Práctica 9: Empleando SSH

Nota: Para desarrollar la práctica se empleará una máquina virtual que contiene una instalación de Ubuntu GNU/Linux 20.04 de 64 bits. Se proporciona fichero OVA en Moovi. Para el uso de este fichero se debe contar con VirtualBox. La máquina virtual tiene un usuario "adminso2" con clave "adminso2" siendo posible la elevación de privilegios mediante el comando *sudo*. El puerto 2222 de la máquina virtual se encuentra mapeado con la interfaz de bucle local del equipo anfitrión lo que permite conexiones ssh empleando el comando "*ssh -p 2222 adminso2@localhost*".

Nota 2: Para facilitar la realización de pruebas se ha habilitado en el equipo so2.atopa.me un servidor OpenSSH y una cuenta *testuser* con clave *12test34*. La ejecución de comandos en este equipo es limitada (ya que se ha creado un jail) y por tanto es probable que no estén disponibles muchos comandos. Pero servirá para hacer redirección de puertos y otras funcionalidades de SSH (particularmente para desarrollar la prueba final).

Nota 3: Ningún usuario distinto (salvo root) puede crear un socket de servidor en un puerto inferior o igual a 1023. Y por supuesto un puerto TCP ó UDP sólo puede ser ocupado por un socket de servidor (no puede haber dos procesos escuchando en el mismo puerto). Emplear *ss -ptan* para detectar problemas derivados de estas limitaciones.

La presente práctica se encuadra en el contexto de los mecanismos disponibles en Ubuntu (o Debian) y recoge una serie de utilidades de SSH.

SSH (o Secure SHell) es el nombre de un protocolo y del programa que lo implementa cuya principal función es el acceso remoto a un servidor por medio de un canal seguro en el que toda la información está cifrada. Además del establecimiento de sesiones remotas seguras (función que conoce la mayoría de usuarios), SSH permite:

- Copiar archivos/directorios de forma segura (incluso simular sesiones FTP cifradas)
- Gestionar claves RSA como mecanismo alternativo de autenticación evitando así, la necesidad de escribir contraseñas al conectar a los dispositivos
- Enviar datos de cualquier otra aplicación por un canal seguro desde un punto hasta otro.
- Mapeo de puertos del equipo (o de otros equipos de la red local) con otros del equipo (o equipo de la red) remoto/a.
- Trasvase de tráfico de un servidor a otro mediante servidores socks.
- Redirigir el tráfico del (Sistema de Ventanas X) para poder ejecutar programas gráficos remotamente.

El puerto TCP asignado es el 22.

9.1. Sesiones remotas SSH

Existen multitud de clientes SSH para distintas plataformas (muchos alumnos conocerán probablemente Putty que está disponible para diversas plataformas incluyendo Windows y Linux) que permiten la ejecución de comandos de forma remota. El uso de estos clientes SSH permite usar la funcionalidad SSH pero de una forma mucho más limitada. En este apartado, el alumno debe familiarizarse con el uso del cliente ssh (interfaz command-line) proporcionada por openssh. Dado que esta implementación es command line, se permitirá la conexión ssh desde un equipo a otros equipos remotos sin necesidad de emplear para nada el interfaz gráfico. El cliente openssh, que se provee en el paquete *openssh-client*, incorpora toda la funcionalidad propia de SSH y permite combinar la ejecución remota y local de comandos.

En este sentido se puede usar la sintaxis básica de conexión con ssh <user>@<host> donde <user> es el nombre de usuario que se va a emplear para la autentificación y <host> es el equipo que se desea conectar bien sea especificado por IP o por nombre. En este caso se establecerá una sesión remota similar a la establecida con Putty con la diferencia que usa únicamente una interfaz de texto.

Al conectar, SSH emplea un mecanismo similar a SSL sólo que no existe la figura de la autoridad certificadora que garantiza que el equipo al que se intenta conectar es el correcto. Para este fin, SSH sustituye la autoridad certificadora por una comprobación manual de la clave pública del servidor. Al conectar por primera vez, se pregunta si el fingerprint de la clave pública es el correcto.

```
$ ssh -p 2222 user@localhost
The authenticity of host '[localhost]:2222 ([127.0.0.1]:2222)' can't be established.
ECDSA key fingerprint is SHA256:XH8IXNOQ3hpwPBxFzu3npW8Qu27ek6IXALzKe6MLe80.
Are you sure you want to continue connecting (yes/no)?
```

Para obtener el fingerprint del servidor se buscará la clave pública correspondiente (en este caso la ECDSA) y se calculará su fingerprint con el comando ssh-keygen.

```
$ ssh-keygen -lf ssh_host_ecdsa_key.pub
256 SHA256:XH8IXNOQ3hpwPBxFzu3npW8Qu27ek6IXALzKe6MLe80 root@debian10 (ECDSA)
```

En la sesión remota se emplea el shell especificado para el usuario en el fichero */etc/passwd* (habitualmente */bin/bash*) y al abandonar el shell (en *Bash* usando el comando *exit*) la sesión SSH se cerrará.

Sin embargo, el cliente openssh provee la posibilidad de emplear una segunda forma que va a ejecutar un comando remotamente y abandonar la sesión. Cuando el comando contiene espacios se coloca entre comillas para indicar que se trata de un único parámetro. Para el uso

de esta forma de llamada simplemente hay que especificar el comando a continuación. Por ejemplo se puede hacer un listado de usuarios conectados al equipo remoto 10.0.2.15 usando el siguiente comando.

```
# ssh user@10.0.2.15 who
user@10.0.2.15's password:
root     tty1         Mar  5 14:02
user     pts/0        Mar 10 14:26 (10.0.2.2)
user     pts/1        Mar  5 14:03 (10.0.2.2)
```

Ejercicio: obtener el porcentaje de memoria RAM utilizada en el equipo remoto.

Algo muy interesante del cliente openssh es que interconecta la entrada/salida del comando ejecutado remotamente con el propio comando *ssh* que se ejecuta localmente. Esto permite que se ejecuten con éxito comandos como los siguientes:

```
# # ver las diferencias entre un fichero local y el mismo en remoto
# ssh administrador@localhost "cat ~/.bash_history" | diff ~/.bash_history -
# # contar todos los procesos en todos los equipos
# suma=0; for i in 192.168.5.2 192.168.5.3; do suma=$(expr $suma + $(ssh user@$i "ps aux | wc -l") );
done; echo "Sum of all users: $suma"
user@192.168.5.2's password:
user@192.168.5.3's password:
Sum of all users: 159
# # copiar bit a bit el contenido de un dispositivo de bloques
# dd if=/dev/sda1 bs=512 | ssh user@192.168.5.2 "dd of=/dev/sda bs=512"
```

Ejercicio: marcar en los comandos anteriores qué parte se ejecuta localmente y cuál remotamente.
Ejercicio 2: Diseña un comando para ver cuántos usuarios tienen asociado el shell /bin/bash en dos o más equipos usando ssh y awk.

```
# suma=0; for i in 192.168.5.2 192.168.5.3; do suma=$(expr $suma +  $(ssh user@$i "awk -F: 'BEGIN
{count=0} /bin\/bash$/ {count++} END {print count}' /etc/passwd" ) ); done; echo "Sum of users with bash
as shell: $suma"
user@192.168.5.2's password:
user@192.168.5.3's password:
Sum of users with bash as shell: 4
```

Ejercicio 3: En el script anterior, el recuento de usuarios (que se hace mediante el comando awk) se hace en el equipo remoto. Transforma el comando anterior para que esta activada (la ejecución del awk) se haga en el equipo local.

9.2. Transferencia de ficheros por SSH

La transferencia de ficheros por ssh podría ser hecha de una forma muy limitada mediante la ejecución remota de comandos cat y su guardado en disco local. Por ejemplo, el comando *ssh user@192.168.5.2 "cat /etc/passwd" > ./passwd* permitiría copiar el archivo remoto */etc/password* aunque no preservará ningún atributo de los ficheros. Este tipo de soluciones es bastante deficiente ya que SSH incorpora mecanismos específicos para copiar ficheros. Entre

ellos se incluyen las herramientas scp, sftp, la posibilidad de combinar ssh con rsync y la existencia de un sistema de ficheros que permite montar localmente carpetas remotas transfiriendo su contenido por SSH. Estas herramientas se muestran en los siguientes apartados.

9.2.1. scp

Esta herramienta se distribuye junto con el cliente de ssh openssh. El comando *scp* es equivalente (en las opciones comúnmente usadas con él al comando *cp* de copia de ficheros en local. La principal diferencia radica en que *scp* permite especificar un origen o un destino remoto (en ningún caso ambos). Para especificar una ruta remota se debe anteponer a la ruta una cadena formada por *<usuario>@<host>:* donde <*usuario*> es el usuario de la conexión ssh y <*host*> es el equipo remoto.

La opción más comúnmente empleada con *scp* es *-r* que indica que se copiaran recursivamente ficheros y directorios.

A continuación se muestran algunos ejemplos de uso:

```
# scp user@192.168.5.2:/etc/passwd .
# scp /etc/passwd user@192.168.5.2:
# scp user@192.168.5.2:~/* .
# scp -r user@192.168.5.2:/etc .
```

9.2.2. sftp

El programa *sftp* implementa la funcionalidad propia del cliente de ftp de FTP en modo texto pero realizando transferencias mediante el protocolo SSH. Esta misma funcionalidad de FTP sobre SSH (SFTP) se puede emplear con clientes gráficos como Filezilla. A continuación se muestra un pequeño ejemplo que pretende ser autoexplicativo del uso del cliente sftp en modo texto.

```
# sftp user@192.168.5.2
user@192.168.5.2's password:
Connected to user@192.168.5.2.
sftp> lpwd
Local working directory: /root
sftp> pwd
Remote working directory: /home/user
sftp> lls
cert.crt  client.crt  client.pem  myCA.key  server.crt   server.pem
cert.key  client.csr  file.txt     myCA.srl  server.csr   xx
cert.pem  client.key  myCA.crt     response  server.key
sftp> ls
mapmem          mapmem.c        mul.c           mul.s           param
stack_over      stack_over.c    stack_over.s    xx.sh
sftp> put response
Uploading response to /home/user/response
response                                      100%   98   133.8KB/s   00:00
```

```
sftp> mput *.crt
Uploading cert.crt to /home/user/cert.crt
cert.crt                                      100% 1415     2.6MB/s   00:00
Uploading client.crt to /home/user/client.crt
client.crt                                    100% 1310     3.0MB/s   00:00
Uploading myCA.crt to /home/user/myCA.crt
myCA.crt                                      100% 1432     1.9MB/s   00:00
Uploading server.crt to /home/user/server.crt
server.crt                                    100% 1310     2.3MB/s   00:00
sftp> get param
Fetching /home/user/param to param
/home/user/param                              100%   31    12.5KB/s   00:00
sftp> mget *.c
Fetching /home/user/mapmem.c to mapmem.c
/home/user/mapmem.c                           100%  760   297.7KB/s   00:00
Fetching /home/user/mul.c to mul.c
/home/user/mul.c                              100%  262   173.3KB/s   00:00
Fetching /home/user/stack_over.c to stack_over.c
/home/user/stack_over.c                       100%  244   165.8KB/s   00:00
sftp> cd ..
sftp> pwd
Remote working directory: /home
sftp> lcd /
sftp> lpwd
Local working directory: /
sftp> cd user
sftp> ls
cert.crt        client.crt      mapmem          mapmem.c        mul.c
mul.s           myCA.crt        param           response        server.crt
stack_over      stack_over.c    stack_over.s    xx.sh
sftp> rm response
Removing /home/user/response
sftp> rm *.crt
Removing /home/user/cert.crt
Removing /home/user/client.crt
Removing /home/user/myCA.crt
Removing /home/user/server.crt
sftp> mkdir kk
sftp> ls -l
drwxr-xr-x    2 user     user         4096 Mar 13 00:29 kk
-rwxr-xr-x    1 user     user       679612 Mar  2 10:00 mapmem
-rw-r--r--    1 user     user          760 Mar  2 09:57 mapmem.c
-rw-r--r--    1 user     user          262 Mar  2 11:02 mul.c
-rw-r--r--    1 user     user         1607 Mar  2 20:59 mul.s
-rw-r--r--    1 root     root           31 Mar  2 12:47 param
-rwxr-xr-x    1 user     user       673800 Mar  2 12:23 stack_over
-rw-r--r--    1 user     user          244 Mar  2 12:16 stack_over.c
-rw-r--r--    1 user     user         1233 Mar  2 12:18 stack_over.s
-rwxr-xr-x    1 root     root          196 Mar  2 12:47 xx.sh
sftp> rmdir kk
sftp> exit
```

10.2.3. Combinando SSH y rsync

El comando rsync permite sincronizar dos ficheros o carpetas (una de ellas remota usando ssh). Es muy importante para que funcione correctamente que el comando esté instalado en ambos extremos de la sincronización. La herramienta rsync detecta dónde se han producido los cambios (en qué extremo) y mantiene en ambos extremos la última versión.

A continuación se muestra la instalación de rsync junto con ejemplos en los que se sincroniza remotamente un fichero y una carpeta (opción -r).

```
# apt-get install rsync
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  rsync
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 397 kB of archives.
After this operation, 746 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian buster/main amd64 rsync amd64 3.1.3-6 [397 kB]
Fetched 397 kB in 1s (610 kB/s)
Selecting previously unselected package rsync.
(Reading database ... 32886 files and directories currently installed.)
Preparing to unpack .../rsync_3.1.3-6_amd64.deb ...
Unpacking rsync (3.1.3-6) ...
Setting up rsync (3.1.3-6) ...
Created symlink /etc/systemd/system/multi-user.target.wants/rsync.service ->
/lib/systemd/system/rsync.service.
Processing triggers for man-db (2.8.5-2) ...
Processing triggers for systemd (241-7~deb10u3) ...
# ssh user@192.168.5.2 "sudo 'apt-get install rsync'"
user@192.168.5.2's password:
Password:
Reading package lists...
Building dependency tree...
Reading state information...
The following NEW packages will be installed:
  rsync
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 397 kB of archives.
After this operation, 746 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian buster/main amd64 rsync amd64 3.1.3-6 [397 kB]
debconf: unable to initialize frontend: Dialog
debconf: (TERM is not set, so the dialog frontend is not usable.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (This frontend requires a controlling tty.)
debconf: falling back to frontend: Teletype
dpkg-preconfigure: unable to re-open stdin:
Fetched 397 kB in 1s (752 kB/s)
Selecting previously unselected package rsync.
(Reading database ... 38398 files and directories currently installed.)
Preparing to unpack .../rsync_3.1.3-6_amd64.deb ...
Unpacking rsync (3.1.3-6) ...
Setting up rsync (3.1.3-6) ...
Created symlink /etc/systemd/system/multi-user.target.wants/rsync.service ->
/lib/systemd/system/rsync.service.
Processing triggers for man-db (2.8.5-2) ...
Processing triggers for systemd (241-7~deb10u3) ...
# rsync -v -e ssh /home/user/.bash_history user@192.168.5.2:~/.bash_history
user@192.168.5.2's password:
.bash_history

sent 156 bytes  received 41 bytes  131.33 bytes/sec
total size is 61  speedup is 0.31
# ssh user@192.168.5.2 "cat .bash_history" | diff - /home/user/.bash_history
user@192.168.5.2's password:
# ssh user@192.168.5.2 "mkdir ~/sync"
user@192.168.5.2's password:
```

```
# rsync -r -v -e ssh /home/user/ user@192.168.5.2:~/sync
user@192.168.5.2's password:
sending incremental file list
.bash_history
.bash_logout
.bashrc
.profile
sendTelegram.sh

sent 5,754 bytes  received 111 bytes  2,346.00 bytes/sec
total size is 5,363  speedup is 0.91
# ssh user@192.168.5.2 "ls -laR ~/sync"
user@192.168.5.2's password:
/home/user/sync:
total 28
drwxr-xr-x 2 user user 4096 Mar 13 00:47 .
drwxr-xr-x 3 user user 4096 Mar 13 00:44 ..
-rw------- 1 user user   61 Mar 13 00:47 .bash_history
-rw-r--r-- 1 user user  220 Mar 13 00:47 .bash_logout
-rw-r--r-- 1 user user 3526 Mar 13 00:47 .bashrc
-rw-r--r-- 1 user user  807 Mar 13 00:47 .profile
-rw-r--r-- 1 user user  749 Mar 13 00:47 sendTelegram.sh
```

Como puede deducirse, el uso de rsync es especialmente útil para la elaboración de copias de seguridad de forma sencilla. De hecho, combinado con la autenticación basada en criptografía asimétrica (RSA), puede llamarse directamente desde una tarea programada (crontab).

9.2.4. sshfs

Como su nombre indica, *sshfs* es un sistema de ficheros que se puede emplear mediante mount y que permite montar remotamente un sistema de ficheros. Para su utilización es necesario instalarlo (paquete sshfs). Una vez que se ha instalado sólo es necesario emplear el comando mount para hacer disponible una carpeta remota en un sistema local. A continuación se muestra un ejemplo:

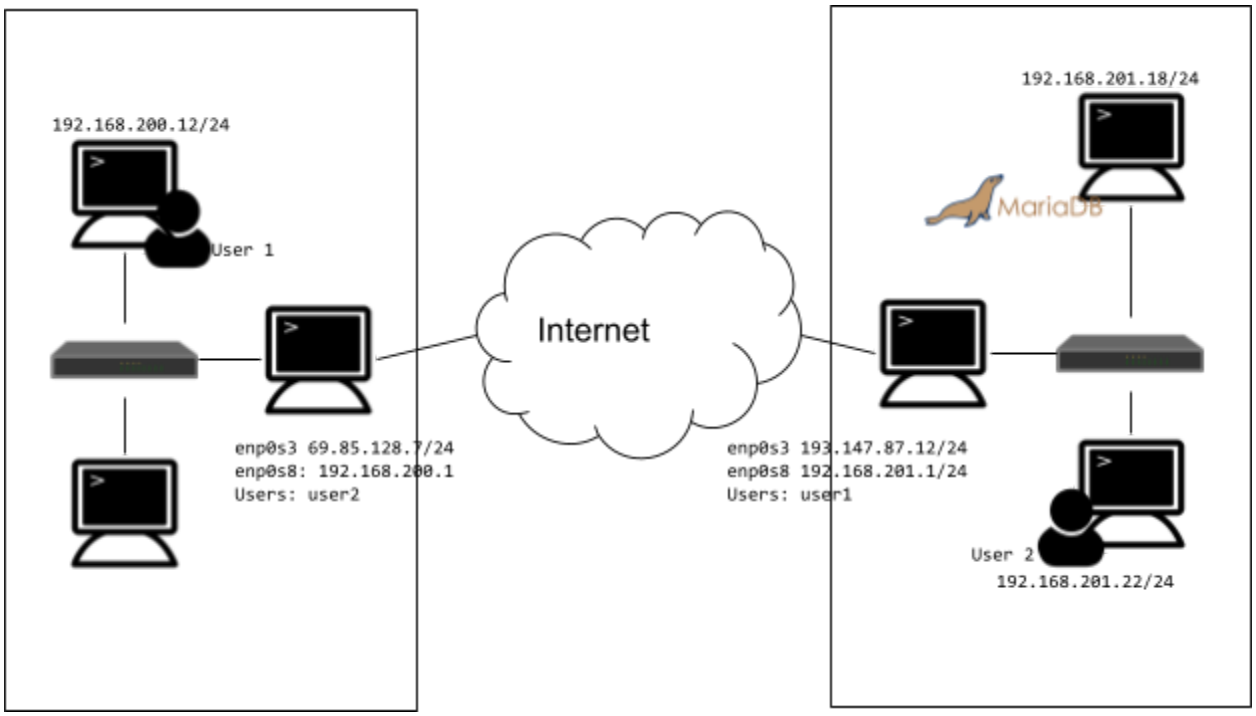
```
# mkdir test_mount
# sshfs user@192.168.5.2:/home/user/sync ./test_mount
user@192.168.5.2's password:
# ls test_mount/
sendTelegram.sh
# umount ./test_mount
# ls test_mount/
```

9.3. Port forwarding

SSH permite redirigir el tráfico entre puertos TCP a través de la conexión SSH cifrada. Esta característica ha supuesto una ventaja de seguridad bastante importante que ha condicionado incluso la propia configuración distribuida con los paquetes de algunas distribuciones.

Para entenderlo de una forma gráfica la idea consiste en que SSH permite disponer en uno de los extremos ssh (**L**ocal o **R**emoto) de la posibilidad de hacer una conexión TCP como cliente

(socket de cliente) que sin SSH sólo sería posible si se realizara desde el otro extremo (y cifrar los contenidos intercambiados).



Escenario 1: Pongamos un ejemplo práctico. Imagina la situación dibujada en la siguiente imagen y que el usuario user1 se encuentra en el equipo 192.168.200.12. Supongamos que este usuario tiene una cuenta ssh en el equipo 193.147.87.12 y que quiere conectarse a un servidor mysql/mariadb que se está ejecutando en el puerto 3306 del equipo 192.168.201.18 con un cliente gráfico (Mysql Workbench, por ejemplo). Sería imposible hacerlo según las reglas de TCP/IP porque ambos equipos están en redes privadas detrás de distintos routers. Sin embargo SSH va a permitir indicar que cualquier conexión que se reciba en un puerto elegido por el usuario (podría ser el mismo 3306) del equipo 192.168.200.12 se redireccione automáticamente mediante SSH al servidor 193.147.87.12 y desde allí, al puerto 3306 del equipo 192.168.201.18. Esto se conoce como *local port forwarding*.

Escenario 2: Además, también se permite definir esta operación a la inversa. Supongamos que el usuario user2 (en el equipo 192.168.201.22) tiene una cuenta de usuario en el equipo 69.85.128.7 y que quiere ayudar a user1 que quiere conectarse al servidor mysql. Para hacerlo decide hacer disponible el puerto 3306 en un puerto escogido por el usuario del servidor remoto 69.85.128.7 para que el usuario user1 se pueda conectar a este puerto. Esto es lo que se conoce como remote port forwarding. Básicamente el servidor SSH en el equipo 69.85.128.7 escucharía en el puerto seleccionado por el usuario recogiendo los paquetes que serían enviados al servidor mysql de forma encriptada.

Los siguientes subapartados recogen de forma detallada las posibilidades de reenvío de puertos ofrecida por SSH y sus utilidades.

9.3.1. Local port forwarding

Como se ha comentado previamente (escenario 1 descrito), el local port forwarding implica que poder abrir desde el cliente de ssh un socket de cliente a cualquier servicio TCP accesible desde el servidor. Con este objetivo se usa la opción *-L* con una descripción similar a lo siguiente:

```
ssh -L <puerto_local>:<especificación_socket_remoto> <usuario>@<equipo_remoto>
Donde:
    • <puerto_local>: Será el puerto donde el cliente ssh escuchará para redirigir la conexión (SSH hará creará un socket de servidor en <puerto_local> pero sólo escuchará en la interfaz de bucle local). El usuario puede escoger el puerto que crea conveniente siempre que no esté ocupado.
    • <especificación_socket_remoto>: Es el socket de cliente que se va a realizar en remoto cuando se acceda al <puerto_local>. En el escenario 1 del ejemplo de arriba esta especificación sería 192.168.201.18:3306.
    • <usuario>@<equipo_remoto>: Como cualquier otra conexión ssh se especifica el usuario y el puerto remoto.
El escenario 1 podría resolverse como:
ssh -L 3306:192.168.201.18:3306 user1@193.147.87.12
```

El local port forwarding está presente ya en muchas configuraciones de Ubuntu o Debian. Así, por ejemplo los servidores de base de datos mysql/mariadb proporcionados por estas distribuciones están configurados para escuchar sólo en la interfaz de bucle local. Aunque pueda parecer que una conexión remota es imposible de realizar, el port forwarding de SSH permite conexiones de cualquier software de gestión con estos servidores aun cuando tienen esta configuración por defecto.

Ejercicio: Piensa en la situación descrita anteriormente. Indica el comando para establecer una conexión a través de SSH a un servidor mysql remoto en el supuesto que este servidor mysql sólo escuche en la interfaz de bucle local (127.0.0.1). Supón que el servidor donde está instalado el mysql tiene una IP pública 193.147.87.45 y dispone del usuario myuser con acceso ssh.
El comando sería: *ssh -L 3306:localhost:3306 myuser@193.147.87.45* y habría que usar para la conexión del cliente el equipo localhost y el puerto 3306.

10.3.1. Remote port forwarding

El remote port forwarding permite justo lo contrario del local port forwarding. La idea es que desde el servidor remoto un usuario pueda crear un socket de cliente a un equipo que sólo se puede acceder desde la red del cliente. Para esto se emplea la opción -R y se usa la siguiente sintaxis.

```
ssh -R <puerto_remoto>:<especificación_socket_local> <usuario>@<equipo_remoto>
```

Donde:

- <puerto_remoto>: Será el puerto donde el cliente ssh escuchará para redirigir la conexión (SSH creará un socket de servidor en <puerto_local> pero sólo escuchará en la interfaz de bucle local). El usuario puede escoger el puerto que crea conveniente siempre que no esté ocupado.
- <especificación_socket_local>: Es el socket de cliente que se va a realizar en remoto cuando se acceda al <puerto_remoto>. En el escenario 2 del ejemplo de arriba esta especificación sería 192.168.201.18:3306.
- <usuario>@<equipo_remoto>: Como cualquier otra conexión ssh se especifica el usuario y el puerto remoto.

El escenario 2 podría resolverse como:

```
ssh -R 3306:192.168.201.18:3306 user2@69.185.128.7
```

Una vez hecho esto en el equipo con la IP 69.85.128.7 habría un socket de servidor en el puerto 3306 de la interfaz de bucle local (127.0.0.1).

Supón una máquina virtual (VirtualBox, por ejemplo) a la que se le ha configurado una interfaz de red virtual que funciona en modo NAT. Esto implica que la máquina virtual puede establecer conexiones de red con equipos pertenecientes a cualquier red accesible desde el equipo anfitrión (incluido el propio equipo anfitrión). Sin embargo, las conexiones en sentido contrario (que el equipo anfitrión o cualquier equipo de su red se conecte a la máquina virtual) no serían posibles. Bajo esta situación, supón que estás probando una aplicación web en la máquina virtual, que la máquina virtual no tiene un sistema de escritorio y que quieres ver cómo se visualiza la aplicación en un navegador de escritorio como Firefox o Chrome que sí están instalados en el equipo anfitrión. Para hacer este tipo de tarea, el uso de un reverse port forwarding haciendo una conexión desde la máquina virtual al equipo anfitrión sería lo más adecuado.

Ejercicio: Suponiendo la situación mencionada antes resolveremos el comando para probar la aplicación con Chrome. Para ello asumimos que el servidor web instalado en la máquina virtual escucha en el puerto 80 de todas las interfaces, que la IP del equipo anfitrión es 192.168.112.45 y que se dispone del usuario myuser que puede establecer una sesión ssh a la máquina anfitrión.
El comando sería (ejecutado como cualquier usuario): *ssh -R 8080:localhost:80 myuser@*192.168.112.45. En el navegador habría que usar la URL http://localhost:8080. Recordad que no se puede crear un serversocket en un puerto por debajo de 1023 salvo que el usuario sea el root.
Ejercicio 2: Y si el servidor web fuera HTTPS?

9.4. Reenvío del tráfico IP de forma segura mediante un servidor SOCKS

Una posibilidad interesante de SSH es que permite reenviar tráfico a un equipo servidor para que sea enrutado desde el servidor remoto (y no desde el equipo local) y con la dirección origen de aquel servidor. Para ello el cliente SSH crea un servidor SOCKS escuchando en un puerto (especificado por el usuario) en la interfaz de bucle local. Todo el tráfico enviado por ese servidor SOCKS será enviado por medio de SSH al servidor remoto y enrutado desde allí. Esto se hace con la opción -D:

```
ssh -D 9000 <usuario>@<equipo_remoto>
    • -D 9000 es el puerto que se escuchará peticiones socks. Sólo será accesible desde la interfaz
      de bucle local.
    • <usuario>@<equipo_remoto>: es el usuario y el equipo remoto que es necesario especificar en
      cualquier conexión ssh.
```

Así por ejemplo, en las redes universitarias del SUG contamos con acceso a varios portales en los que se realiza una autenticación por IP. Concretamente hay varios contratos que el Consorcio de Bibliotecas Universitarias de Galicia (BUGalicia) gestiona incluyendo distintas bases de datos (Sabi, JCR) y acceso a revistas científicas de forma ilimitada. Si contamos con un servidor SSH en una de las redes universitarias, es posible crear un servidor SOCKS que recoja el tráfico del equipo, lo lleve al servidor SSH y este lo enrute desde su ubicación y usando su propia dirección IP.

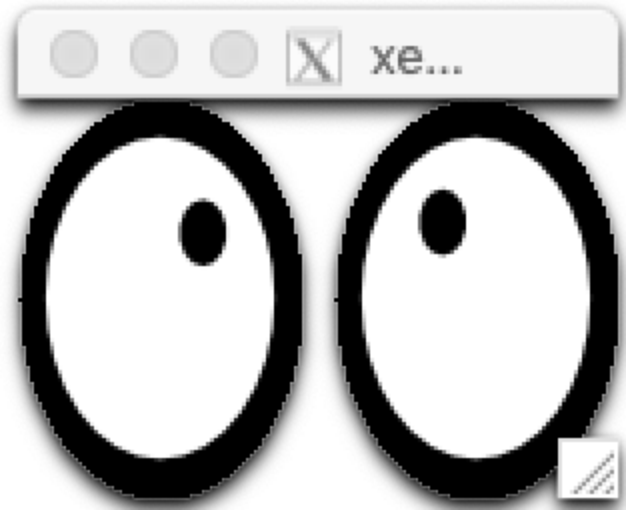
Ejercicio: Supón la situación anterior. Indica los pasos para emplear Firefox para acceder a un portal de una revista de forma ilimitada desde tu casa. Supón que tienes la cuenta myuser en el equipo 193.147.87.45 que se encuentra en la red de la Universidad de Vigo.
El comando sería *ssh -D 9000 myuser@193.147.87.45*. Luego habría que configurar Firefox. Para ello se puede poner en la barra de dirección *about:preferences*, buscar los parámetros de Conexión y cambiar la configuración proxy para que usar el servidor SOCKS (versión 4a ó 5) *127.0.0.1* puerto *9000*. Todo el tráfico generado por Firefox (la resolución DNS es configurable) aparecería como generado desde el equipo *193.147.87.45*.

9.5. X Forwarding

Otra gran utilidad de ssh es permitir ejecutar una aplicación XWindow en un equipo remoto por medio de SSH y representar la interfaz de usuario en el equipo local. Si se usan un equipo Microsoft Windows o MAC es necesario contar con un aplicación servidor X local. En el caso de MAC el servidor es XQuartz y se puede descargar de http://xquartz.org. En el caso de Windows se puede usar XMing ó XcXSrv (https://sourceforge.net/projects/vcxsrv/).
Para esta funcionalidad (XForwarding) se emplea el parámetros -X ó -Y. El uso el primero implica que el servidor remoto no es de absoluta confianza y por tanto le se limitarán las operaciones que las aplicaciones remotas puedan hacer de los datos locales (ssh asegurará que no se pueden hacer screenshots, no se puede hacer keylogging y otras operaciones peligrosas). En el caso de -Y se asume que el servidor remoto es de absoluta confianza y se permite una conectividad completa y sin restricciones. Normalmente empleando -X no se observa ninguna limitación en las aplicaciones por lo que su uso será suficiente en la mayoría de los casos.

```
$ssh user@remote “apt-get install x11-apps”
```

```
# ssh -Y user@remote xeyes
```



9.6. Autenticación RSA

Una conexión usando el protocolo SSH es de por sí segura (más segura que una conexión no cifrada, como telnet) pero presenta ciertos inconvenientes. El primero de ellos hace referencia al uso de contraseñas normales que, como viajan cifradas, no se podrán cifrar. Sin embargo, aunque las claves fuertes de los usuarios podrían limitar el impacto de seguridad, se deja abierta la posibilidad de hacer ataques de fuerza bruta que, si tienen éxito podrían culminar con efectos graves.

Ante este desafío con SSH se creó un mecanismo de autentificación basado en desafío y en criptografía asimétrica. El cliente cuenta con una clave privada. La clave pública correspondiente se configura en todos los usuarios de servidores remotos que se van a autenticar con la misma clave privada. Ya en el proceso de autenticación el cliente envía información sobre su clave pública al servidor. El servidor busca si para el usuario requerido se ha instalado pública del cliente. Si la encuentra, se envía un desafío al cliente que consiste en un valor aleatorio cifrado con la clave pública del cliente. El cliente, para completar la autenticación satisfactoriamente, debe descifrar el desafío con su clave privada y devolver el valor al servidor. El servidor comprobará si el desafío se ha resuelto correctamente y permitirá el inicio de sesión si así ha sido.

Bajo este esquema de autenticación, un par de claves (pública/privada) permite la autentificación en todos los servidores. Como el desafío es aleatorio, un ataque de fuerza bruta es mucho más difícil. Estas son las principales ventajas de este mecanismo de autentificación.

Para habilitar este mecanismo, se debe crear un par de claves. Las claves se guardan por defecto en el directorio home del usuario actual (concretamente en ~/.ssh/id_rsa y ~/.ssh/id_rsa.pub).

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):
Created directory '/home/user/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:5qCbnlPiUkqCVT2uCniJKo5vYxTQV9UKd+x8uV+k7EE user@debian10
The key's randomart image is:
+---[RSA 2048]----+
| .   o...o       |
|. . o + . +      |
| . o . + =   .   |
|  o   . . o o E .|
|oo o .. S  . + o |
|* = +..+    . + .|
|.* =.o  .    o o |
|+ B o+        o  |
|+=.+=.           |
+----[SHA256]-----+
```

La clave privada debe ser custodiada con gran cuidado para evitar su robo. La clave pública se instalará a modo de cerradura en usuarios determinados de servidores remotos usando el comando ssh-copy-id.

```
$ ssh-copy-id user@192.168.5.2
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/user/.ssh/id_rsa.pub"
The authenticity of host '192.168.5.2 (192.168.5.2)' can't be established.
ECDSA key fingerprint is SHA256:XH8IXNOQ3hpwPBxFzu3npW8Qu27ek6IXALzKe6MLe80.
Are you sure you want to continue connecting (yes/no)? yes
/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
user@192.168.5.2's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'user@192.168.5.2'"
and check to make sure that only the key(s) you wanted were added.
```

El proceso de instalación de una clave pública en un usuario/servidor remoto consiste básicamente en añadir al fichero ~/.ssh/authorized_keys la clave pública RSA (~/.ssh/id_rsa.pub). Se podría hacer con el comando *cat ~/.ssh/id_rsa.pub | ssh user@remote cat - > ~/.ssh/authorized_keys*. Cierto es que el comando anterior no comprueba si la clave pública ya se encuentra instalada. Para desinstalar una clave sólo hay que borrar la línea correspondiente en el fichero.

Si la clave que se desea instalar no es la que está en la carpeta por defecto (~/.ssh/id_rsa.pub), se puede usar el comando ssh-copy-id con la opción *-i <path/to/id_rsa>* donde <path/to/id_rsa> es el path a la clave privada. La clave pública necesariamente tiene que estar en el mismo directorio que la clave privada y estar en un fichero llamado igual que donde está almacenado la clave privada pero con extensión .pub.

Una vez hecho esto, usando la clave privada (~/.ssh/id_rsa) se podrá realizar la autenticación RSA de ssh. Simplemente se llama a SSH y, como la clave privada está en el sitio por defecto, la conexión es automática y no se solicita clave.

```
$ ssh user@192.168.5.2
Linux debian10 4.19.0-8-amd64 #1 SMP Debian 4.19.98-1 (2020-01-26) x86_64

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.
Last login: Thu Mar 12 23:29:58 2020 from 192.168.5.2
user@debian10:~$
```

Finalmente indicar que la opción -i de ssh permite especificar un fichero alternativo de clave privada (si es distinto de ~/.ssh/id_rsa).

9.7. Ejercicios

1) Un trabajador de un ayuntamiento que está usando Linux le ha surgido un problema con su sistema y te pide ayuda. Como idea, para resolver el problema se plantea el uso de una sesión SSH. Desgraciadamente, el trabajador no tiene IP pública. Tenéis a vuestra disposición un equipo so2.atopa.me donde hay un servidor OpenSSH y una cuenta *testuser* con clave *12test34*. Describe cómo podrías establecer una conexión SSH al equipo del trabajador del ayuntamiento.

2) Implementa con socat la opción -L de SSH sin usarla. Complétalo usando autenticación RSA.