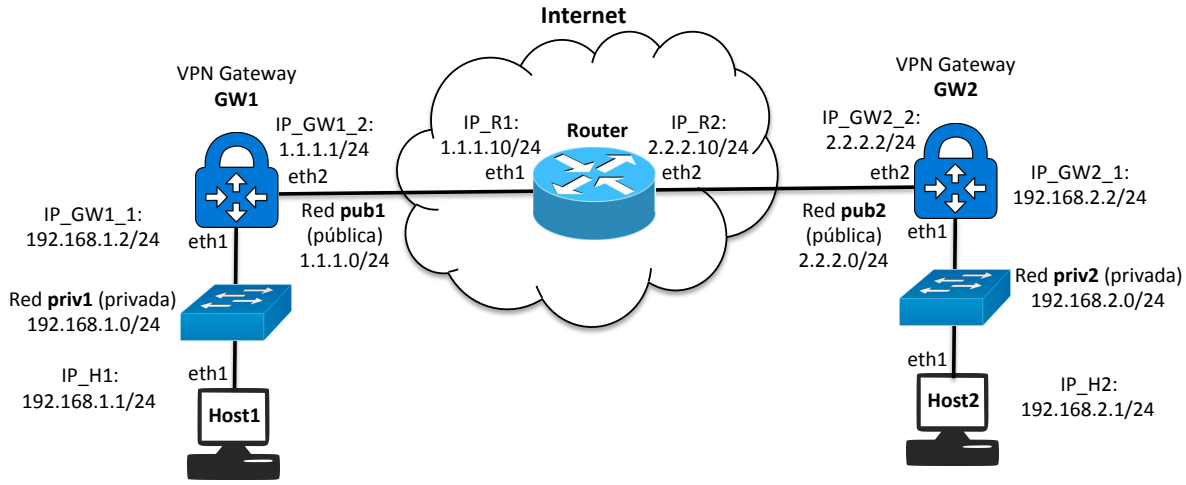
Seguridad en Redes

Práctica 4.5. IPsec y OpenSSH

A. IPsec

A.1 Preparación del entorno

En esta primera parte de la práctica vamos a implementar una VPN de tipo *site-to-site* basada en IPsec (módo túnel). Para ello, vamos a usar cinco MVs (`router`, `gw1`, `gw2`, `host1` y `host2`), dos redes internas que emulan redes privadas (`priv1` y `priv2`), y otras dos redes internas que emulan redes públicas (`pub1` y `pub2`):



Importa una MV, haz 4 clonaciones enlazadas y añade uno o dos interfaces de red a cada MV, según sea necesario, conectados a la red correspondiente, según la figura anterior.

Configura `router`:

```
sudo ifdown eth0
sudo ip link set dev eth1 up
sudo ip link set dev eth2 up
sudo ip addr add 1.1.1.10/24 broadcast + dev eth1
sudo ip addr add 2.2.2.10/24 broadcast + dev eth2
sudo sysctl -w net.ipv4.ip_forward=1
```

Configura `gw1`:

```
sudo apt-get update
sudo apt-get install ssh strongswan
sudo ifdown eth0
sudo ip link set dev eth1 up
sudo ip link set dev eth2 up
sudo ip addr add 192.168.1.2/24 broadcast + dev eth1
sudo ip addr add 1.1.1.1/24 broadcast + dev eth2
sudo ip route add default via 1.1.1.10
sudo sysctl -w net.ipv4.ip_forward=1
```

Configura `gw2`:

```
sudo apt-get update
sudo apt-get install ssh strongswan
sudo ifdown eth0
sudo ip link set dev eth1 up
sudo ip link set dev eth2 up
sudo ip addr add 192.168.2.2/24 broadcast + dev eth1
sudo ip addr add 2.2.2.2/24 broadcast + dev eth2
sudo ip route add default via 2.2.2.10
sudo sysctl -w net.ipv4.ip_forward=1
```

Configura `host1`:

```
sudo ifdown eth0
sudo ip link set dev eth1 up
sudo ip addr add 192.168.1.1/24 broadcast + dev eth1
sudo ip route add default via 192.168.1.2
```

Configura `host2`:

```
sudo ifdown eth0
sudo ip link set dev eth1 up
sudo ip addr add 192.168.2.1/24 broadcast + dev eth1
sudo ip route add default via 192.168.2.2
```

A.2. Configuración manual de IPsec (sin usar IKE)

Este ejercicio consiste en configurar una VPN de tipo *site-to-site* basada en IPsec (modo túnel) entre las dos pasarelas VPN (GW1 y GW2) sin utilizar el protocolo IKE para intercambio de claves. Para ello es necesario configurar las asociaciones de seguridad (SAs) y las políticas de seguridad (SPs) de forma manual en ambas pasarelas. Igualmente, será necesario generar las claves de cifrado y de autenticación también de forma manual (ambos extremos deben usar las mismas claves).

En este ejercicio se usarán claves de cifrado de 192 bits (24 bytes) y claves de autenticación para HMAC de 128 bites (16 bytes). Para generar claves aleatorias en formato hexadecimal, de 24 y 16 bytes, respectivamente, se pueden usar los siguientes comandos:

```
$ dd if=/dev/random count=24 bs=1| xxd -ps
$ dd if=/dev/random count=16 bs=1| xxd -ps
```

Importante: como se trata de claves hexadecimales, hay que añadir el prefijo `0x` delante de dichas claves.

A continuación es necesario definir las asociaciones y políticas de seguridad en el archivo `/etc/ipsec-tools.conf` de ambas pasarelas.

Las asociaciones de seguridad (SAs) realizarán el encapsulado de seguridad ESP de IPsec con cifrado de tipo `3des-cbc` y autenticación de tipo `hmac-md5`. Será necesario definir dos asociaciones de seguridad, una por cada sentido de la comunicación. Será necesario, por tanto, generar dos claves aleatorias de 192 bits para el cifrado y otras dos claves de 128

bits para la autenticación. Adicionalmente, cada asociación de seguridad se identificará con un SPI distinto (Security Parameters Index), en nuestro ejemplo se utilizan los valores de SPI `0x201` y `0x301`, respectivamente.

Un ejemplo de archivo de configuración para la pasarela `GW1` sería el siguiente:

$ cat /etc/ipsec-tools.conf

```
#!/usr/sbin/setkey -f
# Vaciar las SAD y SPD
flush;
spdflush;

# Definir asociacionnes de seguridad (SAs) para ESP
# realizando cifrado con claves de 192 bit (algoritmo 3des-cbc)
# autenticación empleando claves de 128 bits (algoritmo hmac-md5)

add 1.1.1.1 2.2.2.2 esp 0x201 -m tunnel
     -E 3des-cbc 0x7aeaca3f87d060a12f4a4487d5a5c3355920fae69a96c831
     -A hmac-md5 0xc0291ff014dccdd03874d9e8e4cdf3e6;

add 2.2.2.2 1.1.1.1 esp 0x301 -m tunnel
     -E 3des-cbc 0xf6ddb555acfd9d77b03ea3843f2653255afe8eb5573965df
     -A hmac-md5 0x96358c90783bbfa3d7b196ceabe0536b;

# Definir politicas de seguridad (SPs)
spdadd 192.168.1.0/24 192.168.2.0/24 any -P out ipsec
     esp/tunnel/1.1.1.1-2.2.2.2/require;

spdadd 192.168.2.0/24 192.168.1.0/24 any -P in ipsec
     esp/tunnel/2.2.2.2-1.1.1.1/require;
```

El archivo de la pasarela `GW2` sería similar, pero cambiando las políticas de seguridad (*in* y *out* intercambiados):

$ cat /etc/ipsec-tools.conf

```
#!/usr/sbin/setkey -f
# Vaciar las SAD y SPD
flush;
spdflush;

# Definir asociacionnes de seguridad (SAs) para ESP
# realizando cifrado con claves de 192 bit (algoritmo 3des-cbc)
# autenticación empleando claves de 128 bits (algoritmo hmac-md5)

add 1.1.1.1 2.2.2.2 esp 0x201 -m tunnel
     -E 3des-cbc 0x7aeaca3f87d060a12f4a4487d5a5c3355920fae69a96c831
     -A hmac-md5 0xc0291ff014dccdd03874d9e8e4cdf3e6;

add 2.2.2.2 1.1.1.1 esp 0x301 -m tunnel
     -E 3des-cbc 0xf6ddb555acfd9d77b03ea3843f2653255afe8eb5573965df
```

```
      -A hmac-md5 0x96358c90783bbfa3d7b196ceabe0536b;

# Definir politicas de seguridad (SPs)
spdadd 192.168.2.0/24 192.168.1.0/24 any -P out ipsec
      esp/tunnel/2.2.2.2-1.1.1.1/require;

spdadd 192.168.1.0/24 192.168.2.0/24 any -P in ipsec
      esp/tunnel/1.1.1.1-2.2.2.2/require;
```

Una vez definidos los archivos de configuración en ambas máquinas, ejecuta `ipsec` en ambos extremos con la siguiente orden:

```
$ sudo /etc/init.d/setkey start
```

Entrega #1. Entrega los archivos de configuración `/etc/ipsec-tools.conf` de `gw1` y `gw2`.

Para ver que la VPN funciona, podemos establecer una conexión TCP entre las máquinas `host1` y `host2`, por ejemplo, usando `netcat` y generar tráfico entre ambas máquinas.

Ejecuta `wireshark` en el router y observa que todo el tráfico entre `host1` y `host2` que pasa por el router aparece encapsulado en paquetes ESP (observar las direcciones IP origen y destino de dichos paquetes). Analiza los paquetes ESP.

Observa las políticas y asociaciones de seguridad establecidas en `gw1` y `gw2` con las siguientes órdenes:

```
$ sudo ip xfrm policy
$ sudo ip xfrm state
```

Entrega #2. Copia y entrega las salidas de los comandos anteriores en `gw1`.

Una vez finalizada esta parte de la práctica, debes parar el servicio `ipsec` en ambos extremos con la siguiente orden:

```
$ sudo /etc/init.d/setkey stop
```

A.3. Configuración de IPsec usando IKE con clave secreta pre-compartida (psk)

En esta práctica implementaremos el mismo tipo de VPN que en el caso anterior, pero usaremos el protocolo IKE para intercambio automático de claves basado en una clave secreta pre-compartida (psk, *pre-shared key)*.

Se utilizará la implementación de IKE basada en StrongSwan. Para más información consultar la página web (https://www.strongswan.org), la página de manual del comando `ipsec`, que ofrece toda la funcionalidad de strongSwan y su archivo de configuración (`/etc/ipsec.conf`)

Configura la clave secreta, añadiendo al final del fichero `/etc/ipsec.secrets` de `gw1` la siguiente línea:

```
: PSK "Clave secreta muy segura"
```

Haz lo mismo en `gw2`:

```
: PSK "Clave secreta muy segura"
```

Normalmente, se usaría una clave generada aleatoriamente. Para ello, en lugar de una cadena, se puede indicar una secuencia de dígitos hexadecimales (comenzando con `0x`) o datos binarios codificados en Base64 (comenzando con `0s`).

En la configuración, `gw1` será el extremo izquierdo y `gw2`, el derecho. Eso permite tener la misma configuración en ambos extremos de la VPN. Sin embargo, la documentación de strongSwan sugiere denominar izquierdo al extremo local y derecho al remoto (aprovechando que, en inglés, comienzan por la misma letra).

Configura la VPN, añadiendo al fichero `/etc/ipsec.conf` de ambos extremos las siguientes líneas:

```
conn secret
      left=1.1.1.1
      leftsubnet=192.168.1.0/24
      leftauth=psk
      right=2.2.2.2
      rightsubnet=192.168.2.0/24
      rightauth=psk
      type=tunnel
      auto=start
```

Con `auto=start`, la conexión VPN se iniciaría automáticamente, lo cual es necesario si se pretende que la conexión sea permanente.

Adicionalmente, en el archivo de configuración anterior, se podrían incluir los algoritmos de cifrado específicos que usarán los protocolos ESP e IKE, por ejemplo:

```
      esp=aes128-sha256
      ike=aes128-sha256-modp3072
```

(si no se especifican estos parámetros, se usan los algoritmos de cifrado establecidos por defecto)

Reinicia el servicio en ambos extremos:

```
$ sudo ipsec restart
```

Ejecuta `wireshark` en el `router`.

Inicia la conexión VPN en `gw1`:

```
$ sudo ipsec up secret
```

Para ver que la VPN funciona, podemos establecer una conexión TCP entre las máquinas `host1` y `host2`, por ejemplo, usando `netcat` y generar tráfico entre ambas máquinas. Analiza los paquetes ISAKMP y ESP capturados por `wireshark` en el `router`.

Revisa el fichero de registro `/var/log/daemon.log`.

Entrega #3. Copia y entrega los nuevos registros del archivo `/var/log/daemon.log` de `gw1`.

Observa los detalles de la conexión en `gw1` y `gw2` con:

```
$ sudo ipsec status
$ sudo ipsec statusall
```

Observar las políticas y asociaciones de seguridad en `gw1` y `gw2` con:

```
$ sudo ip xfrm policy
$ sudo ip xfrm state
```

Entrega #4. Copia y entrega las salidas de los dos comandos anteriores en `gw1`.

Configura Wireshark para que descifre los paquetes ESP y compruebe su autenticidad.

Para ello, en las preferencias del protocolo ESP (Edit → Preferences… → Protocols → ESP), activa todas las casillas y añade los parámetros de las asociaciones de seguridad.

Una vez finalizada esta parte de la práctica, finaliza la conexión VPN en `gw1` con la siguiente orden:

```
$ sudo ipsec down secret
```

A.4. Configuración de IPsec usando IKE con certificados autofirmados

Repetiremos la misma configuración que en la práctica anterior, pero en este caso, en lugar de usar una clave secreta pre-compartida, usaremos certificados autofirmados para generar la clave. Para generar estos certificados autofirmados, no usaremos OpenSSL, sino que usaremos una utilidad que proporciona IPsec (`ipsec pki`)

Crea una clave RSA y un certificado autofirmado en `gw1`:

```
$ sudo sh -c "ipsec pki --gen > /etc/ipsec.d/private/gw1-key.der"
$ sudo sh -c "ipsec pki --self --in /etc/ipsec.d/private/gw1-key.der
--dn "CN=gw1" > /etc/ipsec.d/certs/gw1-cert.der"
```

Haz lo mismo en `gw2`:

```
$ sudo sh -c "ipsec pki --gen > /etc/ipsec.d/private/gw2-key.der"
$ sudo sh -c "ipsec pki --self --in /etc/ipsec.d/private/gw2-key.der
--dn "CN=gw2" > /etc/ipsec.d/certs/gw2-cert.der"
```

Dado que son certificados autofirmados, deben estar accesibles localmente, ya que no se confiará en ningún certificado de este tipo intercambiado por la red. Ejecuta el siguiente comando en `gw1` para copiar el certificado de `gw2`:

```
$ sudo scp usuario@2.2.2.2:/etc/ipsec.d/certs/gw2-cert.der
/etc/ipsec.d/certs/
```

A continuación, ejecuta el siguiente comando en `gw2` para copiar el certificado de `gw1`:

```
$ sudo scp usuario@1.1.1.1:/etc/ipsec.d/certs/gw1-cert.der
/etc/ipsec.d/certs/
```

Configura las claves privadas, añadiendo al final del fichero `/etc/ipsec.secrets` de `gw1` la siguiente línea:

```
: RSA gw1-key.der
```

(borra la entrada `PSK` que añadiste en el apartado anterior)

Haz lo mismo en `gw2`:

```
: RSA gw2-key.der
```

(borra la entrada `PSK` que añadiste en el apartado anterior)

Configura la VPN, añadiendo al fichero `/etc/ipsec.conf` de ambos extremos las siguientes líneas:

```
conn sscert
      left=1.1.1.1
      leftsubnet=192.168.1.0/24
      leftcert=gw1-cert.der
      leftid="CN=gw1"
      right=2.2.2.2
      rightsubnet=192.168.2.0/24
      rightcert=gw2-cert.der
      rightid="CN=gw2"
      type=tunnel
      auto=start
```

Reinicia el servicio en ambos extremos:

```
$ sudo ipsec restart
```

Ejecuta `wireshark` en el `router`.

Inicia la conexión VPN en uno de los extremos:

```
$ sudo ipsec up sscert
```

Para ver que la VPN funciona, podemos establecer una conexión TCP entre las máquinas `host1` y `host2`, por ejemplo, usando `netcat` y generar tráfico entre ambas máquinas. Analiza los paquetes ISAKMP y ESP capturados por `wireshark` en el `router`.

Revisa el fichero de registro `/var/log/daemon.log` de `gw1` y `gw2`.

Entrega #5. Copia y entrega los nuevos registros del archivo `/var/log/daemon.log` de `gw1`.

Observa los detalles de la conexión en `gw1` y `gw2` con:

```
$ sudo ipsec status
$ sudo ipsec statusall
```

Observa las políticas y asociaciones de seguridad en `gw1` y `gw2` con:

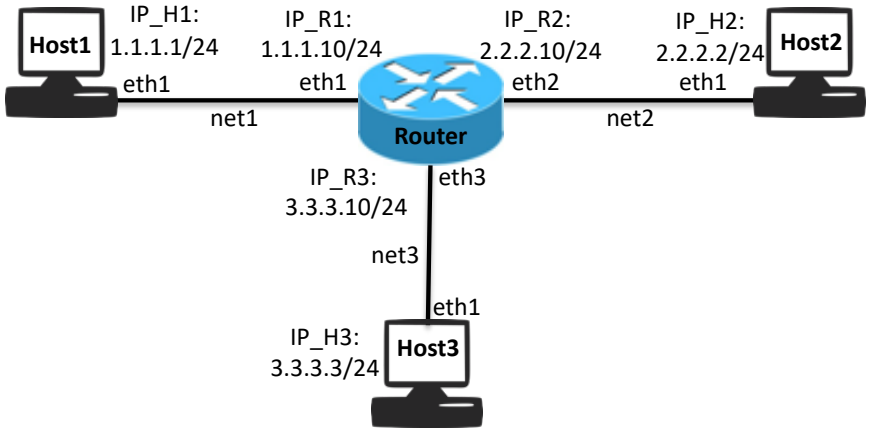
```
$ sudo ip xfrm policy
$ sudo ip xfrm state
```

Entrega #6. Copia y entrega las salidas de los dos comandos anteriores en `gw1`.

B. OpenSSH

B.1. Preparación del entorno

En esta segunda parte de la práctica vamos a aprender el uso de OpenSSH y la creación de túneles SSH mediante reenvío de puertos (*port forwarding*). Para ello, vamos a usar cuatro MVs (`host1`, `host2`, `host3` y `router`) y tres redes internas (`net1`, `net2` y `net3`), tal y como se muestra en la figura.



Importa una MV, haz 3 clonaciones enlazadas. Añade una interfaz de red a cada uno de los hosts (`host1`, `host2`, `host3`) y tres interfaces de red a `router`, cada interfaz conectada a la red correspondiente, según la figura anterior.

Configura `router`:

```
sudo ifdown eth0
sudo ip link set dev eth1 up
sudo ip link set dev eth2 up
sudo ip link set dev eth3 up
sudo ip addr add 1.1.1.10/24 broadcast + dev eth1
sudo ip addr add 2.2.2.10/24 broadcast + dev eth2
sudo ip addr add 3.3.3.10/24 broadcast + dev eth3
sudo sysctl -w net.ipv4.ip_forward=1
```

Configura `host1`:

```
sudo apt-get update
sudo apt-get install ssh apache2
sudo ifdown eth0
sudo ip link set dev eth1 up
sudo ip addr add 1.1.1.1/24 broadcast + dev eth1
sudo ip route add default via 1.1.1.10
```

Edita la página de inicio `/var/www/index.html` del servidor web Apache2 y sustituye su contenido por el siguiente:

```
<html><body>
<h1>Web server running on host1 (1.1.1.1)</h1>
</body></html>
```

Configura `host2`:

```
sudo apt-get update
sudo apt-get install ssh
sudo ifdown eth0
sudo ip link set dev eth1 up
sudo ip addr add 2.2.2.2/24 broadcast + dev eth1
sudo ip route add default via 2.2.2.10
```

Configura `host3`:

```
sudo apt-get update
sudo apt-get install apache2
sudo ifdown eth0
sudo ip link set dev eth1 up
sudo ip addr add 3.3.3.3/24 broadcast + dev eth1
sudo ip route add default via 3.3.3.10
```

Edita la página de inicio `/var/www/index.html` del servidor web Apache2 y sustituye su contenido por el siguiente:

```
<html><body>
<h1>Web server running on host3 (3.3.3.3)</h1>
</body></html>
```

B.2. Métodos de autenticación en OpenSSH

Los dos principales métodos de autenticación en OpenSSH son por contraseña o por clave pública. El método de autenticación por contraseña permite establecer una sesión (shell) segura en una máquina remota introduciendo el nombre de usuario y la contraseña de un usuario existente en la máquina remota. En este primer ejercicio, la máquina `host1` actuará como cliente SSH y la máquina `host2` como servidor SSH.

Para conectarse a `host2` (servidor SSH) desde `host1` (cliente SSH) con el *login* "`usuario`" ejecuta en `host1` el siguiente comando:

```
$ ssh usuario@2.2.2.2
The authenticity of host '2.2.2.2 (2.2.2.2)' can't be established.
ECDSA key fingerprint is c9:a6:24:ea:1d:7a:83:8c:bc:41:3f:bf:1a:79:89:ce.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '2.2.2.2' (ECDSA) to the list of known hosts.
usuario@2.2.2.2's password:
```

En la salida del comando anterior, en primer lugar nos pregunta si confiamos en la identidad del servidor (nos muestra su huella digital o *fingerprint* generada a partir de la clave pública del servidor) y luego nos solicita la contraseña del usuario remoto.

Para ver el proceso de autenticación, usa la opción `-v` (*verbose*).

Para implementar el segundo método de autenticación, basado en clave pública, es necesario generar un par de claves pública/privada para el usuario en la máquina que actúa

como cliente SSH. Para ello ejecuta `ssh-keygen` en la máquina `host1` seleccionando la ubicación por defecto para la clave privada y proporciona una contraseña (*passphrase*) para proteger dicha clave. Se creará un fichero `~/.ssh/id_rsa` con la clave privada y un fichero `~/.ssh/id_rsa.pub` con la clave pública correspondiente.

Añade el contenido del fichero de clave pública al fichero `~/.ssh/authorized_keys` de `host2` (sevidor SSH) con:

```
$ ssh-copy-id 2.2.2.2
```

(El fichero `~/.ssh/authorized_keys` debe tener permisos `600`)

Accede con `ssh` a `host2` desde `host1`:

```
$ ssh usuario@2.2.2.2
```

Pedirá la contraseña (*passphrase*) para acceder a la clave privada, pero la autenticación se realizará mediante la clave pública. Para ver el proceso de autenticación, usa la opción `-v` (*verbose*).

Entrega #7: Copia la salida del comando anterior con la opción `-v`.

Para escribir la contraseña de la clave privada una sola vez, ejecuta en `host1`:

```
$ eval `ssh-agent`
$ ssh-add
```

IMPORTANTE: Al terminar esta parte de la práctica, debes finalizar la conexión `ssh` establecida entre `host1` y `host2`, ejecutando el comando `exit`.

En los siguientes ejercicios vamos a utilizar únicamente la autenticación por contraseña, por tanto, antes de continuar, vamos a borrar todos los ficheros de claves `ssh` creados en `host1` y en `host2`, ejecutando el siguiente comando en ambas máquinas:

```
$ rm -r /home/usuario/.ssh/*
```

B.3. Reenvío de puertos (*port fowarding*) local

La opción `-L port:dst_host:dst_port` especifica el reenvío de un puerto **local** a un destino fijo. Es decir, las conexiones al puerto (local) `port` del cliente SSH, se reenvían sobre el canal seguro establecido con el servidor SSH y de ahí, ya sin seguridad, hasta el puerto `dst_port` de `dst_host`.

El ejemplo siguiente es un caso de uso típico del reenvío de puertos local. Imaginemos que `host1` (que actúa como cliente SSH) está conectado a una red muy segura configurada con un cortafuegos que únicamente permite conexiones SSH **salientes**. `host1` quiere conectarse a un servidor web público, implementado por `host3`, y tiene acceso a un servidor SSH ubicado fuera de su red, implementado por `host2`. Por tanto, las máquinas tienen los siguientes roles:

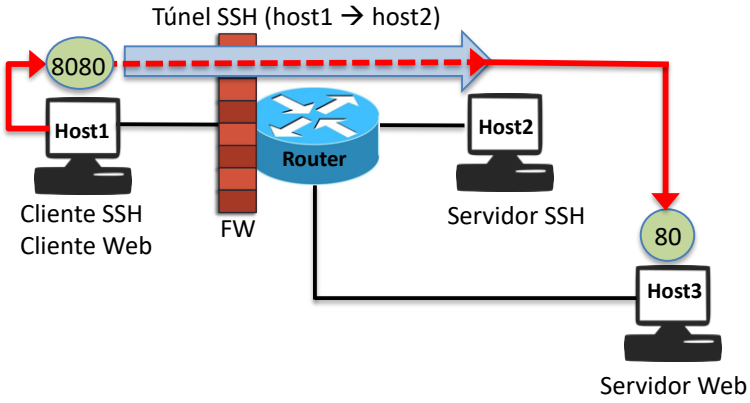
`host1 (1.1.1.1)` → cliente SSH y cliente web
`host2 (2.2.2.2)` → servidor SSH
`host3 (3.3.3.3)` → servidor web

Primero configura el cortafuegos `iptables` en `router`:

1. Define las políticas por defecto (rechazar todo)
```
$sudo iptables -P INPUT DROP
$sudo iptables -P OUTPUT DROP
$sudo iptables -P FORWARD DROP
```
2. Permite todos los paquetes que pertenezcan a conexiones establecidas o relacionadas:
```
$sudo iptables -A FORWARD -m state --state RELATED,ESTABLISHED
-j ACCEPT
```
3. Permite todas la comunicaciones entre las redes `net2` y `net3` (esto habilita todas las comunicaciones entre `host2` y `host3`)
```
$sudo iptables -A FORWARD -i eth2 -o eth3 -j ACCEPT
$sudo iptables -A FORWARD -i eth3 -o eth2 -j ACCEPT
```
4. Permite las conexiones salientes desde la red `net1` dirigidas al puerto `22/ssh` (esto habilita las conexiones SSH salientes desde `host1`)
```
$sudo iptables -A FORWARD -i eth1 -p tcp -m tcp --dport 22
-m state --state NEW -j ACCEPT
```

Comprueba que el servidor web de `host3` es accesible desde `host2`, pero no desde `host1` (para ello abre el navegador web en ambas máquinas e introduce la URL `http://3.3.3.3`)

Para permitir la conexión de `host1` al servidor web, establece un túnel SSH entre `host1` y `host2`, para reenviar el puerto (local) `8080` del cliente SSH al puerto `80` del servidor web (`host3`), tal y como se muestra en la siguiente figura:



Para ello, ejecuta en `host1` el siguiente comando:
```
$ ssh -v -N -L 8080:3.3.3.3:80 usuario@2.2.2.2
```

La opción `-N` evita que se lance una sesión interactiva de *shell* remota, lo cual es útil si solo se quiere establecer el reenvío de puertos. También se suele usar la opción `-f`, que hace que el comando `ssh` se desasocie del terminal y pase a segundo plano.

Entrega #8: Copia la salida del comando `ssh` anterior con la opción `-v`.

A continuación, arranca el navegador web en `host1` e introduce la URL `http://localhost:8080`. Comprueba que se accede a la página del servidor web ubicado en host3.

IMPORTANTE: Al terminar esta parte de la práctica, aborta la conexión `ssh` establecida entre `host1` y `host2` mediante `^C`

B.3. Reenvío de puertos (*port fowarding*) remoto

La opción `-R port:dst-host:dst-port` especifica el reenvío de un puerto **remoto** a un destino fijo. Es decir, las conexiones al puerto (remoto) `port` del servidor SSH, se reenvían sobre el canal seguro establecido con el cliente SSH y de ahí, ya sin seguridad, hasta el puerto `dst-port` de `dst-host`.

Vamos a ver dos casos de uso típicos del reenvío de puertos remoto.

Caso 1. El primer caso de uso es similar al estudiado anteriormente, en el que `host1` (que en este caso actuará como servidor SSH) está conectado a una red muy segura configurada con un cortafuegos, pero en este caso permite únicamente conexiones SSH **entrantes**. El `host1` quiere conectarse a un servidor web público, implementado por `host3`, y existe una máquina ubicada fuera de su red, implementado por `host2` (cliente SSH) que puede conectarse a `host1` mediante SSH. Por tanto, las máquinas tienen los siguientes roles:

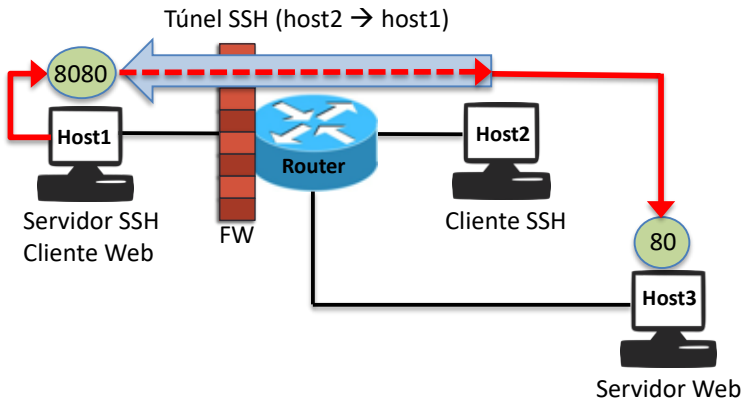
`host1 (1.1.1.1)` → servidor SSH y cliente web
`host2 (2.2.2.2)` → cliente SSH
`host3 (3.3.3.3)` → servidor web

Primero cambia las reglas del cortafuegos `iptables` en `router`:

1. Elimina la regla que permite las conexiones salientes desde la red `net1` dirigidas al puerto `22/ssh`
   ```
   $sudo iptables -D FORWARD -i eth1 -p tcp -m tcp --dport 22
   -m state --state NEW -j ACCEPT
   ```
2. Añade una regla que permite las conexiones entrantes hacia la red `net1` dirigidas al puerto `22/ssh` (esto habilita las conexiones SSH entrantes hacia `host1`)
   ```
   $sudo iptables -A FORWARD -o eth1 -p tcp -m tcp --dport 22
   -m state --state NEW -j ACCEPT
   ```

Comprueba que el servidor web de `host3` no es accesible desde `host1` (para ello abre el navegador web en ambas máquinas e introduce la URL `http://3.3.3.3`)

Para permitir la conexión de `host1` al servidor web, establece un túnel SSH entre `host2` y `host1`, para reenviar el puerto (remoto) `8080` del servidor SSH (`host1`) al puerto `80` del servidor web (`host3`), tal y como se muestra en la siguiente figura:



Para ello, ejecuta en `host2` el siguiente comando:

```
$ ssh -v -N -R 8080:3.3.3.3:80 usuario@1.1.1.1
```

Entrega #9: Copia la salida del comando `ssh` anterior con la opción `-v`.

A continuación, arranca el navegador web en `host1` e introduce la URL `http://localhost:8080`. Comprueba que se accede a la página del servidor web ubicado en `host3`.

IMPORTANTE: Al terminar esta parte de la práctica, aborta la conexión `ssh` establecida entre `host2` y `host1` mediante `^C`

Caso 2. El segundo caso de uso del renvío de puertos remoto es una situación en la que tenemos un servidor (por ejemplo, un servidor web) ubicado dentro una red protegida por un cortafuegos que sólo permite conexiones SSH salientes. Queremos hacer que el servidor web interno (implementado por `host1`) sea visible desde Internet. Esta máquina tiene acceso a un servidor SSH ubicado fuera de su red (implementado por `host2`). En este caso `host3` actuará como un cliente web de Internet. Por tanto, las máquinas tienen los siguientes roles:

`host1 (1.1.1.1)` → cliente SSH y servidor web
`host2 (2.2.2.2)` → servidor SSH
`host3 (3.3.3.3)` → cliente web

Cambia de nuevo las reglas del cortafuegos `iptables` en `router`:

1. Elimina la regla que permite las conexiones entrantes desde la red `net1` dirigidas al puerto `22/ssh`

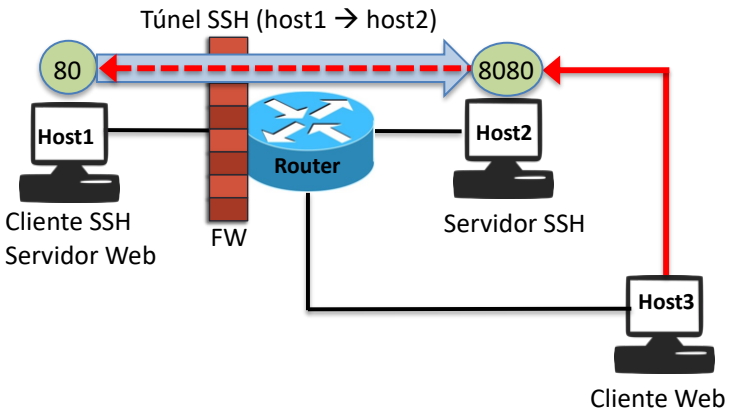
```
$sudo iptables -D FORWARD -o eth1 -p tcp -m tcp --dport 22
-m state --state NEW -j ACCEPT
```

2. Añade una regla que permite las conexiones salientes hacia la red `net1` dirigidas al puerto `22/ssh`

```
$sudo iptables -A FORWARD -i eth1 -p tcp -m tcp --dport 22
-m state --state NEW -j ACCEPT
```

Comprueba que el servidor web de `host1` no es accesible desde `host3` (para ello abre el navegador web en `host3` e introduce la URL `http://1.1.1.1`)

Para permitir la conexión de `host3` al servidor web, establece un túnel SSH entre `host1` y `host2`, para reenviar el puerto (remoto) `8080` del servidor SSH (`host2`) al puerto `80` del propio cliente SSH (`host1`), tal y como se muestra en la siguiente figura:



Para ello, ejecuta en `host1` el siguiente comando:

```
$ ssh -v -N -R 8080:localhost:80 usuario@2.2.2.2
```

Entrega #10: Copia la salida del comando `ssh` anterior con la opción `-v`.

A continuación, arranca el navegador web en `host3` y conéctate al puerto `8080` del `host2`, es decir, introduce la URL `http://2.2.2.2:8080`. Comprueba que `host3` accede a la página del servidor web ubicado en `host1`.