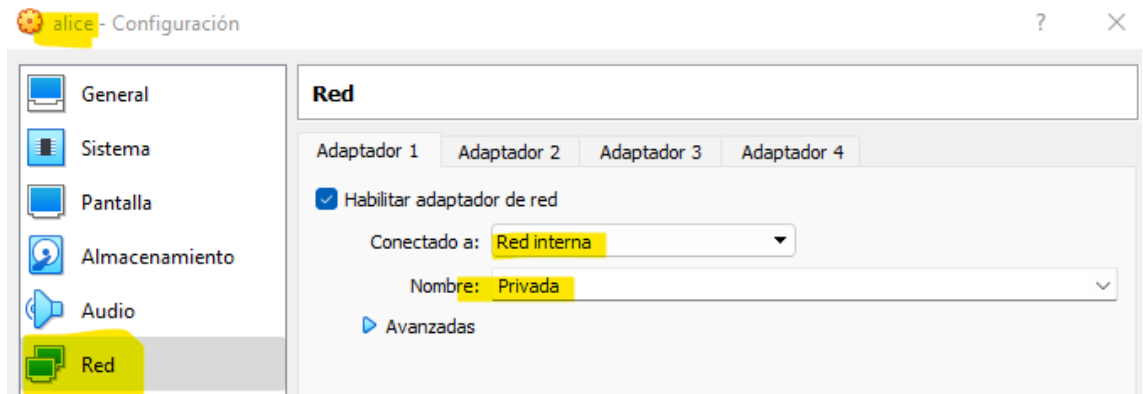
Para comprobar que la interfaz se ha actualizado y creado de forma correcta, lanzo un ping a la ip de 2 paquetes y vemos que los paquetes son transmitidos y recibidos.

```
bob:/home/bob# ping 1.0.0.2 -c 2
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=0.362 ms
64 bytes from 1.0.0.2: icmp_seq=2 ttl=64 time=0.302 ms

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

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.

Al igual que en el punto 2 y 3 modificamos primero las opciones de red de alicé dentro del VirtualBox, modificando el adaptador 1 modificando el nombre a Privada.



Para seguir modificando la dirección ip arranco la máquina de alice e igual que en el punto 2, modifico el fichero `/etc/network/interfaces` y modifico su interfaz de red (eth3) con ip, máscara y gateway.

```
auto lo eth3
iface lo inet loopback

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

Y de nuevo reinicio los servicios de red con el comando `"sudo /etc/init.d/networking restart"` y compruebo con el comando `ip a` si la interfaz de red ha cambiado su configuración:

```
root@alice:/home/alice# sudo /etc/init.d/networking restart
* Reconfiguring network interfaces...
RTNETLINK answers: No such process
SIOCDELRT: No such process
ssh stop/waiting
ssh start/running, process 2409

root@alice:/home/alice# 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 s
    en 1000
    link/ether 08:00:27:f9:69:5e brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.2/24 brd 192.168.1.255 scope global eth3
    inet6 fe80::a00:27ff:fef9:695e/64 scope link
        valid_lft forever preferred_lft forever
```



Para comprobar que la interfaz se ha actualizado y creado de forma correcta, lanzo un ping a la ip de 2 paquetes y vemos que los paquetes son transmitidos y recibidos.

```
alice@alice:~$ ping 192.168.1.2 -c 2
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=0.016 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=64 time=0.017 ms

--- 192.168.1.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.016/0.016/0.017/0.004 ms
```

Como en alice también le asignamos el gateway voy a comprobar si se ha modificado ejecutando el comando “route -n”:

```
root@alice:/home/alice# route -n
Kernel IP routing table
Destination    Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0    0.0.0.0        255.255.255.0   U        0      0        0 eth3
169.254.0.0    0.0.0.0        255.255.0.0     U       1000    0        0 eth3
0.0.0.0        192.168.1.3    0.0.0.0         UG       100    0        0 eth3
```

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

Una vez que ya han sido modificadas todas las redes, voy a realizar pings de una máquina a otras para ver si la configuración de estas es correcta y con ello las máquinas se pueden comunicar entre sí.


Primero realizo un ping de dos paquetes desde Bob a una de las ip de Mallet (1.0.0.1), después otro ping a la otra ip de Mallet (192.168.1.3) y con esto hemos comprobado que Bob tiene conexión con Mallet, para comprobar si también la tiene con Alice voy a hacer un ping a su ip (192.168.1.2) y cómo podemos ver también se completa el ping por tanto Bob puede comunicarse con cualquiera de las otras dos máquinas virtuales.

```
bob:/home/bob# ping 1.0.0.1 -c 2
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=1.42 ms
64 bytes from 1.0.0.1: icmp_seq=2 ttl=64 time=0.775 ms

--- 1.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1004ms
rtt min/avg/max/mdev = 0.775/1.098/1.421/0.323 ms
bob:/home/bob# ping 192.168.1.3 -c 2
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=0.385 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=64 time=0.732 ms

--- 192.168.1.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.385/0.558/0.732/0.178 ms
bob:/home/bob# ping 192.168.1.2 -c 2
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=3.14 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=63 time=0.808 ms

--- 192.168.1.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1005ms
rtt min/avg/max/mdev = 0.808/1.978/3.148/1.170 ms
```



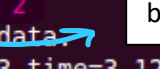
Ahora para comprobar la comunicación total entre máquinas, desde alice voy a realizar pings a ambas ips de mallet (1.0.0.1 y 192.168.1.3) y a bob (1.0.0.2).

```
root@alice:/home/alice# ping 192.168.1.3 -c 2
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=0.437 ms
64 bytes from 192.168.1.3: icmp_seq=2 ttl=64 time=0.357 ms

--- 192.168.1.3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.357/0.397/0.437/0.040 ms
root@alice:/home/alice# ping 1.0.0.1 -c 2
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=0.412 ms
64 bytes from 1.0.0.1: icmp_seq=2 ttl=64 time=0.394 ms

--- 1.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.394/0.403/0.412/0.009 ms
root@alice:/home/alice# ping 1.0.0.2 -c 2
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=3.12 ms
64 bytes from 1.0.0.2: icmp_seq=2 ttl=63 time=0.488 ms

--- 1.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.488/1.808/3.128/1.320 ms
```



Como se puede observar existe comunicación total entre las máquinas puesto que los pings realizados son transmitidos y recibidos con éxito.

En cuanto al valor del ttl podemos ver que cuando el paquete atraviesa el router es decir, pasa por mallet, este se decrementa en 1 su valor, puesto que se completa un salto.

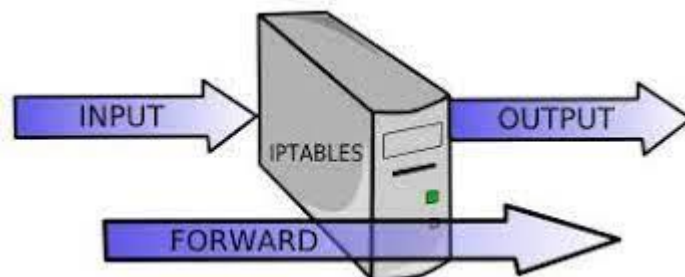
En resumen, si el ping va de bob aallet o de alicia aallet, el valor del ttl es 64, ya que no se completa ningún salto, pero si el paquete necesita pasar por el router (mallet) es decir, ping de bob a alicia o de alicia a bob el ttl = 63 puesto que necesita pasar por mallet y se completa un salto.

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

En iptables existen distintas cadenas como INPUT, OUTPUT, FORWARD etc. Que indican por donde van a circular los paquetes dentro del sistema, y las diferencias entre ellas son:

- **INPUT:** hace referencia a los paquetes que tienen como destino el equipo local independientemente del origen que estos tengan. O dicho de otra forma, paquetes que entran en el host, solo entran no salen.
- **OUTPUT:** hace referencia a los paquetes que son generados en el equipo local y que van a salir de este y de nuestra red.
- **FORWARD:** hace referencia a los paquetes que pasan por el equipo local, pero que son generados en equipos remotos y además tienen como destino final otro equipo/s diferente/s.  $FORWARD = INPUT + OUTPUT$ .

En forma de esquema:



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

Antes de configurar ninguna regla voy a ver con el comando `"sudo iptables -L"` que reglas del firewall están ya asignadas.

```
mallet@mallet:~$ sudo iptables -L
[sudo] password for mallet:
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
```

Ahora tenemos que añadir la regla que nos pide, para ello ejecutamos el comando “`iptables -A FORWARD -s 1.0.0.1/8 -d 192.168.1.0/24 -p icmp -icmp-type 8 -j DROP`” como superusuario. El comando es “-p icmp” porque es el protocolo icmp como indica el enunciado y para decir que únicamente permita icmp 8 “-icmp-type 8”.

```
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
```

Una vez creada lo comprobamos realizando el comando “`iptables -L`” y viendo si la nueva regla creada está.

```
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
```


La regla que hemos creado va a impedir que un ping de bob a alice se pueda realizar ya que únicamente está permitiendo que se realicen peticiones ICMP desde la red interna, por ello voy a realizar un ping de bob a alice, y no debería de realizarse con éxito puesto que ahora el firewall va a estar denegando el aso de estos paquetes

```
bob:/home/bob# ping 192.168.1.2 -c 2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
--- 192.168.1.2 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1001ms
```

Como se puede ver en la captura de arriba, realizo un ping de dos paquetes de bob a alice, que son trasmitidos, pero no recibidos. Esto es debido a la nueva regla que hemos introducido en el firewall, en el router, enallet, que deniega el paso de estos paquetes.

Ahora voy a borrar esa nueva regla creada y efectuar de nuevo el ping de bob a alice, y veremos como ahora sí que llegan los paquetes y nada los retiene. El comando indicado con la flecha es el encargado de borrar la regla.

```
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
```



Efectúo de nuevo el ping:

```
bob:/home/bob# ping 192.168.1.2 -c 2  
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=7.69 ms  
64 bytes from 192.168.1.2: icmp_seq=2 ttl=63 time=1.27 ms  
  
--- 192.168.1.2 ping statistics ---  
2 packets transmitted, 2 received, 0% packet loss, time 1002ms  
rtt min/avg/max/mdev = 1.273/4.485/7.697/3.212 ms
```

Ahora como vemos los paquetes son transmitidos y recibidos, por tanto, la regla era la encargada de retenerlos y evitar que llegasen en este caso a alice.

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.

Ahora necesitamos crear una regla que permita solo a la ip 192.168.1.3 o maltet, conectarse con la ip 192.168.1.2 o alice. Para ello primero creamos las conexiones FORWARD ssh que pasan por el router. Para ello ejecuto el comando "iptables -A FORWARD -s 0.0.0.0/0 -p tcp --dport 22 -j DROP" Elegimos el puerto TCP 22 porque es el que se corresponde con las conexiones ssh.

## TCP 22

SSH opera en el puerto **TCP 22** de forma predeterminada (aunque esto se puede cambiar si es necesario). 27 may 2022

Una vez ejecutado el comando que crea la regla, ejecutamos “iptables -L” y vemos que existen las dos reglas creadas, la del apartado anterior y esta.

```
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       icmp -- 1.0.0.0/8             192.168.1.0/24      icmp echo-request
DROP       tcp  -- anywhere             anywhere            tcp dpt:ssh
Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

Con esta nueva regla que hemos añadido al firewall nos debería de permitir realizar una conexión ssh de Mallet a Alice o de Mallet a Bob pero no de Bob a Alice y viceversa puesto que con la nueva regla añadida el router debería de denegarnos estas conexiones ssh que pasan por él.

Primero voy a hacer una conexión de Mallet a Alice que debería de permitirnos conectar, puesto que en ningún momento atraviesa el router:

```
root@mallet:/home/mallet# ssh alice
The authenticity of host 'alice (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 'alice,192.168.1.2' (RSA) to the list of known hosts.
root@alice'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.

You have new mail.
Last login: Sat Oct  1 14:45:26 2022 fromallet.local
root@alice:~# exit
logout
Connection to alice closed.
```

Como se puede ver hemos podido establecer la conexión, ahora voy a realizar una conexión de Bob a Mallet que también nos debería permitir conectarnos, puesto que en ningún momento atraviesa el router:

```
bob:/home/bob# sshallet
root@mallet's password:
```

Sin embargo, si ahora intentamos realizar una conexión ssh que atraviesa el router como puede ser una conexión de Bob a Alice o viceversa no debería de dejarnos puesto que el router deniega estas.

```
bob:/home/bob# ssh alice  
bob:/home/bob#
```

```
alice@alice:~$ ssh bob  
ssh: connect to host bob port 22: No route to host
```

Como hemos dicho no podemos establecer la conexión ssh si esta tiene la obligación de pasar por el router, puesto que la regla FORWARD que hemos añadido la deniega.