OpenSSH

Miguel Ortuño
Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad Rey Juan Carlos

Septiembre de 2022

OpenSSH

- PGP: Pretty Good Privacy. Software criptográfico creado por Phil Zimmermann, año 1991. Base de la norma Open PGP.
- GPG: GNU Privacy Guard. Herramienta para cifrado y firmas digitales, que reemplaza a PGP
- Se puede emplear algoritmos como RSA o ed25519 (algo más moderno), ambos se consideran seguros. El algoritmo DSA ya no es recomendable
- Las distribuciones orientadas a sistemas empotrados no suelen usar OpenSSH sino Dropbear, un cliente y un servidor de ssh, compatible con OpenSSH, más ligero

Criptografía de clave pública

Aparece con el algoritmo Diffie-Hellman, año 1976

- Clave de cifrado o pública E y de descifrado o privada D distintas (asimétricas)
- $D(E(P)) = P$
- Muy dificil romper el sistema (p.e. obtener D) teniendo E.
- Permite intercambiar claves por canal no seguro
- La clave privada sirve para descifrar. Debe mantenerse en secreto
- La clave pública sirve para cifrar. Puede conocerla todo el mundo (lo importante es que se conozca la clave correcta)

- Conociendo la clave pública de alguien, podemos cifrar un mensaje que solo él, con su clave privada, podrá descifrar
- Los algoritmos de clave pública son mucho más lentos que los de clave secreta (100 a 1000 veces). Por eso se suelen usar sólo para el intercambio de claves simétricas de sesión
- También sirve para autenticar (como en OpenSSH)
  Queremos desde una sesión en una máquina local, abrir otra sesión en una máquina remota sin volver a teclear contraseña
  - Una máquina remota, no fiable, contiene clave pública
  - Máquina local, fiable, contiene la clave privada
  - La máquina remota envía un reto cifrado con la clave pública, si la máquina local lo descifra, el usuario queda autenticado y puede abrir sesión en la máquina remota sin teclear contraseña

Uso de OpenSSH

```
ssh usuario@maquina
```

Abre una sesión remota mediante una conexión segura en la máquina indicada, con el usuario indicado.

- La primera vez que abrimos una sesión en una máquina, ssh nos indica la huella digital de la máquina remota

```
The authenticity of host 'gamma23 (212.128.4.133)' can't be established.
RSA key fingerprint is de:fa:e1:02:dc:12:8d:ab:a8:79:8e:8f:c9:7d:99:eb.
Are you sure you want to continue connecting (yes/no)?
```

- Si necesitamos la certeza absoluta de que esta máquina es quien dice ser, deberíamos comprobar esta huella digital por un medio seguro, alternativo

- El cliente ssh almacena las huellas digitales de las máquinas en las que ha abierto sesión en el fichero `~/.ssh/known_hosts`
- El servidor almacena su propia huella digital en los ficheros

  `/etc/ssh/ssh_host_rsa_key`
  `/etc/ssh/ssh_host_ed25519_key`

  Si la huella que tiene el host en la actualidad no coincide con la huella que tenía el host en la primera conexión, ssh mostrará un error
  Esto puede suceder porque
  - Alguien esté suplantando la identidad del host
  - El host ha sido reinstalado y el administrado no ha conservado estos ficheros

```
ssh -C -X usuario@maquina
```

- La opción `-X` (mayúscula) redirige la salida del cliente X Window de la máquina remota al servidor X Window de la máquina local
  Esto permite lanzar aplicaciones gráficas en la máquina remota, usarán la pantalla local
  Es necesario
  - `X11Forwarding yes`
    en `/etc/ssh/sshd_config` en la máquina remota
  - Que la máquina local admita conexiones entrantes
- La opción `-C` (mayúscula) comprime el tráfico. En conexiones rápidas es conveniente omitir esta opción

Además de para abrir una sesión en una máquina remota, ssh permite la ejecución de una única orden en la máquina remota

- `ssh jperez@alpha ls`

  Ejecuta `ls` en la máquina remota. Muestra en la máquina local el stdout.

- `ssh jperez@alpha 'echo hola > /tmp/prueba'`

  Ejecuta en alpha

  `echo hola > /tmp/prueba`

- `ssh jperez@alpha "echo $HOSTNAME > /tmp/prueba"`

  Al poner comilla doble, la variable se expande en la máquina local. La orden completa, redirección incluida, se ejecuta en la máquina remota
- `ssh jperez@alpha echo $HOSTNAME > /tmp/prueba`

  Al no poner comilla, la variable se expande en la máquina local. El resultado se redirige al fichero `/tmp/prueba` de la máquina local
- `ssh jperez@alpha 'echo $HOSTNAME > /tmp/prueba'`

  Al poner comilla simple y recta, la variable se expande en la máquina remota

Generación de claves

Para evitar teclear contraseña en cada ssh, podemos autentificarnos con claves asimétricas
Una vez configurado para ssh, también queda configurado para los servicios que corren sobre este (scp, sshfs)

- Se generan con `ssh-keygen`
- Se puede añadir una *pass phrase*. Es una contraseña adicional, tradicional. Pero no viaja por la red. Equivalente a la llave del armario de las llaves

Un usuario genera sus claves ejecutando en su *home* (de la máquina local) la orden *ssh-keygen*

- rsa:

```
orden para generar las claves:
    ssh-keygen -t rsa
fichero donde quedará (por omisión) la clave privada:
    ~/.ssh/id_rsa
fichero donde quedará (por omisión) la  clave pública:
    ~/.ssh/id_rsa.pub
```

- ed25519:

```
orden generar las claves:
    ssh-keygen -t ed25519
fichero donde quedará (por omisión) la clave privada:
   ~/.ssh/id_ed25519
fichero donde quedará (por omisión) la clave publica:
   ~/.ssh/id_ed25519.pub
```

Para poder entrar en máquina remota sin emplear contraseña, llevamos la clave pública a la máquina remota, y la escribimos en el fichero

- `~/.ssh/authorized_keys` (Redhat, Debian)
- `/etc/dropbear/authorized_keys` (OpenWrt)

Este fichero (de la máquina remota) en principio no existe

- La primera vez que añadamos una clave, podemos renombrar el fichero con la clave pública para que pase a llamarse `authorized_keys`
- Si posteriormente añadimos otras claves públicas, las pegamos inmediatamente después de las que ya existan, usando un editor de texto o una redirección de la shell

Permisos

Es necesario que el directorio `~/.ssh` (local y remoto):

- Tenga el dueño y el grupo del usuario
- Tenga permisos 700
- Contenga todos sus ficheros con permisos 600
- Todos sus ficheros pertenezcan al usuario y tengan como grupo el del usuario

Es necesario que en mi *home* solo yo pueda escribir

- En OpenWrt, también es necesario que `/etc/dropbear/authorized_keys` tenga permisos 600
- En Docker, en la máquina donde corre el servidor es necesario configurar el demonio:

```
echo "IdentityFile ~/.ssh/id_ed25519" >> /etc/ssh/ssh_config
```

O bien

```
echo "IdentityFile ~/.ssh/id_rsa" >> /etc/ssh/ssh_config
```

ssh-copy-id

Se puede usar la orden `ssh-copy-id`, que se encarga de

- Copiar la clave pública a la máquina remota
- Crear `authorized_keys` si no existe
- Cambiar todos los permisos

```
ssh-copy-id [-i [identity_file]] [user@]machine
```

Ejemplo:

```
ssh-copy-id -i id_ed25519.pub  jperez@iota34
```

Ejemplo: Configuración típica

Una clave privada distinta para cada uno de mis ordenadores

- Soy jperez, a veces trabajo localmente en pc-casa, a veces trabajo localmente en pc-oficina
- Desde ambos sitios quiero entrar en pc-remoto
- Desde casa entro en la oficina
- Desde la oficina, entro en casa
- Creo una clave privada jperez@pc-casa
- Creo una clave privada jperez@pc-oficina
- En el `authorized_keys` de pc-remoto:
  concateno claves públicas de jperez@pc-casa y jperez@pc-oficina
- En el `authorized_keys` de pc-casa
  Escribo la clave pública de jperez@pc-oficina
- En el `authorized_keys` de pc-oficina
  Escribo la clave pública de jperez@pc-casa

Ejemplo: Configuración alternativa

La misma clave para todos mis ordenadores
Aunque a las claves se les pone por omisión una etiqueta usuario@máquina (que aparece como comentario al final de la clave), solo es un comentario orientativo, una misma clave privada puede usarse desde distintas máquinas

- Creo una clave privada jperez, y la copio en pc-casa y en pc-oficina
- En el `authorized_keys` de pc-casa, de pc-oficina y de pc-remoto escribo la clave pública de jperez

Este enfoque es menos flexible y menos seguro

Es posible usar varias claves privadas (cada una en su fichero), basta indicar a ssh cuál (o cuáles) debe emplear

```
ssh jperez@alpha
    # intenta autenticarse con  ~/.ssh/id_rsa o ~/.ssh/id_ed25519
    # (clave por omisión)

ssh -i ~/.ssh/id_alumno alumno@pc01   #
    # lo intenta con id_alumno y con la clave por omisión

ssh -i ~/.ssh/id_alumno -i ~/.ssh/id_profe alumno@pc01
    # lo intenta con id_alumno, con id_profe  y con la clave por omisión
```

scp también admite la opción -i

```
scp -i ~/.ssh/id_alumno alumno@pc01:/tmp/test .
```

(sshfs no admite la opción -i)

ssh-agent

La manera habitual de autentificarse es mediante el demonio *ssh-agent*

- *ssh-agent* contestará por nosotros, gestionando retos y repuestas cifrados
- *ssh-agent* tiene que ser el padre de nuestra shell, o nuestra sesión x
  - Las distribuciones Linux con X Window suelen tenerlo instalado
  - Si no está funcionando (como en ubuntu server)
    `exec ssh-agent /bin/bash`
    Esto hace que nuestra shell actual sea reemplazada por *ssh-agent*, quien a su vez creará una shell hija suya

- `ssh-add` añade una identidad a ssh-agent
- `ssh-add -l` indica las identidades manejadas por el ssh-agent

En ocasiones, por ejemplo si no empleamos *pass phrase*, el *ssh-agent* no es necesario

Depuración

- En el cliente:
  `ssh -v` o `ssh -vv` o `-vvv`
- En el servidor:
  `/var/log/auth.log`

Los errores más frecuentes suelen ser ficheros de configuración con nombre incorrecto o permisos incorrectos

Configuración de ssh en el cliente

El usuario puede configurar el comportamiento de su cliente ssh en el fichero `~/.ssh/config`
Algunas de las opciones más interesante son

- Especificar cuál será el usuario por omisión para una maquina en particular, sin necesidad de especificarlo en cada ocasión
- Indicar el nombre de una máquina sin usar ni la dirección IP, ni el nombre completo (FQDN) ni modificar el fichero `/etc/hosts` (Que exije privilegios de root)
- Enviar periódicamente un mensaje al servidor para que mantenga abierta la conexión

Fichero ~/.ssh/config de ejemplo

```
host alpha
user juanperez
hostname 192.168.19.27
# Permite hacer directamente 'ssh alpha', sin indicar mi usuario
# (Si el usuario coincide en máquina local y remota, tampoco hace
# falta especificarlo)

host alpha
user juanperez
hostname alpha.midominio.com
# Similar al caso anterior, pero con FQDN

host *
  ServerAliveInterval 60
# Mantiene la conexión abierta
```

Mas información en

`man 5 ssh_config`

Configuración adicional

- `/etc/ssh/ssh_config`
- `/etc/ssh/sshd_config`

sshfs

Supongamos que, usando la red, quiero trabajar con unos datos que están en una máquina remota, su sitio no es la máquina en la que yo estoy sentado. Tal vez porque uso un ordenador móvil, pero los datos no son móviles
Disponemos de muchísimas soluciones, cada una con sus ventajas e inconvenientes
Podemos trabajar:

- Por ssh
  Pero estamos limitados a la shell. No podemos usar ninguna aplicación gráfica, resulta muy limitado en Windows, ...
- Con una sesión gráfica remota: vnc, escritorio remoto de Windows, X window en remoto, etc
  Pero necesitamos una conexión relativamente buena y cargamos mucho la máquina remota, toda la aplicación está en la máquina remota

- Podemos sincronizar al estilo Dropbox.
  Pero necesitamos una cuenta, vinculada a 1 persona, con limitaciones de tamaño, dependencia del proveedor, etc
  Además se pueden provocar discrepancias, y los ficheros solo se guardan en la máquina remota cuando abro una sesión en la máquina remota y sincronizo
- Podemos montar un sistema de ficheros por NFS (como en nuestros laboratorios Linux), por SAMBA o similar
  Pero hace falta mucha administración en la máquina remota, y normalmente, por motivos de seguridad, el cliente solo podrá estar en sitios muy concretos
- Podemos usar una VPN, cuya administración no es trivial
- Podemos usar un directorio compartido de VirtualBox, pero solo en el caso (muy) particular de un *guest* VirtualBox y un *host* que lo soporte
- Otra de las alternativas es sshfs

sshfs: Secure SHell FileSystem
Sistema de ficheros de red basado en FUSE (*Filesystem in userspace*)
Permite usar un sistema de fichero remoto como si fuera local

- Hay disponibles implementaciones libres y gratuitas, para Linux, macOS y Windows (SSFS-Win)
- Menos eficiente pero más seguro que NFS
- No hace falta ninguna administración en la máquina remota (servidor) , basta con que tengamos una cuenta de ssh ordinaria
- En el cliente basta instalar el paquete `sshfs` y ejecutar una única orden

Inconvenientes de sshfs

- Dependemos continuamente de la red (con sistemas tipo Dropbox solo hace falta red cuando sincronizamos)
- En el ordenador local necesitamos todas las aplicaciones (mientras que con sistemas tipo vnc, el cliente puede ser mucho más ligero)
- Más pesado y menos eficiente que NFS o samba

Punto de montaje

En Unix/Linux, el punto de montaje es un directorio del sistema de ficheros *normal*, local, donde queremos que sea visible un nuevo sistema de ficheros
En el caso de sshfs, el nuevo sistema de ficheros será el de la máquina remota, a la que accedemos por ssh

- El punto de montaje es un directorio ordinario, que tiene que existir antes de montar el sistema remoto
- Suele estar vacío, pero puede contener ficheros
  En el caso de sshfs, si no está vacío hay que añadir la opción `-o nonempty`
  - Al montar un sistema de ficheros en un punto de montaje, el contenido del punto de montaje queda inaccesible
  - Al desmontar, el contenido vuelve a ser visible

Montar un directorio con sshfs

- Montar el *home* remoto:

```
sshfs  usuario@maquina: /punto/de/montaje
```

- Montar un directorio remoto cualquiera

```
sshfs  usuario@maquina:/un/directorio  /punto/de/montaje
```

(Siempre path absoluto, no soporta ~)

- Desmontar:

```
fusermount -u /punto/de/montaje
```

- Para poder usar sshfs (en el cliente) sin tener privilegios de root, es necesario activar la opción `user_allow_other` en el fichero `/etc/fuse.conf` (y reiniciar el demonio o la máquina)
- En conexiones lentas puede ser conveniente añadir la opción -C para que comprima el tráfico

```
sshfs -C usuario@maquina:/path  /punto/de/montaje
```

Túneles con SSH

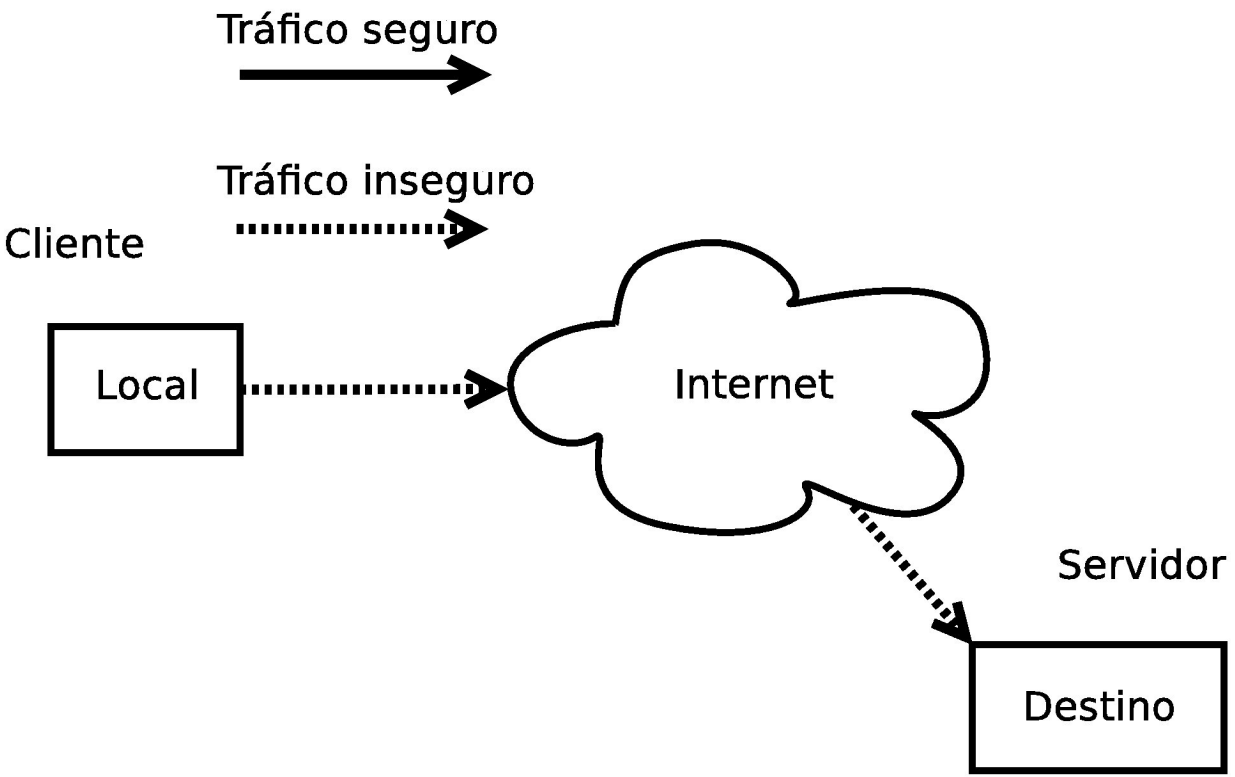
SSH permite hacer túneles, aka *port forwarding*
Concepto similar al de VPN, pero no es una verdadera VPN

- Se redirige un único puerto
- Solamente TCP, no UDP

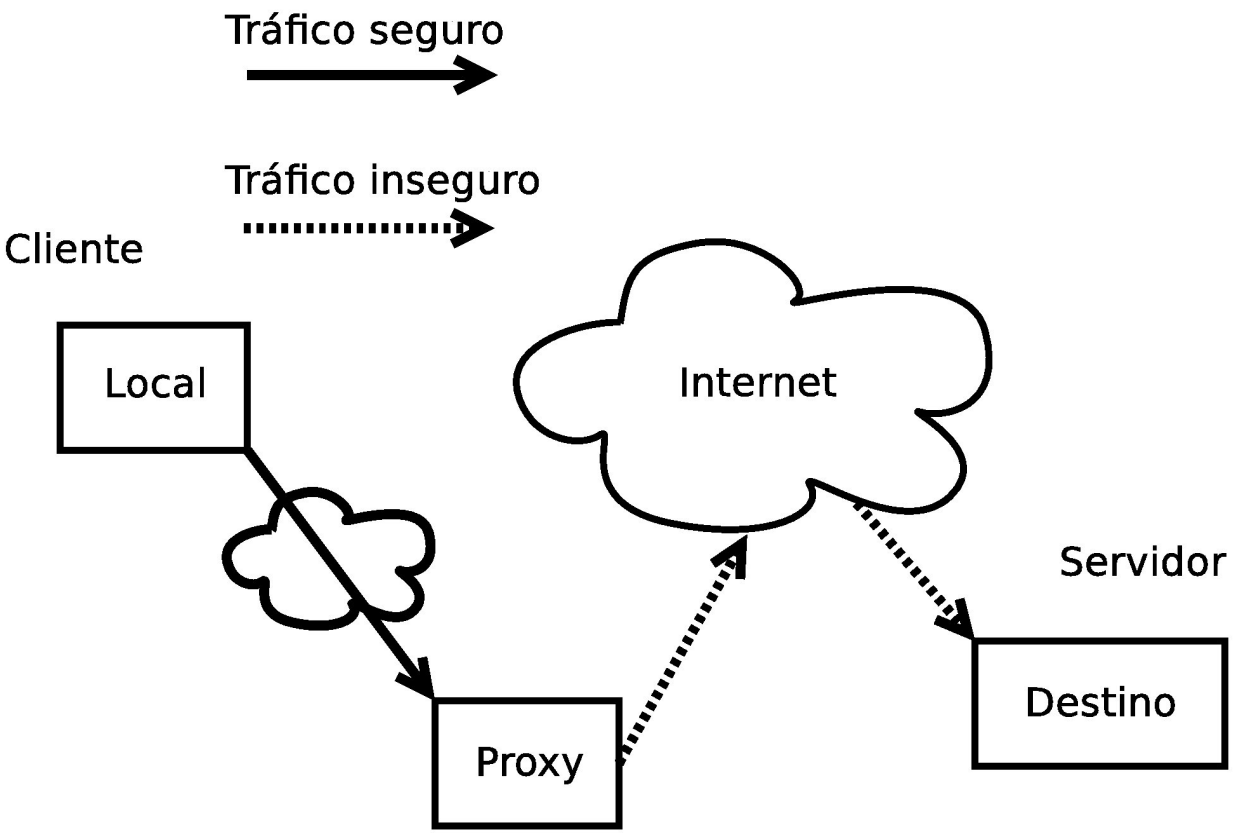
A través de un túnel, las conexiones a cierto puerto TCP de una máquina se redirigen a otro puerto TCP en otra máquina
Dos tipos:

- Túnel local, aka túnel (*a secas*). (*local tunnel, tunnel*)
- Túnel remoto, aka túnel inverso. (*remote tunnel, reverse tunel*)

Túnel local



Escenario típico donde usamos un servicio sobre un canal no seguro



Si tenemos cuenta en una máquina acccesible mediante ssh, podemos usarla como proxy

- Establecemos un túnel ssh desde la máquina local al proxy

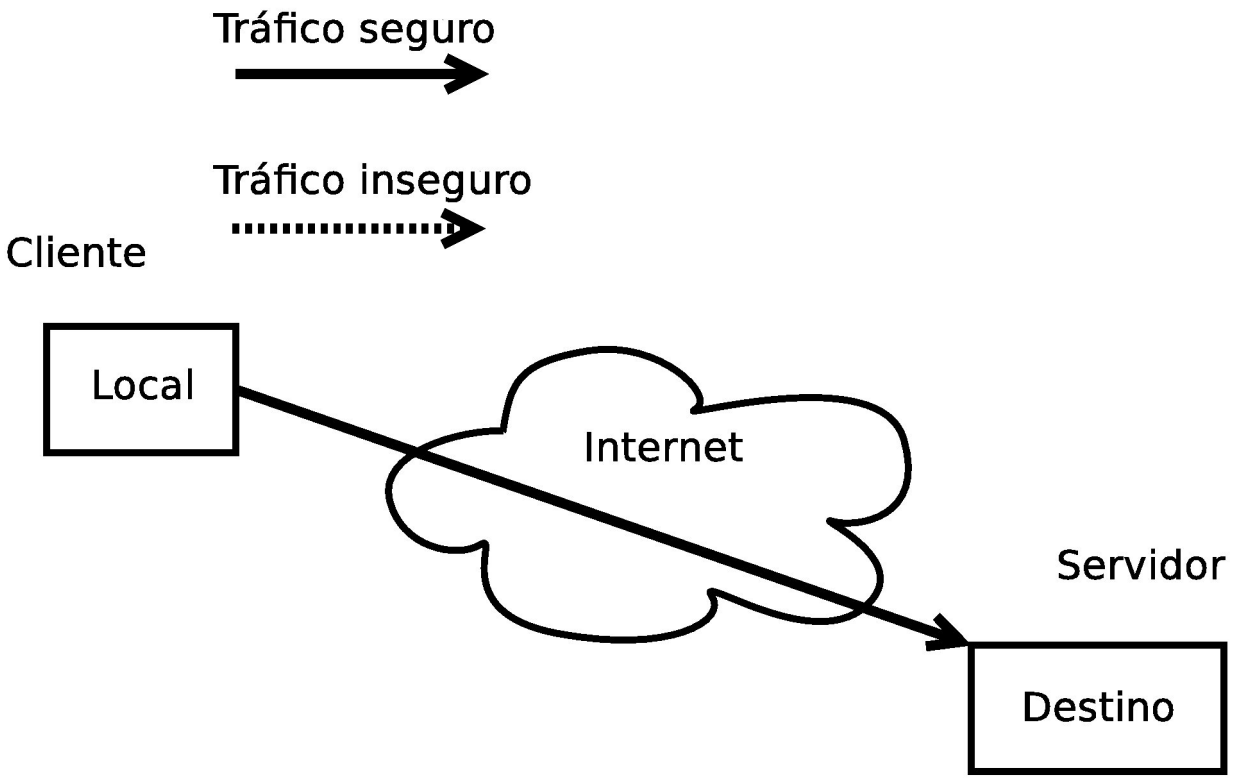
Ventajas del túnel local

- Permite asegurar el primer tramo, que suele ser el más peligroso
- Para el servidor, las peticiones vienen desde el proxy, no desde la máquina local. Esto es de utilidad
  - Si se trata de un servicio vinculado a la IP del cliente,
  - Para evitar cortafuegos, censura, etc
  - Burlar restricciones en la red del cliente

  El tráfico viaja cifrado en la red de la máquina local, el administrador de esa red pierde todo control sobre él
  - El administrador de la red local puede solucionar este problema prohibiendo todo tráfico cifrado, y con ello los túneles

Inconvenientes

- Encaminamiento en triángulo
- El proxy es un cuello de botella



En caso de que tengamos una cuenta en el servidor, accesible mediante ssh, podemos asegurar todo el trayecto

- Establecimiento del túnel. En `maquina_local` ejecutamos

  `ssh -L puerto_local:maquina_destino:puerto_remoto usuario@proxy`

- Uso del túnel:
  Indicamos al cliente que se conecte a `puerto_local` en `maquina_local`
  (El servicio estará realmente en el `puerto_destino` de `maquina_remota`)

- `ssh -L`
  además de redirigir los puertos, abre una sesión de shell ordinaria en el proxy
- `ssh -fNL`
  Hace que el túnel se lance en segundo plano (tras preguntar contraseñas), pero no abre una sesión de shell en el proxy
  - Esto puede ser conveniente para el usuario experimentado, así no ocupa el terminal
  - Pero despista al usuario principiante, pues no resulta tan claro si el desvío de puerto está activo o no

Ejemplo:
Acceder al servidor web en `bilo.gsyc.es`, usando `epsilon01` como proxy

- Establecemos el túnel en la máquina local:

  ```
  ssh -L 8080:bilo.gsyc.es:80 milogin@epsilon01.aulas.gsyc.es
  ```

- Usamos el túnel:
  En la máquina local, introducimos en el navegador web la url `http://localhost:8080`
  El cliente cree conectarse a su máquina local, de hecho eso hace. Pero el túnel redirige ese tráfico al proxy, y del proxy al destino

Proxy SOCKS

Un túnel local ordinario no sirve para navegar normalmente por el web

- La técnica anterior nos permite usar un túnel ssh para acceder a 1 servidor web
- Pero en cuanto hagamos clic sobre un enlace fuera de la máquina remota, dejamos de usar el proxy
- Para una sesión de navegación ordinaria tendríamos que abrir 10, 15, 20 túneles...

Pero openssh puede hacer *port forwarding* dinámico, como servidor del protocolo SOCKS

Configuración de proxy SOCKS

1. El usuario establece un túnel desde la máquina local hasta el proxy, añadiendo la opción -D e indicando un puerto local, con lo que se instala en la máquina local un servidor SOCKS
   - `ssh -D puerto_local usuario@proxy`

   El puerto estándar es el 1080, puede usarse cualquier otro
2. El usuario indica a su navegador web que todas las peticiones debe hacerlas al servidor SOCKS que está en `maquina_local:puerto_local`
3. El servidor de la máquina local pedirá al proxy que haga las peticiones, y este se las hará al servidor web
4. El servidor web responderá al proxy, que reenvía la respuesta al servidor SOCKS, de donde la lee el navegador web

Error frecuente: el usuario indica a su navegador que el servidor SOCKS está en el proxy. Esto es incorrecto. El servidor SOCKS está en la máquina local

La configuración del navegador es, obviamente, dependiente del navegador. Cualquier navegador con un mínimo de calidad permitirá hacerlo

- Firefox:

  `editar | preferencias | general | proxy de red | configuración manual del proxy | host SOCKS`

- Google Chrome
  Es necesario lanzarlo desde la shell con el parámetro adecuado
  - Linux

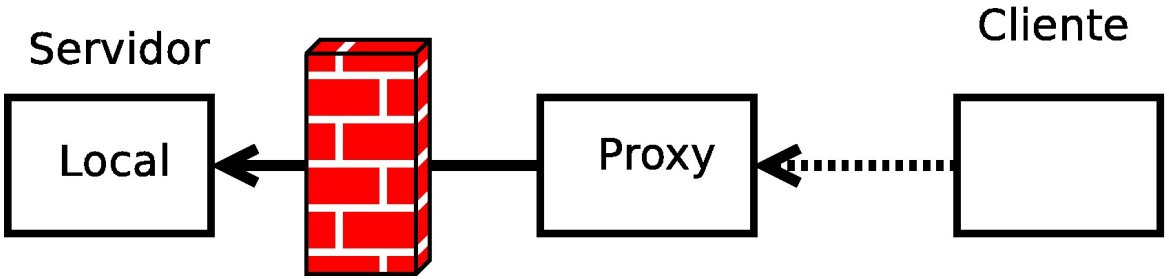
    ```
    google-chrome --proxy-server="socks://localhost:1080"
    ```

  - macOS

    ```
    open -a Google\ Chrome     \
    --args --proxy-server="socks://localhost:1080"
    ```

- Las conexiones ssh pueden caerse.
  En vez de `ssh`, podemos emplear `autossh`, que monitoriza la conexión y la reinicia si se cae
  Esto es útil combinado con el acceso automático sin contraseña
- La aplicación `tsocks` permite usar un proxy SOCKS de forma transparente a las aplicaciones (aplicaciones no preparadas para usar SOCKS)

Túnel remoto



Con un túnel remoto, aka túnel inverso, podemos traer a la máquina local las conexiones a cierto puerto del proxy
Esto puede tener al menos tres utilidades

1. Proteger el servidor
2. Distribuir servicios
3. Acceder a servidor tras NAT, sin configurar el NAT

Utilidad 1: Proteger el servidor

Un túnel inverso puede servir para proteger un servidor

- El proxy es necesariamente una máquina expuesta, puede necesitar gran visibilidad y muchos servicios (por ejemplo un servidor web)
- Pero el resto de los servicios, los colocamos en el servidor, en una máquina distinta, debidamente aislada

De esta forma, si un atacante comprometiera el proxy

- Tendríamos un problema, podría hacer p.e. un *website defacement*
- Pero el problema estaría contenido, no tendría acceso al resto de servicios, p.e. la base de datos

Naturalmente, cabe la posibilidad de que el atacante rompa la seguridad del túnel y acceda al servidor, pero esto añade una barrera adicional, relativamente robusta

Utilidad 2: Distribuir servicios

El proxy es una máquina distinta al servidor

- Puede ser útil para equilibrar la carga
- Puede ser útil si queremos combinar distinto software o distintos sistemas operativos

Utilidad 3:Acceder a servidor tras NAT, sin configurar el NAT

Tenemos un servidor tras un NAT

- La técnica habitual para permitir conexiones entrantes a este servidor es hacer *port forwarding* aka *abrir puertos* en el router que hace NAT
- Pero en vez esto, podemos conseguir un resultado similar, sin necesidad de modificar la configuración del router NAT

Usar un túnel ssh inverso (en vez del *port forwarding* tradicional), puede ser de utilidad:

- Si es un NAT que nosotros no podemos administrar (somos usuarios ordinarios, sin privilegios de administrador)
- Si *abrir los puertos* del NAT es incómodo
  - Hay un NAT tras otro NAT, tendría que configurar ambos
  - El NAT solo se puede configurar via web, típicamente a mano, resulta complicado automatizarlo
    (Mientras que el túnel inverso se prepara con 1 mandato desde la shell, fácil de incluir en un script, cron, etc)
  - Es necesario detener y reiniciar algún demonio (como el NAT de VirtualBox)

Inconvenientes de esta técnica:

- Dependemos de la existencia y disponibilidad del proxy
- El proxy necesita una IP pública
  - O bien el proxy está tras un NAT que sí podemos administrar. Pondríamos como dirección del proxy la del router que hace NAT, y redigiríamos a su vez esa conexión a una máquina de la red privada
- Solo es aplicable a TCP, no a UDP
- Estamos cifrando todo el tráfico (puede que no sea necesario)

- Establecimiento del túnel. En `maquina_local` ejecutamos

  ```
  ssh -R puerto_proxy:localhost:puerto_local  usuario@proxy
  ```

- Uso del túnel:
  Indicamos al cliente que se conecte a `puerto_proxy` en `proxy`
  El cliente cree conectarse al proxy, de hecho lo está haciendo.
  Pero el túnel redirige el tráfico al `puerto_local` de `localhost`, que es donde está el servicio
- Como en el túnel local, las opciones `-f` y `-N` son aplicables, con el mismo significado

Ejemplo: Tengo el servicio de escritorio remoto de `pilder01` en el puerto 5900 de la dirección privada 192.168.1.8
Quiero usar como proxy la máquina `miproxy.gsyc.es`, puerto 15900

- En `pilder01` ejecuto:

  ```
  ssh -R 15900:localhost:5900  milogin@miproxy.gsyc.es
  ```

- Para acceder a este servicio, el cliente ejecuta

  ```
  vinagre miproxy.gsyc.es:15900
  ```

  (el servicio está realmente en el puerto 5900 de pilder01)

Para que el cliente pueda estar en una máquina distinta a la máquina proxy, es necesario:

1. En el proxy, en en el fichero `/etc/ssh/sshd_config` añadir la entrada `GatewayPorts yes`
2. `sudo /etc/init.d/ssh restart`

Por omisión, esta opción no está activada (y solo `root` puede activarla)

Por tanto:

- Si tenemos una cuenta `ssh` ordinaria en el proxy, podemos hacer el túnel inverso, pero el cliente deberá estar en el mismo proxy
  - Además, el cliente deberá usar el nombre `localhost` y la dirección IP 127.0.0.1
    En el ejemplo anterior, el cliente solo podría estar en `proxy.gsyc.es` y debería ejecutar

    ```
    vinagre localhost:15900
    ```

    o bien

    ```
    vinagre 127.0.0.1:15900
    ```

    Pero no podría usar el nombre `proxy.gsyc.es`
- Si tenemos privilegios de adminitrador en el proxy y añadimos a `sshd_config` la opción `GatewayPorts yes`, el cliente podrá estar en cualquier lugar

Otro ejemplo de túnel inverso

- `mortuno@gsyc:~$ ssh -R 8080:localhost:80 mortuno@miproxy.gsyc.es`

- El cliente accede a
  `http://myproxy.gsyc.es:8080`
  donde verá
  `http://gsyc.es`

Resumen

Recapitulando, resumimos así el uso de los túneles directo e inverso

- En todos los casos:
  El ssh se hace desde la máquina local hasta el proxy (no hasta la máquina remota)
- Túnel directo:

```
ssh -L puerto_local:maquina_destino:puerto_remoto usuario@proxy
```

  - El servicio está en `maquina_destino:puerto_remoto`
  - El cliente está en la máquina local, se conecta al puerto local
- Túnel inverso:

```
ssh -R puerto_proxy:localhost:puerto_local usuario@proxy
```

  - El servicio está en `máquina local:puerto_local`
  - El cliente está en una máquina remota cualquiera, desconocida. Se conecta a `proxy:puerto_proxy`

scp

scp va sobre ssh, por tanto

- Usa el mismo servidor (sshd), escuchando en el mismo puerto (22)
- Si preparamos un túnel para entrar en el servidor ssh de una máquina, también podemos hacer scp a esa máquina
- La única diferencia es que, en el cliente, para indicar el puerto
  - ssh usa `-p` (minúscula)
  - scp usa `-P` (mayúscula)

Ejemplo

Tenemos la máquina virtual `pc01`, en el *host* `zeta01`, conectada a la red a través de NAT

- Establecemos el túnel (en este caso, remoto)

```
user@pc01:~$ ssh -R 2222:localhost:22 milogin@zeta01
```

- Desde el *host*, accedemos al servidor de ssh en `pc01`

```
milogin@zeta01:~$ ssh -p 2222 user@localhost  # p minúscula
```

- Desde el *host*, copiamos el fichero `holamundo.txt` al directorio `/tmp` de `pc01`

```
milogin@zeta01:~$ scp -P 2222 holamundo.txt user@localhost:/tmp
                                                # P mayúscula
```

autossh

Una conexión ssh en desuso se cortará automáticamente. Podemos evitarlo en el lado del cliente añadiendo en `.ssh/config`

```
host *
    ServerAliveInterval 60
```

O mejor aún, usando `autossh`

```
apt update; apt upgrade; apt install -y autossh
```

No lanzaremos ssh directamente desde el terminal, sino desde autossh. Típicamente con opciones como

```
-M 0 -o "ServerAliveInterval 30" -o "ServerAliveCountMax 3"
```

Esto hace que cada 30 segundos se compruebe la conexión. Y se relance ssh si falla 3 veces seguidas.

```
#!/bin/bash
PUERTO_PROXY=9999
USUARIO=jperez
PROXY=miproxy
PUERTO_LOCAL=22
autossh -M 0 -o "ServerAliveInterval 30" -o "ServerAliveCountMax 3" \
   -R ${PUERTO_PROXY}:localhost:${PUERTO_LOCAL} ${USUARIO}@${PROXY}
```

Más información en este enlace:
ssh tunnelling for fun and profit autossh