Informe de la tercera actividad GSI

@ALVPI

Contents

1 Parte 1

1.1 Funciones Hash

Son algoritmos que convierten una entrada (mensaje, carácter, fichero,...) en una salida de tamaño fijo conocida como **valor hash**.

1.1.1 Propiedades útiles por las cuales son interesantes desde el punto de vista de la Seguridad de la información

Dentro de las funciones hash, tenemos una que es clave para la seguridad de la información, la cual es que la función hash es **irreversible**, eso ¿qué quiere decir?, que el coste computacional para revertir el resultado de una función hash es tan alto que no se puede dar marcha atrás a la función.
Además de esta propiedad clave, tenemos que el resultado de las funciones hash es único.

1.2 Sentido/Utilidad de las funciones Hash en el ámbito de la Seguridad de la información

En esta disciplina de la informática nos interesa utilizar funciones hash puesto que es una manera rápida y segura de securizar nuestra información.
De tal manera que no es el mismo proceso que el cifrado, puesto que este es reversible, con un mayor o menor grado de dificultad en esta tarea pero es posible revertirlo, mientras que si empleamos una función hash para **hashear** una información, nos aseguramos de que es de sentido único, además de rápido e irreversible.

1.3 Tabla sobre las características de algunas funciones Hash

Función	Número caracteres en hexadecimal	Número de bits	¿Vulnerable?
MD5	32	128	Sí
SHA1	40	160	Sí
SHA256	256	1024	No
SAH512	512	2048	No

Table 1: Vulnerable hace referencia a la presencial de colisiones

1.4 Herramientas para la construcción de Hashes

En esta sección, lo que haremos será emplear diferentes funciones de creación de hashes nativas de las máquinas para observar cómo se comportan bajo diferentes parámetros.

1.4.1 md5sum

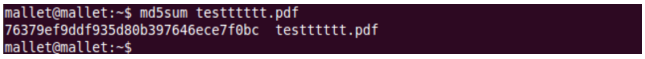

Figure 1: Resultado de hashear un fichero en md5

Figure 2: Ejemplo de aplicar un hash md5 a una letra

1.4.2 sha1sum

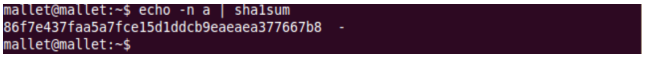

Figure 3: Ejemplo de aplicación de la función hash sha1sum a una letra

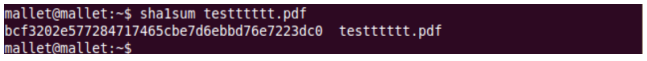

Figure 4: Ejemplo de aplicación de la función hash sha1sum a un fichero

1.4.3 sha256sum

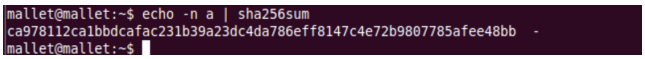

Figure 5: Ejemplo de aplicación de la función hash sha256sum a una letra

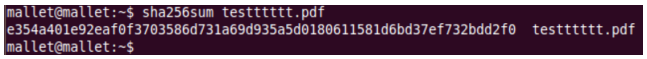

Figure 6: Ejemplo de aplicación de la función hash sha256sum a un fichero

1.4.4 sha512sum

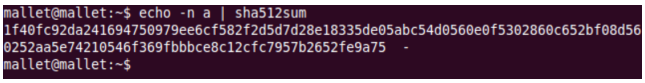

Figure 7: Ejemplo de aplicación de la función hash sha512sum a una letra

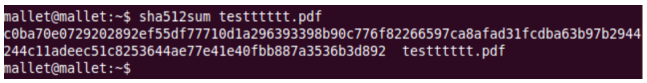

Figure 8: Ejemplo de aplicación de la función hash sha512sum a una letra

1.5 hasid

Esta herramienta **no** viene preinstalada en nuestras máquinas Ubuntu, por lo que tenemos dos alternativas: instalar otra distribución, como por ejemplo Kali, que tiene instaladas una gran cantidad de herramientas para pentesting, o "actualizar" los repositorios con los que se tiene que conectar la máquina para actualizar el sistema e instalar lo necesario.
Aunque haya optado por la primera opción, voy a explicar de manera muy breve cómo se haría esto:

1. `sudo nano /etc/apt/sources.list`
2. reemplazamos todas las direciciones url por la siguiente

   `http://old-releases.ubuntu.com/ubuntu/`
3. `sudo apt-get update`
4. `pip install hash id`

En kali, he aplicado el mismo comando para intentar identificar el hash que ha sido empleado sobre una clave

```
echo -n a | md5sum | awk'{print $1 }' | hashid
```

1.5.1 Resultados de aplicar `hashid`

```
┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$ echo -n  'a' | md5sum | awk '{print $1}' | hashid
Analyzing '0cc175b9c0f1b6a831c399e269772661'
[+] 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

┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$
```

Figure 9: Lista de posibles funciones de hash que se han podido aplicar sobre ese hash hecho en md5

```
┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$ echo -n  'a' | sha1sum | awk '{print $1}' | hashid
Analyzing '86f7e437faa5a7fce15d1ddcb9eaeaea377667b8'
[+] SHA-1
[+] Double SHA-1
[+] RIPEMD-160
[+] Haval-160
[+] Tiger-160
[+] HAS-160
[+] LinkedIn
[+] Skein-256(160)
[+] Skein-512(160)

┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$
```

Figure 10: Lista de posibles funciones de hash que se han podido aplicar sobre ese hash hecho en sha1

```
┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$ echo -n  'a' | sha256sum | awk '{print $1}' | hashid
Analyzing 'ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb'
[+] Snefru-256
[+] SHA-256
[+] RIPEMD-256
[+] Haval-256
[+] GOST R 34.11-94
[+] GOST CryptoPro S-Box
[+] SHA3-256
[+] Skein-256
[+] Skein-512(256)

┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$
```

Figure 11: Lista de posibles funciones de hash que se han podido aplicar sobre ese hash hecho en sha252

```
┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$ echo -n  'a' | sha512sum | awk '{print $1}' | hashid
Analyzing '1f40fc92da241694750979ee6cf582f2d5d7d28e18335de05abc54d0560e0f5302860c652bf08d5
60252aa5e74210546f369fbbbce8c12cfc7957b2652fe9a75'
[+] SHA-512
[+] Whirlpool
[+] Salsa10
[+] Salsa20
[+] SHA3-512
[+] Skein-512
[+] Skein-1024(512)

┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$
```

Figure 12: Lista de posibles funciones de hash que se han podido aplicar sobre ese hash hecho en sha512

1.6 Función hash sobre un fichero modificado

Una de las principales ventajas en tema de seguridad sobre las funciones hash, o el hecho de hashear un fichero, es que ante un cambio en el mismo, el resultado de aplicar la función a dicho fichero varía, puesto que las funciones no es que se apliquen solo al título sino que se aplica sobre todo el fichero.
Pero al cambiar los permisos estamos cambiando los metadatos del sistema operativo, no modificamos el fichero tal cual, por lo que el resultado no debería cambiar. De tal manera que con `chmod` modificaré los permisos de los ficheros y compararemos el resultado de cada una de las funciones sobre el fichero para probar nuestra teoría.
testttttt.pdf

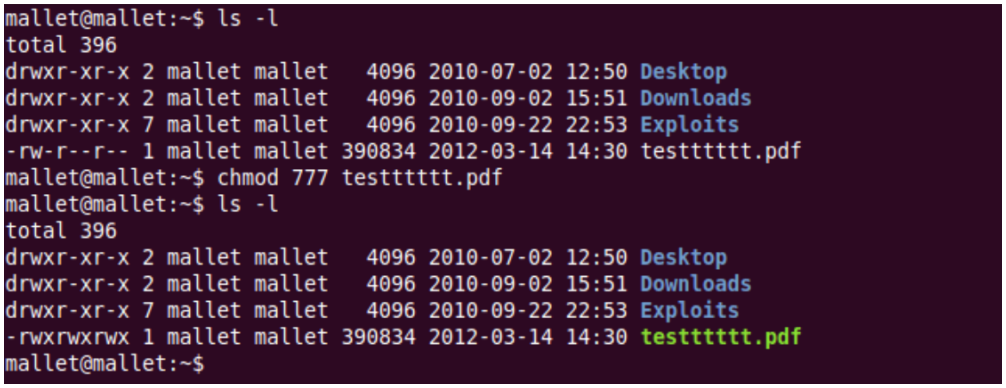

Figure 13: cambio de los permisos en testttttt.pdf para que usuario,grupo y el resto tengas todos los permisos

1.6.1 md5sum

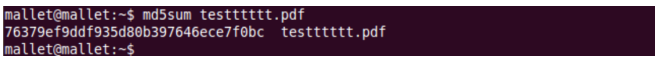

Figure 14: Resultado de hashearlo antes de cifrado

Figure 15: Resultado de hashear tras el chmod

1.6.2 sha1sum

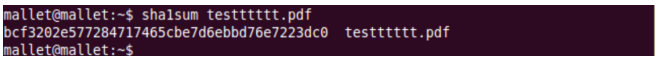

Figure 16: Resultados de aplicar la función hash antes de modificar los permisos

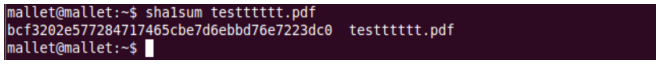

Figure 17: Resultados de aplicar la función después de la modificación

1.6.3 sha256sum

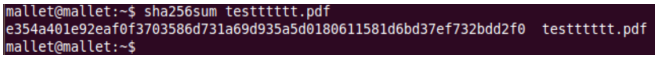

Figure 18: Resultados de aplicar la función hash antes de modificar los permisos

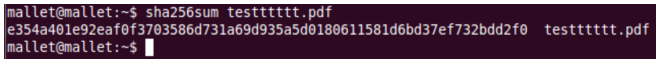

Figure 19: Resultados de aplicar la función después de la modificación

1.6.4 sha512sum

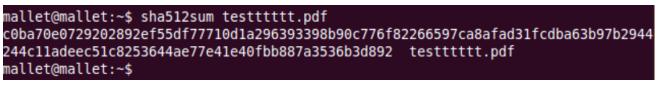

Figure 20: Resultados de aplicar la función hash antes de modificar los permisos

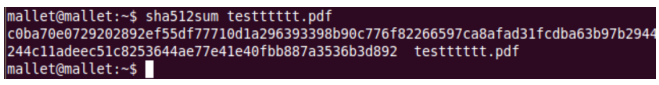

Figure 21: Resultados de aplicar la función después de la modificación

1.7 Creación de usuarios en Kali

La práctica nos pide crear dos usuarios, el problema es que root está creado por defecto, por lo que haremos para él será cambiar la contraseña a la proporcionada en la práctica.

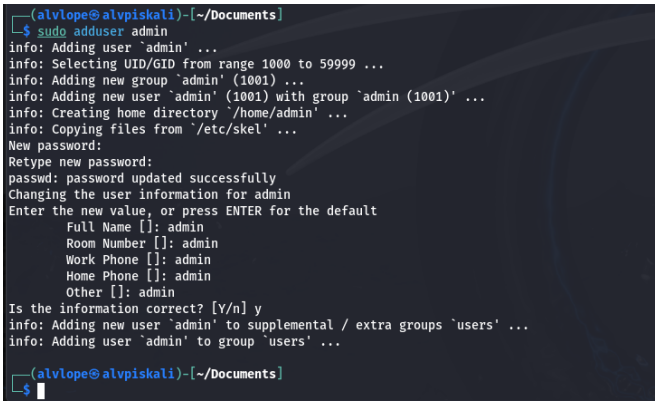

Figure 22: Proceso de creación de usuario

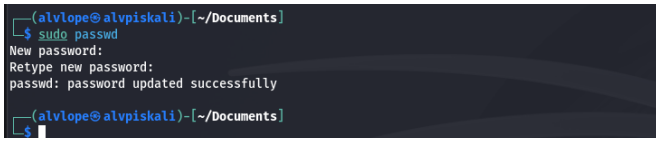

Figure 23: al tener permisos de sudo y ejecutar el cambio de contraseña sin proporcionarle un usuario, estamos cambiar la contraseña del root

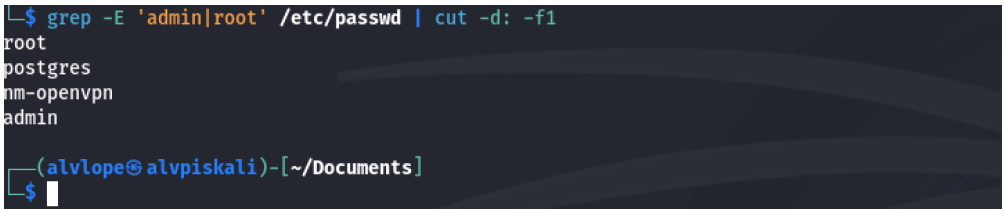

Figure 24: comprobamos que los usuarios se han creado de manera correcta

1.8 ¿Para qué se puede utilizar la herramienta *John the Ripper*

Esta herramienta, popularmente conocida solo como John, nos permite descifrar contraseñas, aunque también es ampliamente utilizada en ataques de fuerza bruta (empleando un diccionario), los cuales consisten en probar todas las posibles combinaciones de contraseñas para acceder a nuestro objetivo.

Figure 25: Hashid no es capaz de identificar qué tipo de hash es y

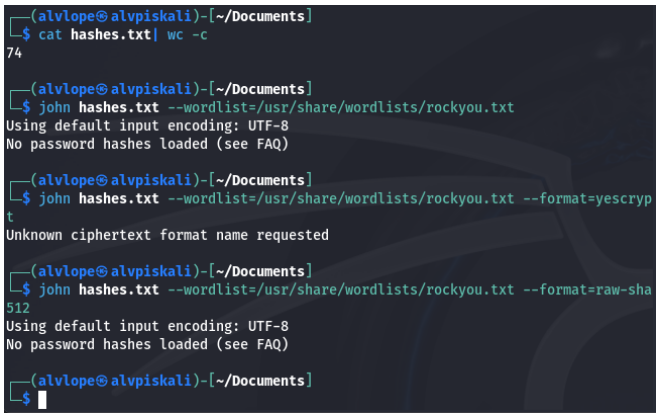

Figure 26: Intentamos sacar la contraseña con john y un cifrado 512 y un yescrypt

El problema de estas contraseñas es que `$y$` es un cifrado del tipo *bcrypt* pero no es ningún identificador estándar, por lo que tendremos que utilizar algún diccionario, como puede ser **rockyou.txt** (que ya viene descargado en la distribución de Kali), pero al tener 74 caracteres no se corresponde con ninguna de las funciones hash y al no tener pistas de por dónde tirar, no he podido ejecutar el ataque.
Como conclusión, John the Reaper es una herramienta muy poderosa, pero realmente no es infalible, puesto que si hay varias funciones de hash aplicadas sobre el mismo dato o la función empleada no está bien documentada, no vamos a poder ejecutar el ataque de fuerza bruta.
Para conseguir el ataque de fuerza bruta, he cambiado la función hash del sistema a **md5**

```
  GNU nano 8.1                /etc/pam.d/common-password *
#for other options.

# As of pam 1.0.1-6, this file is managed by pam-auth-update by default.
# To take advantage of this, it is recommended that you configure any
# local modules either before or after the default block, and use
# pam-auth-update to manage selection of other modules.  See
# pam-auth-update(8) for details.

# here are the per-package modules (the "Primary" block)
password        [success=1 default=ignore]      pam_unix.so obscure md5
# here's the fallback if no module succeeds
password        requisite                       pam_deny.so
# prime the stack with a positive return value if there isn't one already;
# this avoids us returning an error just because nothing sets a success code
# since the modules above will each just jump around
password        required                        pam_permit.so
# and here are more per-package modules (the "Additional" block)
password        optional        pam_gnome_keyring.so
# end of pam-auth-update config
```

Figure 27: Cambio cifrado md5 en `/etc/pam.d/common-password`

```
┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$ sudo nano /etc/pam.d/common-password
[sudo] password for alvlope:

┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$ sudo passwd admin
New password:
Retype new password:
passwd: password updated successfully

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

┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$ sudo cat /etc/shadow | grep -E "admin|root" > hashes.txt

┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$ cat hashes.txt
root:$1$OkdzFdlD$QG2kmQpIv5cg8zB3c7kqP.:20060:0:99999:7:::
admin:$1$j8V4mBNa$7JiVeUsykuXD8YK19q2fy.:20060:0:99999:7:::

┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$ john --wordlist=/usr/share/wordlists/rockyou.txt hashes.txt > resultado.txt
Warning: detected hash type "md5crypt", but the string is also recognized as "md5crypt-long"
Use the "--format=md5crypt-long" option to force loading these as that type instead
Using default input encoding: UTF-8
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
1g 0:00:00:38 DONE (2024-12-03 17:28) 0.02630g/s 370956p/s 370976c/s 370976C/s !!!0mc3t..*7¡Vamos!
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$ cat resultado.txt
Loaded 2 password hashes with 2 different salts (md5crypt, crypt(3) $1$ (and variants) [MD5 256/256 AVX2 8x3])
123456           (admin)

┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$
```

Figure 28: Proceso que he seguimos para sacar eCual hash de las contraseñas md5

1.9 Buildhash

```
 GNU nano 8.1                                buildbash.sh
#!/bin/bash
#Comprobamos que nos hayan pasado fichero
if [ -z "$1" ]; then
        echo "Por favor proporcione un fichero como argumento"
        exit 1
fi
#verificamos que el fichero exista
if [ ! -f "$1" ]; then
        echo "El archivo '$1' no existe"
        exit 1
fi
#Vamos a leer el fichero linea por linea
while IFS= read -r filename; do
        #Eliminamos todos los espcios antes y después
        filename=$(echo "$filename" | sed 's/^[ \t]*//;s/[ \t]*$//')
        #Vamos a saltar las lineas vacias del fichero
        [ -z "$filename" ] && continue
        #Como cada linea tiene que ser un fichero que exite vamos a comprobar esto
        if [ -f "$filename" ]; then
                #Aplico md5sum al contenido del fichero
                fileHash=$(md5sum "$filename" | awk '{print $1}')
                #Saco las propiedades del archivo
                fileProp=$(stat --format="%s %y %a %U %G" "$filename")
                #Calculamos el hash de las propiedads
                propHash=$(echo "$fileProp" | md5sum | awk '{print $1}')
                #Imprimimos la cadena con el nombre fichero;hash fichero;hash propiedades
                echo "$filename;$fileHash;$propHash"
        else
                echo "El archivo '$1' no exite"
                exit
        fi
done < "$1"
```

Figure 29: Buildhash en shellscript

```
┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$ chmod +x buildbash.sh

┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$ ./buildbash.sh fich.txt
hola.txt;8609d126e5037cf8e8f0d453d046f61a;c87e8e62ba0ebb9b9c0fd59c3f0290f7
adios.txt;d41d8cd98f00b204e9800998ecf8427e;8b27b1fdeb2c6a3892ceb02093db1cec

┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$ cat fich.txt
hola.txt
adios.txt


┌──(alvlope㉿alvpiskali)-[~/Documents]
└─$
```

Figure 30: Ejecución de Buildhash

2 Parte 2

2.1 Diagrama de red

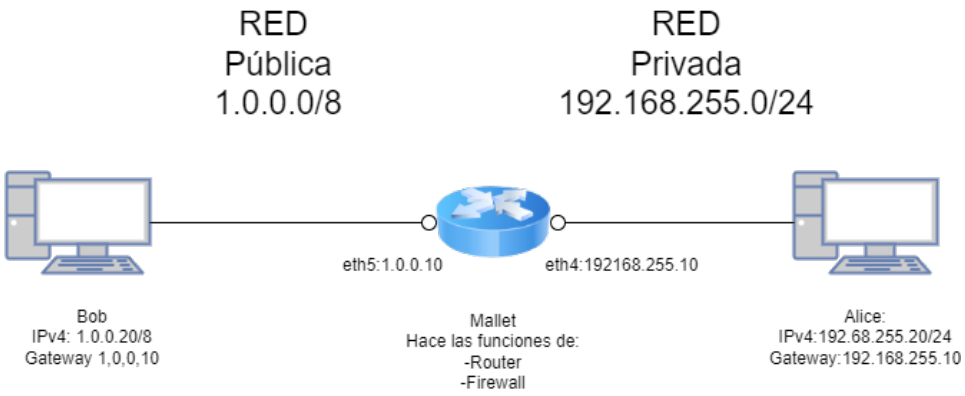

Figure 31: Diagrama de la red

2.2 Configuración de Mallet para que actue como router

2.2.1 Configuración del bit de enrutamiento

```
mallet@mallet:~$ sudo nano /etc/sysctl.conf
[sudo] password for mallet:
mallet@mallet:~$ sudo sysctl -p
net.ipv4.ip_forward = 1
mallet@mallet:~$
```

Figure 32: Configuración del bit de enrutamiento para IPv4

Al escribir `net.ipv4.ip_forward = 1` nos aseguramos de que el bit de enrutamiento esté activado. Posteriormente guardamos los cambios con `sysctl -p`.

2.2.2 Configuración de la red interna "Privada" y "Pública"

La primero que debemos hacer es configurar una nueva interfaz de red y asignar la red a la que deben ir (la red nat para la pública y la red interna para la privada)

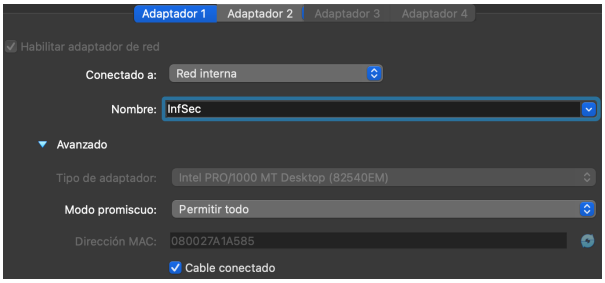

Figure 33: Cambiamos el adaptado 1 para la red interna

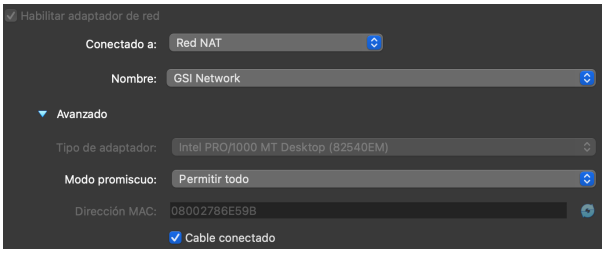

Figure 34: Usamos la red Nat para que sea nuestra red púlica

Posteriormente a esto, entramos en Mallet y configuramos el bit de enrutamiento para que nuestra máquina actúe de router.

```
 GNU nano 2.2.2            File: /etc/sysctl.conf


# Uncomment the next line to enable TCP/IP SYN cookies
#net.ipv4.tcp_syncookies=1

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

# Uncomment the next line to enable packet forwarding for IPv6
#net.ipv6.conf.all.forwarding=1


###################################################################
# Additional settings - these settings can improve the network
# security of the host and prevent against some network attacks
# including spoofing attacks and man in the middle attacks through
# redirection. Some network environments, however, require that these
# settings are disabled so review and enable them as needed.
#
# Ignore ICMP broadcasts
```

Figure 35: Habilitamos el bit de enrutamiento

```
mallet@mallet:~$ sudo nano /etc/sysctl.conf
[sudo] password for mallet:
mallet@mallet:~$ sudo sysctl -p
net.ipv4.ip_forward = 1
mallet@mallet:~$
```

Figure 36: Reflejamos esos cambios en el sistema

Ahora lo que haremos será configurar las interfaces de red para ambas redes con las IPs que nos indica el enunciado.

```
 GNU nano 2.2.2          File: /etc/network/interfaces                Modified

auto eth4
iface eth4 inet static
        address 192.168.255.10
        netmask 255.255.255.0
        network 192.168.1.0
        broadcast 192.168.1.255

# configuracion de eth5 (red "publica")
auto eth5
iface eth5 inet static
        address 1.0.0.10
        netmask 255.0.0.0
        network 1.0.0.0
        broadcast 1.255.255.255
```

Figure 37: Interfaces eth4 y eth5 modificadas para satisfacer el enunciado

address: es la dirección IP de la interfaz.

netmask: es la máscara de subred asociada a la dirección IP.

network: Dirección de la subred.

broadcast: La dirección que permite enviar paquetes a todos los hosts de la red.

Reiniciamos la configuración de red y hacemos un `ip a` para cerciorarnos de que las tarjetas de red se han configurado correctamente.

```
mallet@mallet:~$ sudo /etc/init.d/networking restart
 * Reconfiguring network interfaces...
ssh stop/waiting
ssh start/running, process 1623
ssh stop/waiting
ssh start/running, process 1677
                                                           [ OK ]
mallet@mallet:~$
```

Figure 38: reiniciamos la configuración 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:a1:a5:85 brd ff:ff:ff:ff:ff:ff
    inet 192.168.255.10/24 brd 192.168.1.255 scope global eth4
    inet6 fe80::a00:27ff:fea1:a585/64 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 08:00:27:86:e5:9b 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:fe86:e59b/64 scope link
       valid_lft forever preferred_lft forever
mallet@mallet:~$
```

Figure 39: Vemos si se ha configurado bien

2.3 Configuración de Bob y Alice

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

Figure 40: Configuraciones de la interfaz `eth0` de Bob

```
  GNU nano 2.2.2                File: /etc/network/interfaces

auto lo
iface lo inet loopback

auto eth4
iface eth4 inet static
        address 192.168.255.20
        netmask 255.255.255.0
        gateway 192.168.255.10
```

Figure 41: Configuración de la interfaz `eth4` de Alice

2.4 Pruebas de que las 3 máquinas se comunican

En esta parte, me he encontrado con dos problemas:

- El primero de ellos es que, por error, eliminé el fichero `/etc/network/interfaces` de Mallet, por lo que tuve que reinstalar la máquina para poder realizar correctamente la configuración del bit de enrutamiento.
- El segundo problema ocurrió al configurar a Alice. Al ejecutar `ip route -show`, me aparecía una ruta `169.254.0.0`, que corresponde a una puerta de enlace local (*APIPA*). Esto sucedió porque, al cambiar el adaptador de red de NAT (usado en prácticas anteriores) a una red interna, la ruta no se actualizó automáticamente.

```
alice@alice:~$ ip route show
192.168.255.0/24 dev eth4  proto kernel  scope link  src 192.168.255.20
169.254.0.0/16 dev eth4  scope link  metric 1000
default via 192.168.255.10 dev eth4  metric 100
alice@alice:~$ ip addr show eth4
2: eth4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 08:00:27:2a:ab:8d brd ff:ff:ff:ff:ff:ff
    inet 192.168.255.20/24 brd 192.168.255.255 scope global eth4
    inet6 fe80::a00:27ff:fe2a:ab8d/64 scope link
       valid_lft forever preferred_lft forever
alice@alice:~$ sudo ip route del 169.254.0.0/16
```

Figure 42: Eliminación de la ruta APIPA para Alice.

Vamos a verificar la comunicación entre las máquinas:

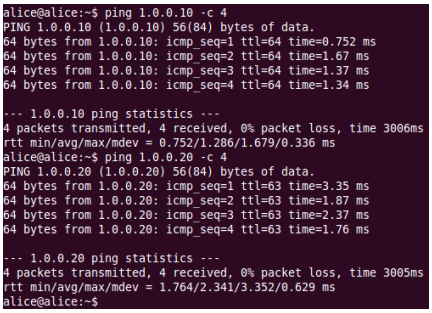

Figure 43: Pings desde Alice a la gateway **Públics** y Bob

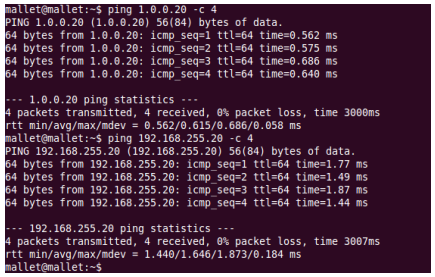

Figure 44: Pings desde Mallet a Bob y Alice

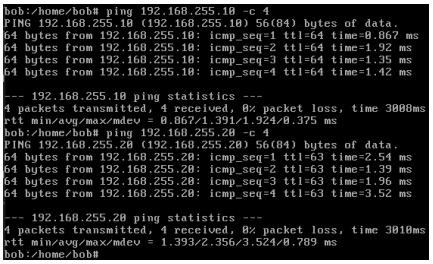

Figure 45: Pings Bob gateway **Privada** y a Alice

2.4.1 ¿Qué ocurre con el valor de `ttl` cuando un paquete atraviesa un router?

El time to live (`ttl`) se reduce en uno por cada router que el paquete atraviesa.
Recordemos que el `ttl` se utiliza para que no existan paquetes en la red que puedan viajar por ella de manera indefinida debido a problemas de enrutamiento, de tal manera que cuando llega a 0 este paquete se descarta enviando un mensaje de tipo **ICMP** *"time exceeded"*.

2.5 Iptables

Es una herramienta de filtrado de paquetes en sistemas Linux que permite configurar reglas para controlar el tráfico de red entrante, saliente y dentro del sistema. Actúa como un firewall sobre paquetes,direcciones IP, puertos, protocolos y otras características del tráfico.

2.5.1 Diferencias entre INPUT,OUTPUT y FORWARD

Cadena	Papel	Afecta a...
INPUT	Filtra el tráfico entrante destinado a la propia máquina. Se aplica cuando los paquetes llegan a las interfaces de red de la máquina.	El tráfico que llega a la máquina
OUTPUT	Filtra el tráfico saliente generado por la propia máquina, es decir, el tráfico que la máquina envía hacia otros dispositivos o redes.	El tráfico que sale de la máquina
FORWARD	Filtra el tráfico que pasa a través de la máquina, es decir, cuando la máquina actúa como un router o puente de red, pero el tráfico no está destinado a la máquina misma.	El tráfico que pasa por la máquina

Table 2: Cadenas de iptables

2.6 Configuración de las reglas de Firewall

Para esto, lo que tendremos que hacer es dentro de mallet jugar con las cadenas de iptable para poder generar los filtros deseados.

2.6.1 Realizar peticiones ICMP desde la red Pública (la interna)

```
mallet@mallet:/etc$ sudo iptables -L -n -v
Chain INPUT (policy ACCEPT 104 packets, 11044 bytes)
 pkts bytes target     prot opt in     out     source               destination
    2   168 DROP       icmp --  *      *       1.0.0.0/8            1.0.0.10
    4   336 ACCEPT     icmp --  *      *       192.168.255.0/24     1.0.0.0/8
    0     0 DROP       icmp --  *      *       1.0.0.0/8            1.0.0.10
    0     0 DROP       icmp --  *      *       1.0.0.0/8            192.168.255.10
    0     0 DROP       icmp --  *      *       1.0.0.0/8            192.168.255.0/24

Chain FORWARD (policy ACCEPT 20 packets, 1680 bytes)
 pkts bytes target     prot opt in     out     source               destination
    2   168 DROP       icmp --  *      *       0.0.0.0/0            192.168.255.0/24    icmp type 8
    0     0 DROP       icmp --  *      *       0.0.0.0/0            1.0.0.10            icmp type 8

Chain OUTPUT (policy ACCEPT 521 packets, 45712 bytes)
 pkts bytes target     prot opt in     out     source               destination
mallet@mallet:/etc$
```

Figure 46: Las reglas de Iptable

En este punto, comenzamos borrando todas las normas previas de iptables (por defecto en nuestra máquina, la política era aceptar todo tipo de paquetes).
Posteriormente debemos evitar 3 acciones (están enumeradas en el mismo orden que los comandos de iptables)

1. Evitar que desde la red Pública se pueda hacer un ping a cualquier dirección de la red Privada.
2. Evitar que cualquier dispositivo de la red pública pueda hacer un ping a la gateway de dicha red.
3. Evitar que cualquier dispositivo de la red Pública pueda hacer un ping a la gateway de la red Privada.

La segunda y tercera regla pueden parecer un poco absurdas, pero si nos paramos a pensarlo, si un atacante puede acceder a las gateways de la red, realizando un nmap, puede saber qué puertos y servicios están abiertos y a través de esa acción, con una conexión telnet, tomar acceso de manera remota a hosts de la red privada.
Adicionalmente, hemos rechazado estos paquetes con un `DROP` para que así, el atacante, no tenga información de si hay un firewall o que el host no está activo.

2.6.2 Comprobación de que los filtros funcionan

```
alice@alice:~$ 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=1.18 ms
64 bytes from 192.168.255.10: icmp_seq=2 ttl=64 time=0.752 ms

--- 192.168.255.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.752/0.969/1.186/0.217 ms
alice@alice:~$ 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=1.69 ms
64 bytes from 1.0.0.10: icmp_seq=2 ttl=64 time=1.20 ms

--- 1.0.0.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 1.207/1.452/1.698/0.248 ms
alice@alice:~$ 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=1.50 ms
64 bytes from 1.0.0.20: icmp_seq=2 ttl=63 time=1.37 ms

--- 1.0.0.20 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 1.379/1.442/1.505/0.063 ms
alice@alice:~$
```

Figure 47: Comprobamos que desde Alice, se pueden seguir realizando pings

```
bob:/home/bob# 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

bob:/home/bob# 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 1004ms

bob:/home/bob# 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 999ms

bob:/home/bob#
```

Figure 48: Comprobamos que las restricciones se han aplicado a Bob

```
mallet@mallet:/etc$ sudo iptables -L -n -v
Chain INPUT (policy ACCEPT 110 packets, 11548 bytes)
 pkts bytes target     prot opt in     out     source               destination
    4   336 DROP       icmp --  *      *       1.0.0.0/8            1.0.0.10
   10   840 ACCEPT     icmp --  *      *       192.168.255.0/24     1.0.0.0/8
    0     0 DROP       icmp --  *      *       1.0.0.0/8            1.0.0.10
    2   168 DROP       icmp --  *      *       1.0.0.0/8            192.168.255.10
    0     0 DROP       icmp --  *      *       1.0.0.0/8            192.168.255.0/24

Chain FORWARD (policy ACCEPT 32 packets, 2688 bytes)
 pkts bytes target     prot opt in     out     source               destination
    4   336 DROP       icmp --  *      *       0.0.0.0/0            192.168.255.0/24    icmp type 8
    0     0 DROP       icmp --  *      *       0.0.0.0/0            1.0.0.10            icmp type 8

Chain OUTPUT (policy ACCEPT 717 packets, 63938 bytes)
 pkts bytes target     prot opt in     out     source               destination
mallet@mallet:/etc$
```

Figure 49: Las reglas de iptables han recibido tráfico.

2.6.3 ¿Cómo conseguir que estas reglas sean permanentes?

Lo primero es generar un fichero `/etc/iptables.rules` de tal manera que guardaremos en él nuestras reglas. Posteriormente, hacemos que los scripts se encarguen de restaurar y almacenar las reglas de iptables.

```
mallet@mallet:/etc$ cat iptables.rules
# Generated by iptables-save v1.4.4 on Mon Dec  2 11:33:25 2024
*filter
:INPUT ACCEPT [1783:83671]
:FORWARD ACCEPT [24:2016]
:OUTPUT ACCEPT [2496:142830]
-A INPUT -d 1.0.0.10/32 -p icmp -m icmp --icmp-type 8 -j DROP
-A INPUT -d 192.168.255.10/32 -p icmp -m icmp --icmp-type 8 -j DROP
-A FORWARD -d 192.168.255.0/24 -p icmp -m icmp --icmp-type 8 -j DROP
COMMIT
# Completed on Mon Dec  2 11:33:25 2024
mallet@mallet:/etc$
```

Figure 50: `cat` del fichero que almacena las reglas establecidas hasta ahora

```
mallet@mallet:/etc/network/if-pre-up.d$ cat iptables-restore
#!/bin/sh
iptables-restore < /etc/iptables.rules
mallet@mallet:/etc/network/if-pre-up.d$ cat iptables-save
#!/bin/sh
iptables-save > /etc/iptables/rules.v4
mallet@mallet:/etc/network/if-pre-up.d$ sudo nano iptables-save
mallet@mallet:/etc/network/if-pre-up.d$
mallet@mallet:/etc/network/if-pre-up.d$ cat iptables-save
#!/bin/sh
iptables-save > /etc/iptables.rules
mallet@mallet:/etc/network/if-pre-up.d$ sudo nano iptables-load.sh
mallet@mallet:/etc/network/if-pre-up.d$ sudo chmod +x /etc/network/if-pre-up.d/i
ptables-load.sh
```

Figure 51: Configuración de los Scripts de iptables

Posteriormente comprobamos si las reglas se han guardado utilizando el comando `sudo iptables -L -n -v` obteniendo el siguiente resultado.

```
mallet@mallet:~$ iptables -L -n -v
iptables v1.4.4: can't initialize iptables table `filter': Permission denied (yo
u must be root)
Perhaps iptables or your kernel needs to be upgraded.
mallet@mallet:~$ sudo !!
sudo iptables -L -n -v
[sudo] password for mallet:
Chain INPUT (policy ACCEPT 36 packets, 5814 bytes)
 pkts bytes target     prot opt in     out     source               destination

    0     0 DROP       icmp --  *      *       1.0.0.0/8            192.168.255.
0/24
    0     0 DROP       icmp --  *      *       1.0.0.0/8            1.0.0.10

    0     0 DROP       icmp --  *      *       1.0.0.0/8            192.168.255.
10
    0     0 DROP       icmp --  *      *       1.0.0.0/8            192.168.255.
10

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

    0     0 ACCEPT     all  --  eth4   eth4    0.0.0.0/0            0.0.0.0/0

    0     0 ACCEPT     all  --  eth4   eth5    0.0.0.0/0            1.0.0.10

    0     0 ACCEPT     all  --  eth4   eth5    0.0.0.0/0            0.0.0.0/0


Chain OUTPUT (policy ACCEPT 40 packets, 5974 bytes)
 pkts bytes target     prot opt in     out     source               destination

mallet@mallet:~$
```

Figure 52: Caption

Al aparecer las normas, podemos afirmar que sí que se han guardado de manera permanente

2.6.4 Evitar que Bob haga conexiones ssh a Alice

Para este punto debemos añadir dos reglas más a iptable.

```
mallet@mallet:/etc$ sudo iptables -A FORWARD -i eth5 -o eth4 -p tcp --dport 22 -j DROP
mallet@mallet:/etc$ iptables -L -v -n
iptables v1.4.4: can't initialize iptables table `filter': Permission denied (you must be root)
Perhaps iptables or your kernel needs to be upgraded.
mallet@mallet:/etc$ sudo iptables -L -v -n
Chain INPUT (policy ACCEPT 110 packets, 11548 bytes)
 pkts bytes target     prot opt in     out     source               destination
    4   336 DROP       icmp --  *      *       1.0.0.0/8            1.0.0.10
   10   840 ACCEPT     icmp --  *      *       192.168.255.0/24     1.0.0.0/8
    0     0 DROP       icmp --  *      *       1.0.0.0/8            1.0.0.10
    2   168 DROP       icmp --  *      *       1.0.0.0/8            192.168.255.10
    0     0 DROP       icmp --  *      *       1.0.0.0/8            192.168.255.0/24

Chain FORWARD (policy ACCEPT 32 packets, 2688 bytes)
 pkts bytes target     prot opt in     out     source               destination
    4   336 DROP       icmp --  *      *       0.0.0.0/0            192.168.255.0/24    icmp type 8
    0     0 DROP       icmp --  *      *       0.0.0.0/0            1.0.0.10            icmp type 8
    0     0 DROP       tcp  --  eth5   eth4    0.0.0.0/0            0.0.0.0/0           tcp dpt:22

Chain OUTPUT (policy ACCEPT 894 packets, 80437 bytes)
 pkts bytes target     prot opt in     out     source               destination
mallet@mallet:/etc$
```

Figure 53: Con esta regla impedimos que la red pública pueda hacer conexiones ssh a la red privada.

```
mallet@mallet:/etc$ sudo iptables -A FORWARD -i eth5 -o eth4 -p tcp --dport 22 -s 192.168.255.20 -j ACCEPT
mallet@mallet:/etc$ sudo iptables -A FORWARD -i eth4 -o eth5 -p tcp --dport 22 -s -j DROP
Bad argument `DROP'
Try `iptables -h' or 'iptables --help' for more information.
mallet@mallet:/etc$ sudo iptables -A FORWARD -i eth4 -o eth5 -p tcp --dport 22 -j DROP
mallet@mallet:/etc$
```

Figure 54: Solo permitimos que Alice haga los ssh en la red pública.

En este caso, como debemos impedir que todos los dispositivos de la red Privada no puedan recibir sshs de la red Pública, lo que hemos hecho ha sido aplica un filtro a la interfaz de la red Pública por lo que todo lo que pase por la gateway de la red Privada hacía el puerto 22 automáticamente lo desechamos. Vamos a ver si hemos conseguido configurar bien las iptables:

```
alice@alice:~$ 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: Tue Dec  3 10:49:43 2024
bob@bob:~$
```

Figure 55: Alice puede hacer un ssh sin ningún tipo de problema.

```
bob:/home/bob# ssh alice@192.168.255.20
ssh: connect to host 192.168.255.20 port 22: Connection timed out
bob:/home/bob# _
```

Figure 56: Bob, no puede hacer un ssh a Alice.