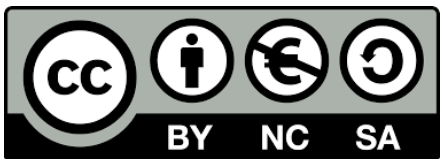
SSH

Antonio Boronat Pérez

SMX2 – Servicios en Red



Índice

Introducción....3

Instalar OpenSSH....3

Conexión remota....4

Configuración del servidor SSHD....5

Autenticación de usuarios....6

Autenticación con clave de cifrado....6

Reenvío X11 (X-Windows)....10

Transferencia segura de ficheros....11

Reenvío de puertos TCP / SSH Tunneling....11

Red Privada Virtual (VPN)....16

Introducción

La definición de **SSH** (*Secure Shell*), según la Wikipidia , es un protocolo y programa que tiene como principal función dar acceso remoto a un servidor usando un canal de comunicación seguro en el que la información transmitida está cifrada, es decir SSH viene a sustituir el servicio Telnet que transmite la información en texto plano. Además, este servicio permite el intercambio seguro de ficheros mediante un FTP seguro mediante cifrado, gestión de claves de cifrado, creación de túneles y la conexión de sesiones X-Windows para ejecutar programas gráficos en sistemas remotos.

El servicio SSH es estándar de forma que podemos encontrar diferentes implementaciones y está disponible para diferentes sistemas operativos. Entre las implementaciones la más popular es el proyecto OpenSSH.

Instalar OpenSSH

Este servicio comunica un cliente con un servidor, por tanto en aquellos sistemas a los que deseemos acceder remotamente de forma segura debemos instalar la aplicación servidora con:

```
# apt install openssh-server
```

en este proceso instalará las aplicaciones para el acceso remoto y para el intercambio seguro de ficheros y se generan las claves, pública y privada, necesarias para establecer las conexiones seguras con los clientes de estos servicios. Concretamente se instalan: *openssh-server, openssh-sftp-server y ssh-import-id.*

La aplicación cliente *ssh* suele estar instalada por defecto.

La gestión del servicio, comprobar el estado, arranque, parada y reinicio con las órdenes:

```
# systemctl start [ stop | restart | status ] ssh
```

```
# service ssh start [ stop | restart | status ]
```

El nombre del proceso asociado a este servicio es *sshd*.

Conexión remota

Como hemos comentado la aplicación *ssh* sustituye a *telnet*, en los dos casos el servicio tiene una arquitectura cliente-servidor, en el caso de *telnet* el puerto usado es el 21 mientras que *ssh* usa el 22.

Al establecer una de estas conexiones obtenemos una terminar en el sistema remoto al que nos hemos conectado, la diferencia en los dos es que con *ssh* la conexión es segura mediante cifrado de forma que si se intercepta la comunicación su contenido no podrá ser recuperado.

Para establecer una conexión remota ejecutamos en el cliente:

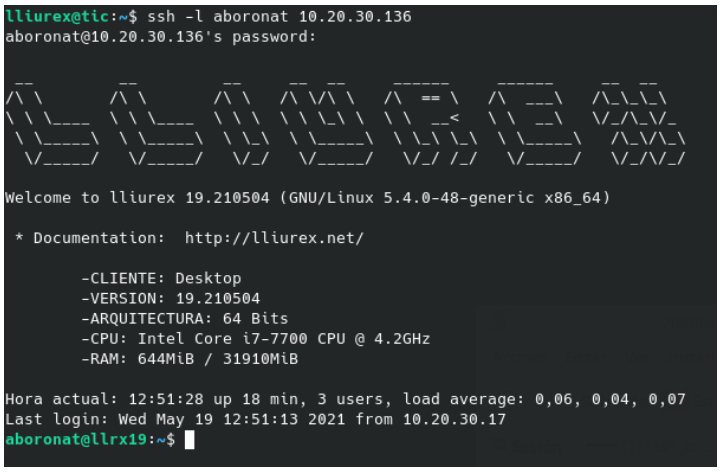
```
$ ssh  [opciones]   IP o nombre_sistema_remoto
```

la primera vez que nos conectemos a un sistema remoto nos mostrará la huella digital (*fingerprint*) de este sistema y nos preguntará si deseamos conectarnos, en caso afirmativo, guarda esta información en *~/.ssh/known_hosts* y no nos preguntará mientras no cambie este valor o lo eliminemos del fichero.

Entre las opciones una que debemos indicar con frecuencia es el usuario con el que hacemos login en el sistema remoto y del que debemos conocer su contraseña en ese sistema.

```
$ ssh ..[opciones] usuario_remoto@IP o nombre_sistema_remoto
```

```
$ ssh  -l usuario_remoto   IP o nombre_sistema_remoto
```



ahora estamos en una terminal del sistema remoto y las órdenes se ejecutarán en ese sistema remoto.

Las dos órdenes anteriores son equivalentes.

Para finalizar la conexión escribimos *exit* y volvemos a la terminal local en la que iniciamos la sesión remota.

Configuración del servidor SSHD

La configuración básica de un servidor OpenSSH es sencilla, y la encontramos en le fichero */etc/ssh/sshd_config* solo nombraremos unos pocos parámetros:

En primer lugar en *Port* podemos cambiar el puerto de escucha del servicio, recordemos que por defecto es el 22, por otro puerto libre que nos pueda interesar en nuestra organización.

En *ListenAddress* podemos indicar la IP en la que atiende el servicio si queremos que lo haga en unas direcciones concretas y no en todas las que tiene configurado el sistema. Por defecto, permite el acceso desde todas la interfaces de red, representado por la IP 0.0.0.0

En el apartado *# Logging* podemos establecer los parámetros para registrar las incidencias y funcionamiento del servicio. Con *SysLogFacility* indicamos el código con el que se va a guardar la información registrada, por defecto es AUTH y los registros se almacenarán por defecto en */var/log/auth.log.* Podemos cambiar este valor por DAEMON, USER, LOCAL0, …, LOCAL7 para que se guarden en el fichero correspondiente. Con LogLevel indicamos las importancia de los eventos a partir de la cual serán registrados, por defecto es INFO, lo que hace que registre muchos sucesos relacionados con el servicio, pero podemos cambiarlo por valores superiores como NOTICE o WARN si queremos recopilar menos información sobre el funcionamiento de sshd.

Con `#PermitRootLogin prohibit-password` establecemos si se puede acceder remotamente a nuestro sistema como el usuario *root,* por defecto está prohibido (valor *no*) por razones de seguridad, es mejor acceder como un usuario normal y usar *sudo.* Si queremos permitir el acceso el valor será *yes* Si vale `prohibit-password` entonces podrá entrar como *root* y sin password, lo que puede ser muy cómodo a la vez que peligroso, además esta opción depende de otras opciones tanto de sshd como del propio sistema, como veremos más adelante, si necesitamos que root entre sin contraseña lo haremos usando el acceso mediante certificados digitales.

La opción *MaxSessions* núm. permite limitar el número de sesiones simultaneas.

Con *AllowUsers* y *DenyUsers* permite crear listas de usuarios permitidos o prohibidos para acceder de forma remota. También se puede aplicar a grupos con *AllowGroups* y *DenyGroups*.

Con *PubKeyAuthentication yes* | *no* indica si se permite el acceso con clave pública, por defecto sí.

La opción *X11Forwarding yes* permite el acceso remoto a aplicaciones gráficas, lo veremos más adelante.

Con *PermitTunnel yes* | *no* | *point-to-point* permite la creación de túneles de comunicación entre extremos lo estudiaremos en el punto correspondiente en este documento.

Finalmente, tenemos la opción que afecta al uso de FTP seguro y que se establece en la opción

Subsystem sftp /usr/lib/openssh/sftp-server

La lista completa de opciones las puedes consultar con la orden: *$ man sshd_config*

Autenticación de usuarios

Como hemos visto anteriormente, podemos establecer una conexión con servidor remoto y también una sesión de FTP seguro (sftp) indicando el nombre de un usuario del sistema al que nos vamos a conectar y su contraseña. Esta sería la manera más sencilla con la que autenticarnos usando ssh, pero también podemos hacerlo usando claves de cifrado y también mediante una lista de ordenadores a los que se permite la conexión remota a nuestro servidor ssh.

Autenticación con clave de cifrado

Para usar este tipo de autenticación, en primer lugar hemos de generar las claves de cifrado, son una pareja: clave pública y clave privada. En general, la clave pública se distribuye sin ningún problema a quien tenga que cifrar una información para enviárnosla y la clave privada la mantenemos guardada en un lugar protegida y se usa para descifrar la información que se ha cifrado con nuestra clave pública. Está claro que con la clave pública no es posible de ninguna manera descifrar nada que se haya cifrado con ella misma.

El proceso de autenticación sigue los pasos siguientes:

- De forma segura y solo lo haremos una vez, copiamos la clave pública en el servidor remoto al que deseamos acceder.
- Cuando queremos conectarnos, indicamos en la orden ssh usuario IP o nombre_servidor
- El servidor genera un número aleatorio, lo cifra con nuestra clave publica y nos lo envía.
- En nuestro sistema se usa la clave privada para descifrar el número y se lo enviamos como respuesta al servidor.
- Si nuestra respuesta coincide con el número que había generado nos dará acceso, ya que solo podríamos haberlo descifrado con la clave privada correcta.

Este mecanismo de validación se le llama *challange* o *desafío*. Los pasos descritos los realizan de forma automática los procesos ssh y sshd.

Para generar el par de claves usamos la orden:

```
$ ssh-keygen -t rsa
```

Creará las claves pública y privada. La privada la podemos proteger con una contraseña de forma que cuando necesite ser usada nos la preguntarán, aunque hay casos en que no es conveniente asignar contraseña para facilitar procesos automatizados de conexión, siendo suficiente que el fichero que contiene la clave privada esté protegido con los permisos de acceso apropiados.

El resultado de la orden anterior son dos ficheros, guardados en nuestro directorio *home* en el subdirectorio .ssh: *id_rsa.pub* para la clave pública y *id_rsa* para la clave privada, pero se puede cambiar al crearlo.

```
lliurex@tic:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/lliurex/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/lliurex/.ssh/id_rsa.
Your public key has been saved in /home/lliurex/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:m3YdGawkXE6A53pRHSAAdBd0Oi4cYS0wuAoLOXXwOcg lliurex@tic
The key's randomart image is:
+---[RSA 2048]----+
|  .+=o==*o=o..   |
| .oo.=o+oB...    |
| oEo+ .+=.o o    |
|= .  o oo+ . o   |
|o+    o.S.. o    |
|o     ...o . .   |
|       .+ . .    |
|       . .       |
|                 |
+----[SHA256]-----+
lliurex@tic:~$
```

```
lliurex@tic:~$ cd .ssh
lliurex@tic:~/.ssh$ ls -l
total 12
-rw------- 1 lliurex lliurex 1679 may 20 17:38 id_rsa
-rw-r--r-- 1 lliurex lliurex  393 may 20 17:38 id_rsa.pub
-rw-r--r-- 1 lliurex lliurex  222 may 19 12:47 known_hosts
lliurex@tic:~/.ssh$
```

Como hemos comentado al describir el proceso *challange*, ahora debemos transferir la clave pública al servidor al que queremos acceder remotamente. Esta clave se copiará en el *home* del usuario del servidor remoto usado para acceder, en el subdirectorio *.ssh* dentro del fichero *authorized_keys*. La copia del valor de la clave pública al fichero en servidor la podemos hacer de diferentes formas, por ejemplo copiando el fichero al servidor y luego con un editor de texto plano añadir la clave al final del fichero *authorized_keys*, pero es más sencillo usar la orden *ssh-copy-id* que realiza todo el proceso en un solo paso:

```
$ ssh-copy-id -i .ssh/id_rsa.pub  usuario_remoto@IP o nombre_servidor
```

```
lliurex@tic:~$ ssh-copy-id -i .ssh/id_rsa.pub lliurex@10.20.30.136
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: ".ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
lliurex@10.20.30.136's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'lliurex@10.20.30.136'"
and check to make sure that only the key(s) you wanted were added.

lliurex@tic:~$
```

para realizar la copia nos pide el password del usuario remoto para poder acceder al sistema.

Ahora ya no nos pide la contraseña del usuario remoto (tampoco la de la clave privada ya que no la asignamos en este ejemplo).

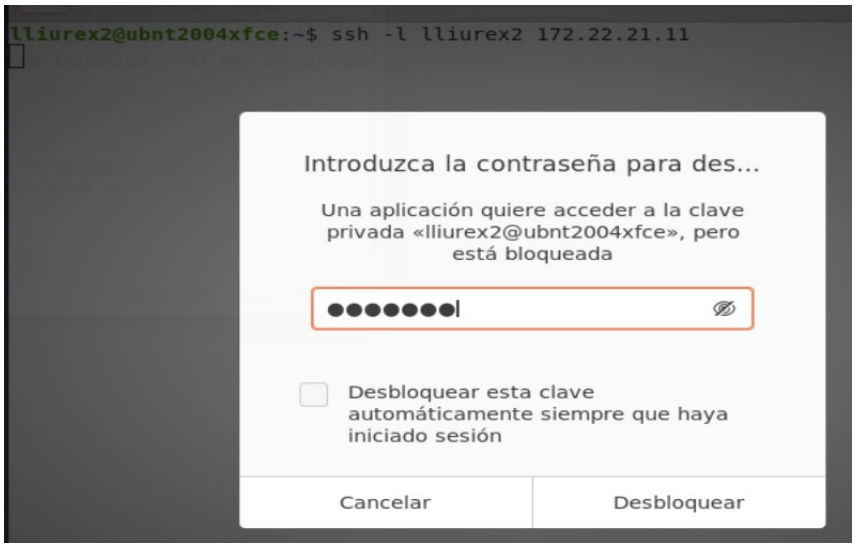
```
lliurex@tic:~$ ssh -l lliurex 10.20.30.136

 __         __         __     __  __     ______     ______     __  __
/\ \       /\ \       /\ \   /\ \/\ \   /\  == \   /\  ___\   /\_\_\_\
\ \ \____  \ \ \____  \ \ \  \ \ \_\ \  \ \  __<   \ \  __\   \/_/\_\/_
 \ \_____\  \ \_____\  \ \_\  \ \_____\  \ \_\ \_\  \ \_____\   /\_\/\_\
  \/_____/   \/_____/   \/_/   \/_____/   \/_/ /_/   \/_____/   \/_/\/_/

Welcome to lliurex 19.210504 (GNU/Linux 5.4.0-48-generic x86_64)

 * Documentation:  http://lliurex.net/

        -CLIENTE: Desktop
        -VERSION: 19.210504
        -ARQUITECTURA: 64 Bits
        -CPU: Intel Core i7-7700 CPU @ 4.2GHz
        -RAM: 757MiB / 31910MiB

Hora actual: 18:03:33 up 1:17, 2 users, load average: 0,10, 0,06, 0,02
lliurex@llrx19:~$
```

En caso de usar contraseña en la clave privada se puede usar el proceso *ssh-agent* al que le podemos indicar esta contraseña y se encargará de usarla cuando sea necesario. Para esto ejecutamos la orden:

```
$ ssh-add [fichero]
```

Si no indicamos ningún fichero buscará la clave privada en *.ssh/id_rsa*

Si hemos establecido la *passphrase* al crear la clave cuando accedemos al sistema remoto nos la pide antes de dar acceso:



para evitar este paso como se ha dicho usamos la orden:

```
lliurex2@ubnt2004xfce:~$ ssh-add
Enter passphrase for /home/lliurex2/.ssh/id_rsa:
Identity added: /home/lliurex2/.ssh/id_rsa (lliurex2@ubnt2004xfce)
lliurex2@ubnt2004xfce:~$
```

a partir de ahora entra como si no tuviese *passphrase*, cuando la necesita es *ssh-agent* quien la proporciona.

Reenvío X11 (X-Windows)

Esta facilidad permite ejecutar remotamente aplicaciones gráficas. Con la conexión remota tenemos una terminal en la que ejecutamos aplicaciones de consola en el sistema remoto, pero habilitando esto podemos hacerlo también con las que necesitan de entorno gráfico.

La configuración en el sistema servidor, */etc/ssh/sshd_config* habilitar:

```
X11Forwarding yes
```

En el sistema cliente, */etc/ssh/ssh_config* habilitar: `ForwardX11 yes`

Con las configuraciones anteriores para usar el reenvío X11 ejecutamos ssh con la opción -X:

```
$ ssh -X usuario@servidorSSH
```

Y en la sesión remota ejecutamos la aplicación gráfica que no interese, por ejemplo Firefox:

```
$ firefox &
```

Para que funcione necesitamos tener instaladas X-Windows y un gestor de ventanas en el sistema cliente para visualizar la ventana en la que se ejecuta la aplicación del servidor. Si el sistema cliente no lo tiene debemos ejecutar para instalarlo : `# apt-get install xorg fvwm`

Cada vez que necesitemos las X-Windows en el cliente ejecutamos: `$ startx`

Transferencia segura de ficheros

Esta es otra de las funcionalidades ofrecidas por OpenSSH, con ella podemos intercambiar ficheros entre sistemas remotos de forma segura. Hay dos órdenes, la más sencilla *scp* copia ficheros entre el sistema local y el remoto como lo haría al orden *cp* de Linux y la otra es *sftp* que trabaja como el servicio *ftp* pero en un entorno de transferencia seguro.

Sintaxis de *scp*:

```
$ scp path/fichero usuario@servidorSSH:path/fichero_destino
```

```
$ scp usuario@servidorSSH:path/fichero_destino path/fichero
```

La seguridad aplicada es la misma que para *ssh,* de forma que pedirá la contraseña o passphrase si es necesaria.

Para *sftp* la orden sería:

```
$ sftp [usuario@]servidorSSH
```

El funcionamiento será similar al de ftp, creando una sesión interactiva para el intercambio de ficheros en los dos sentidos de la conexión. Lo veremos en el tema sobre el servicio FTP.

Reenvío de puertos TCP / SSH Tunneling

Esta funcionalidad permite la conexión (reenvío) de puertos asociados a aplicaciones TCP del sistema cliente. Puede usarse para añadir seguridad mediante cifrado a aplicaciones antiguas, como por ejemplo POP3, también puede servir para pasar a través de cortafuegos o para que administradores de sistemas abran puertas traseras (*backdoor*) hacia la red interna de la organización para acceder remotamente, por ejemplo desde su domicilio. Es evidente que el reenvío de puertos debe usarse con mucha precaución ya que también puede usarse para realizar ataques a la red desde el exterior.

El reenvío de puertos puede realizarse en los dos sentidos de la conexión segura SSH. Veamos en primer lugar el reenvío de puertos locales. Se trata de reenviar un puerto del cliente hacia el exterior, ya sea el destino el servidor SSH u otro sistema accesible desde este.

La mecánica consiste en que el cliente escucha peticiones en el puerto indicado y cuando la recibe la envía de forma segura al servidor SSH que la reenvía al puerto especificado del sistema remoto. Si el servidor SSH no es el sistema destino se le considera servidor de salto (*jump server*).

La orden sería:

```
$ ssh -L [Dir_local]:portLocal:sist_Destino:portRemoto  servidor_SSH
```

La opción **-L** indica el reenvío local, del sistema cliente.

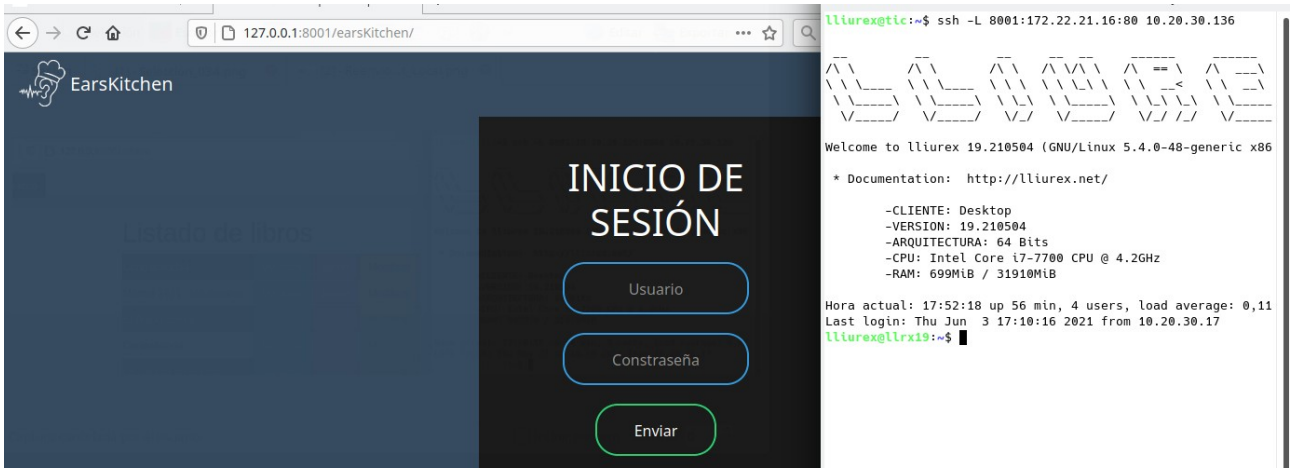
`Dir_local` , opcional, sería el nombre, IP o 127.0.0.1, para que el reenvío solo pueda usarlo nuestro sistema cliente, si no lo indicamos cualquier sistema podría conectarse a nuestro sistema cliente en este puerto y tendría acceso al puerto reenviado a través del túnel que se establece. Por tanto, si solo vamos a usarlo en el sistema que estamos configurando conviene poner este valor y evitar accesos indeseados.

`portLocal` será el número del puerto al que hacemos referencia en el sistema cliente para acceder a través del túnel al sistema remoto.

`sist_Destino` pondremos el nombre o IP del sistema remoto al que accedemos con el reenvío de puertos. Donde estará la aplicación asociada al puerto remoto establecido.

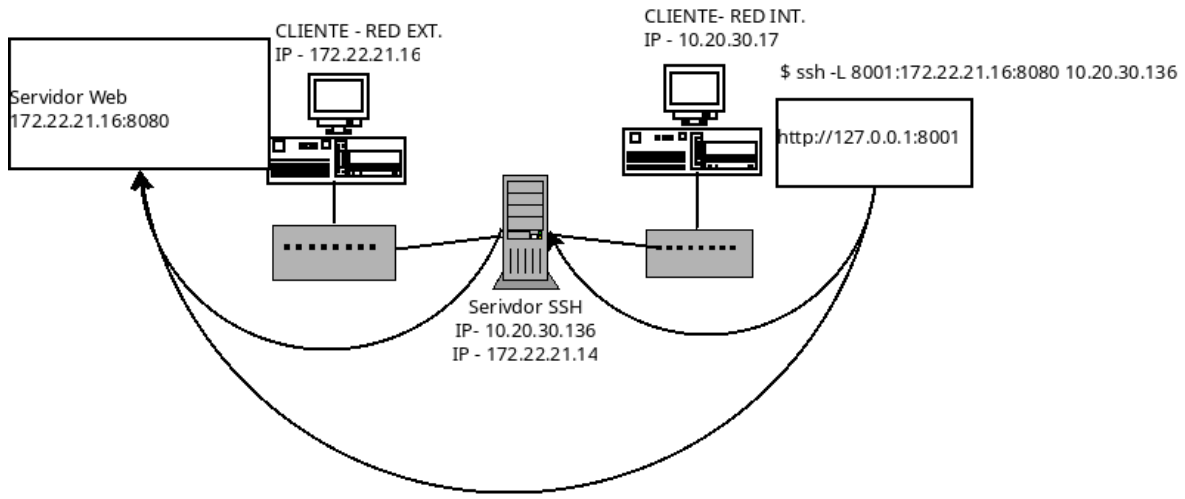
portRemoto número del puerto al que accedemos en el sistema especificado en el valor anterior.

Finalmente, `servidor_SSH` será el nombre o IP del servidor SSH usado para crear el túnel entre el sistema cliente que lo crea y el sistema remoto en el que está el puerto al que se desea acceder.



En la captura de pantalla anterior, no se especifica sistema cliente, el puerto local es 8001, el sistema remoto que tiene la aplicación a la que deseamos acceder es 172.22.21.16 que escucha en el puerto 8080 y el servidor SSH que crea el túnel es 10.20.30.136. Como se puede ver al poner en el navegador la IP local y el puerto reenviado (8001) nos carga la aplicación web que está ejecutándose en el sistema 172.22.21.16 en el puerto 8080. Es decir que accedemos a la aplicación remota como si fuese local.

El esquema de conexión sería:



Para el reenvío de puertos remotos las orden sería:

```
$ ssh -R portRemoto:sist_Local:portLocal  servidor_SSH
```

La opción **-R** indica el reenvío remoto.

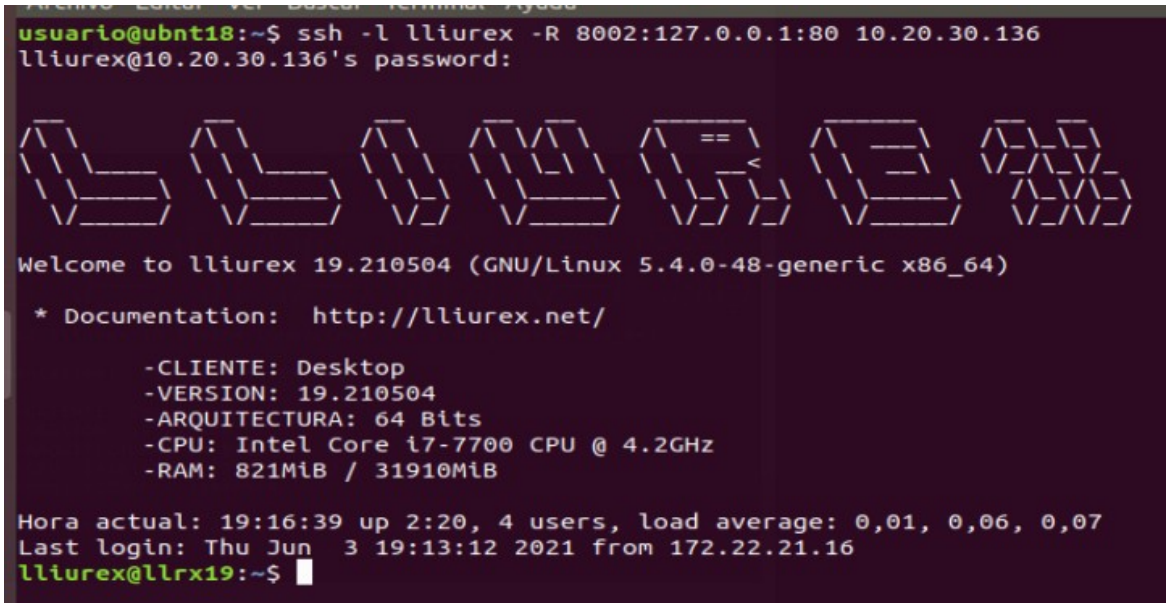
`portRemoto` será el número del puerto del sistema remoto al que se puede conectar cualquiera y que será reenviado hacia el puerto de nuestro sistema cliente (local). Si en este valor ponemos 0 (cero) el sistema elige un puerto dinámicamente e informa al cliente del puerto a usar y si en la orden ponemos al opción -O forward mostrará el puerto por pantalla.

`sist_Local` pondremos el nombre o IP del sistema cliente (local), también puede ser el localhost o 127.0.01.

`portLocal` número del puerto local al que se están reenviando las solicitudes de acceso desde el servidor SSH y que le llegan desde el sistema remoto.

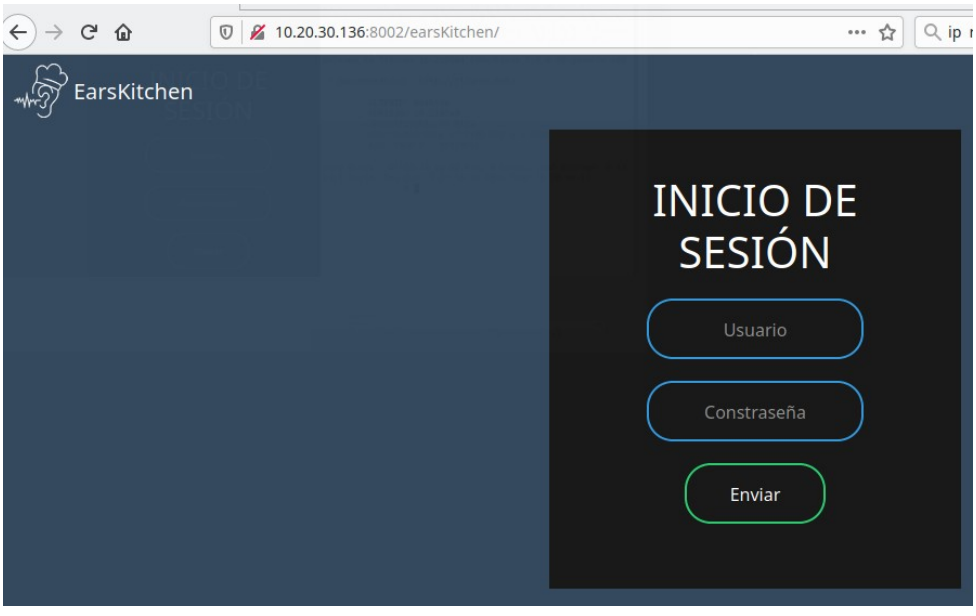
Finalmente, `servidor_SSH` será el nombre o IP del servidor SSH usado para crear el túnel entre el sistema cliente que lo crea y los sistemas que acceden a este puerto.

En este caso cabe recordar que estamos dando acceso desde sistemas remotos a un sistema en nuestra red interna y por tanto esto puede facilitar tanto el acceso de un trabajador o un cliente de nuestra organización que se nos conecte desde su casa u oficina o bien una puerta para atacantes desde internet.

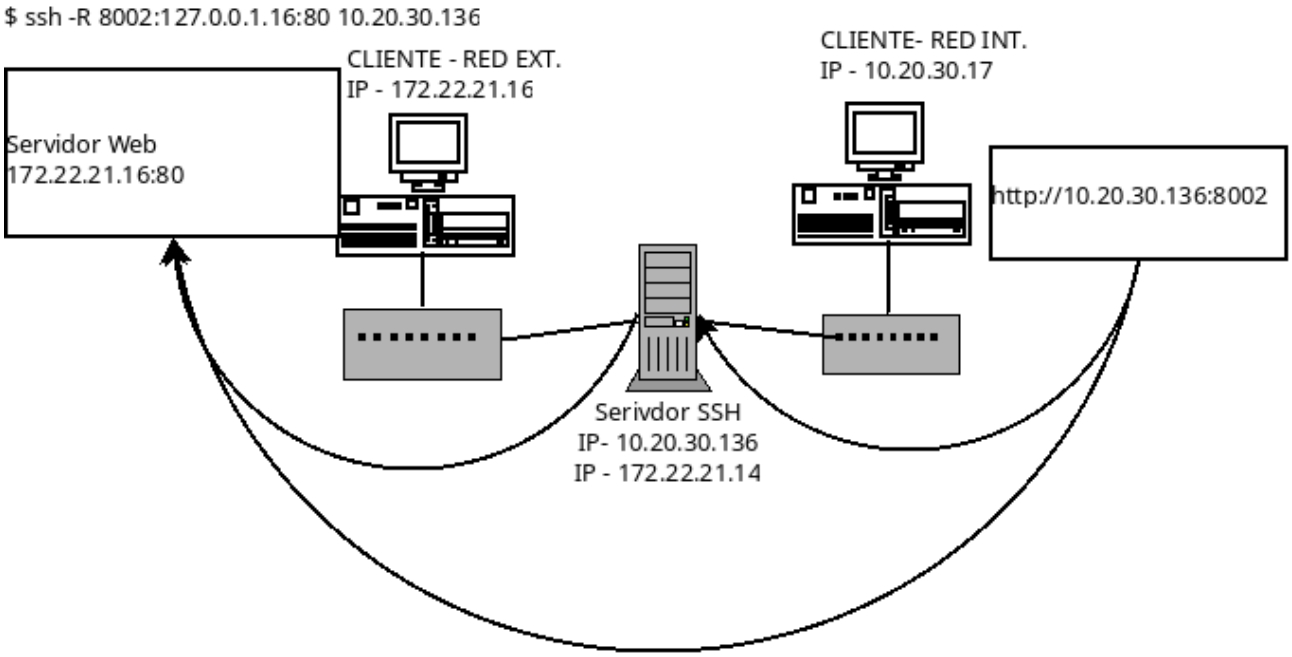


En la captura de pantalla tenemos como sistema cliente (local) 172.22.21.16 el puerto remoto será 8002, como sistema local podemos 127.0.0.1, pero también podemos poner 172.22.21.16. el puerto local en el que escucha la aplicación a redirigir es 80 y el servidor SSH 10.20.30.136.

Un sistema remoto se conecta, por ejemplo con el navegador al 10.20.30.136:8002 y accede a la aplicación web ejecutada en el sistema cliente (local).



El esquema de conexión sería:



Para el reenvió de puertos se debe configurar en */etc/ssh/sshd_config*

AllowTcpForwarding yes o all → permite el reenvío de puertos, opción por defecto.

AllowTcpForwarding no →impide el reenvío.

AllowTcpForwarding local →solo el reenvío de puertos locales.

AllowTcpForwarding remote →solo el reenvío de puertos remotos.

Para la configuración del funcionamiento del reenvío a puertos remotos tenemos también:

GatewayPorts no → impide la conexión a puertos reenviados por parte de sistemas diferentes al propio servidor, opción por defecto.

GatewayPorts yes → permite a cualquiera la conexión, incluso si está en internet.

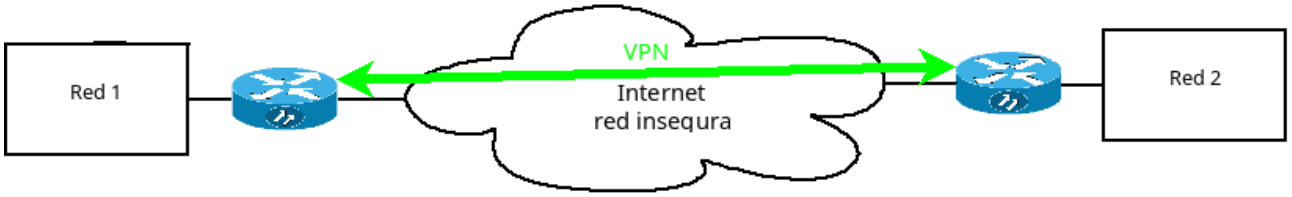
GatewayPorts clientspecified → en este caso el sistema cliente en la orden de reenvío especifica la IP a la que se permite la conexión al puerto:

```
$ ssh -R IP_permitida:portRemoto:sist_Local:portLocal  servidor_SSH
```

solo esta IP podrá conectarse al puerto remoto a través del servidor SSH.

Red Privada Virtual (VPN)

Una VPN, *Virtual Private Network* en inglés, se puede definir como una tecnología que permite conectar dos redes remotas usando una red intermedia insegura, como puede ser internet, de forma que la información transmitida entre las dos redes se realiza de forma segura y como si las dos redes estuvieran físicamente conectadas desde el punto de vista de los usuarios como una red punto a punto.



Para crear las VPN hay muchas soluciones, tanto basadas en hardware como en software. A su vez las implementaciones se pueden realizar en diferentes niveles del modelo de red OSI, hay implementaciones sobre todo en los niveles 2 y 3, pero también las hay en los niveles 4 y 7. Por tanto, el concepto de VPN es fundamental actualmente y la oferta para ajustarse a la diferentes necesidades muy amplia.

En nuestro caso no nos vamos a extender demasiado, centrándonos en las posibilidades que ofrece OpenSSH. Básicamente, consiste en crear una VPN de nivel 3 (nivel de Red OSI) en el que se crea un modelo de encapsulado de paquetes de datos creando un túnel entre los extremos de la VPN (***tunneling***). Lo que hace es tomar los paquetes de datos que se generan en el nivel de red que se envía entre las redes, los cifra para que no se acceda a su contenido y los mete en nuevos paquetes de nivel de red que se enviará a través de la interfaces de red que implementan el túnel de la VPN. Es decir que se transmiten los paquetes de nivel de red cifrados en otros paquetes de nivel de red que se serán los que viajarán por la red insegura. Cuando llegan a destino la interfaz de la VPN extrae estos paquetes los descifra y los pasa a la interfaz de red destino de los paquetes de nivel de red originales.

Como sabemos el nivel de Transporte, nivel 4 OSI, puede ser el protocolo TCP y el UDP. Para OpenSSH usa TCP, es decir que para crear la VPN encapsula paquetes TCP dentro de paquetes TCP, esto puede crear serios problemas de funcionamiento de la VPN, la razón básicamente deriva de usar los protocolos de entrega confiable, así que si hay problemas en la conexión, el protocolo empezará a pedir retransmisiones de paquetes, pero en los dos niveles tanto en el TCP que interno y

en el externo de forma que en algunos casos las retransmisiones y confirmaciones de paquetes puede llegar a colapsarse y dejar de funcionar. Pero a pesar de este problema, OpenSSH permite crear una VPN de forma muy sencilla que para usarse ocasionalmente puede resultar muy útil. Evidentemente, para implantar VPN permanentes o de mucho tráfico se deberá buscar alternativas más fiables y / o eficientes.

Una solución alternativa muy empleada y sin el problema de usar TCP sobre TCP es OpenVPN

https://openvpn.net

esta se basa en usar TCP sobre UDP.

Veamos la configuración de una VPN con OpenSSH:

En el servidor SSH:

Habilitar root, actualmente este usuario no está activado y para acceder a sus derechos usamos la orden *sudo*, pero en este caso usaremos el propio usuario *root* :

```
#passwd root (dar una contraseña)
# mkdir /root/.ssh
```

En */etc/ssh/sshd_config*:

```
PermitTunnel yes
PermitRootLogin yes
```

En el cliente generamos una clave y la pasamos al servidor:

```
# ssh-keygen -t rsa (No poner nada en la passphrase)
# cd .ssh
# ssh-copy-id -i id_rsa.pub usuario@servidorSSH
```

Órdenes para establer la VPN, ejecutamos primero las del cliente:

Para crear el túnel para la VPN usaremos una interfaz de red denominada ***tun.*** Esta necesita que el kernel del sistema tenga cargado el módulo correspondiente y lo comprobamos con la orden:

```
# modprobe tun (Puede no ser necesario en sistemas actuales)
# ssh -f [-C] -w 0:0 IPServidorSSHD -N
```

(-C para usar compresión)

```
#ifconfig tun0 10.10.10.2 pointopoint 10.10.10.1 netmask 255.255.255.252
```

configuramos la interfaz de red *tun0* con la primera IP para el cliente, es una red punto a punto, la segunda IP será para el servidor SSH y la máscara de subred en la que solo se permite dos sistemas, los definidos en la orden.

Las órdenes en el servidor serán:

```
# modprobe tun (Puede no ser necesario en sistemas actuales)
#ifconfig tun0 10.10.10.1 pointopoint 10.10.10.2 netmask 255.255.255.252
```

Mediante las órdenes anteriores hemos definido una red para dos hots a los que se accede mediante las interfaces *tun0* de forma que el tráfico entre los dos host viajará como si se tratara de una red física entre ellos, mientras que en realidad, el tráfico que llega a las *tun0* se cifra y empaqueta dentro de otros paquetes TCP convencionales que envían por la red convencional. Así si se capturan los paquetes no se puede ver la información de los paquetes internos.

Veamos un ejemplo con las capturas del pantalla prinero del cliente:

```
root@tic:~# modprobe tun
root@tic:~# ssh -f -C -w 0:0 10.20.30.136 -N
root@tic:~# ifconfig tun0 10.10.10.2 pointopoint 10.10.10.1 netmask 255.255.255.252
root@tic:~#
```

```
root@tic:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.20.30.17  netmask 255.255.255.0  broadcast 10.20.30.255
        inet6 fe80::8781:f215:fcdf:7aff  prefixlen 64  scopeid 0x20<link>
        ether b0:6e:bf:2c:66:9d  txqueuelen 1000  (Ethernet)
        RX packets 132553  bytes 187878065 (187.8 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 102767  bytes 8722053 (8.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 16  memory 0xf7000000-f7020000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Bucle local)
        RX packets 4664  bytes 408676 (408.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4664  bytes 408676 (408.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST>  mtu 1500
        inet 10.10.10.2  netmask 255.255.255.252  destination 10.10.10.1
        inet6 fe80::9501:9bca:d22b:658c  prefixlen 64  scopeid 0x20<link>
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 500  (UNSPEC)
        RX packets 5  bytes 240 (240.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 7  bytes 336 (336.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@tic:~#
```

En el servidor:

```
root@llrx19:/etc/ssh# modprobe -v tun
root@llrx19:/etc/ssh# cd
root@llrx19:~# ifconfig tun0 10.10.10.1 pointopoint 10.10.10.2 netmask 255.255.255.252
```

```
root@llrx19:~# ifconfig tun0
tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST>  mtu 1500
        inet 10.10.10.1  netmask 255.255.255.252  destination 10.10.10.2
        inet6 fe80::bd2f:8c4a:f07:7170  prefixlen 64  scopeid 0x20<link>
        unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 500  (UNSPEC)
        RX packets 165  bytes 16302 (16.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 164  bytes 191185 (191.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@llrx19:~#
```

Ahora si usamos las IP del tunel la información viaja protegida entre el servidor y el cliente: