SSH Tunneling

Pablo Garaizar Sagarminaga
txipinet@txipinet.com

SSH, una historia ajetreada...

Quizá ahora, cuando las miles de posibilidades que ofrece SSH han inundado nuestros sistemas Linux, olvidemos la controversia que ha rodeado a SSH desde sus orígenes.

"Apuesto a que un montón de gente no sabe que ssh 1.2.12 tiene una bonita licencia", con esta frase escrita en un foro de mensajes, el grupo OpenSSH se daba a conocer.

SSH 1.2.12 fue desarrollado por un programador finlandés, Tatu Ylönen, que publicó su trabajo bajo una licencia libre. Pronto el éxito de su programa lo llevo a patentar la marca registrada "SSH" y crear una empresa con fines comerciales. Las siguientes versiones dejaron de ser libres y se permitió su empleo únicamente para usos no comerciales. Los desarrolladores de OpenBSD se dieron cuenta de que su sistema operativo, centrado en la criptografía y seguridad, se quedaba "cojo" sin SSH. Además, las encuestas afirmaban que lo primero que la mayoría de usuarios de OpenBSD añadía después de instalar el sistema era SSH. Con estas ideas en la cabeza y bastante buenas intenciones, surgió el proyecto OpenSSH, cuyo principal objetivo consistió en desarrollar una implementación totalmente libre de SSH. Los primeros pasos de este proyecto estuvieron centrados en utilizar solamente código libre y portable. Tanto es así que pusieron especial empeño en evitar problemas con patentes y restricciones gubernamentales: la mayoría de los desarrolladores residían fuera de Estados Unidos, a excepción de Niels Provos, un alemán afincado en Michigan, que cruzó la frontera a Canadá para enviar su código desde una pequeña tienda local de informática en Ontario (nada quería dejarse a la ligera).

Todos estos esfuerzos no fueron vistos con tan buenos ojos por el desarrollador inicial de SSH, que veía como su proyecto comercial se iba al traste. Después de varias demandas judiciales y muchas discusiones, OpenSSH pudo mantener el "SSH" en su nombre y seguir con el trabajo que estaban realizando.

Figura 1. OpenSSH, una implementación libre de SSH.

¿Qué es un túnel SSH?

Esta bonita historia no debe hacernos olvidar que OpenSSH (SSH en lo sucesivo, para acortar) es una herramienta indispensable en la administración de sistemas. La mayoría de los protocolos que empleamos en nuestras comunicaciones están basados en diseños de hace casi 30 años, cuando la seguridad en redes telemáticas no era un problema. Telnet, FTP, POP3, protocolos de uso cotidiano, descuidan la seguridad y confidencialidad de los datos que envían. De nada sirve proteger nuestros servidores, implantar una buena política de contraseñas y actualizar las versiones de nuestros demonios, si luego cuando un usuario de POP3, por ejemplo, quiere ver su correo electrónico desde la universidad, envía su usuario y contraseña en texto plano por la red. Para evitar esto tenemos dos alternativas: bien elegir protocolos que sean seguros, bien hacer seguros los protocolos inseguros. De las dos opciones, la segunda permite reutilizar muchos más clientes y servidores ya programados, ayuda a enmascarar la complejidad de la seguridad en la transmisión y es una solución siempre escalable. Por todo esto, vamos a ver cómo convertir nuestros protocolos tradicionales en protocolos seguros, y qué tiene que ver SSH en todo ello.

La idea en la que se basa este procedimiento es la de hacer un túnel por el cual viajarán los datos de manera segura ("tunneling"). El símil con la vida real está bastante bien escogido: imaginemos que la tasa de mortalidad de los diferentes medios de transporte fuese equivalente a la posibilidad de que la seguridad de nuestros datos sea violada, y que el tren representase un canal seguro y el automóvil un canal inseguro. El túnel del Canal de la Mancha sería el ejemplo perfecto para ilustrar el concepto de "tunneling": cuando

queremos viajar a través de él con nuestro coche, tenemos que subirnos a un vagón para coches y luego cruzar el túnel en tren. Hemos incrementado sensiblemente la seguridad del viaje, porque la probabilidad de un accidente de tren es muchísimo menor que la de un accidente de coche. Este mismo proceso es el que se da a la hora de establecer un túnel seguro en la comunicación entre cliente y servidor. En cada uno de los extremos del túnel están las aplicaciones estándar (un demonio POP3 estándar, nuestro cliente de correo favorito...) y la comunicación se asegura haciendo uso de toda la potencia criptográfica de SSH. Para ello tenemos que realizar un procedimiento similar al que supondría subir nuestro coche al tren, establecer mediante SSH un reenvío de los datos gracias a una técnica denominada "port-forwarding". SSH recoge los datos que el cliente quiere enviar y los reenvía por el túnel o canal seguro, al otro lado del túnel se recogen los datos y se reenvían al servidor conveniente:

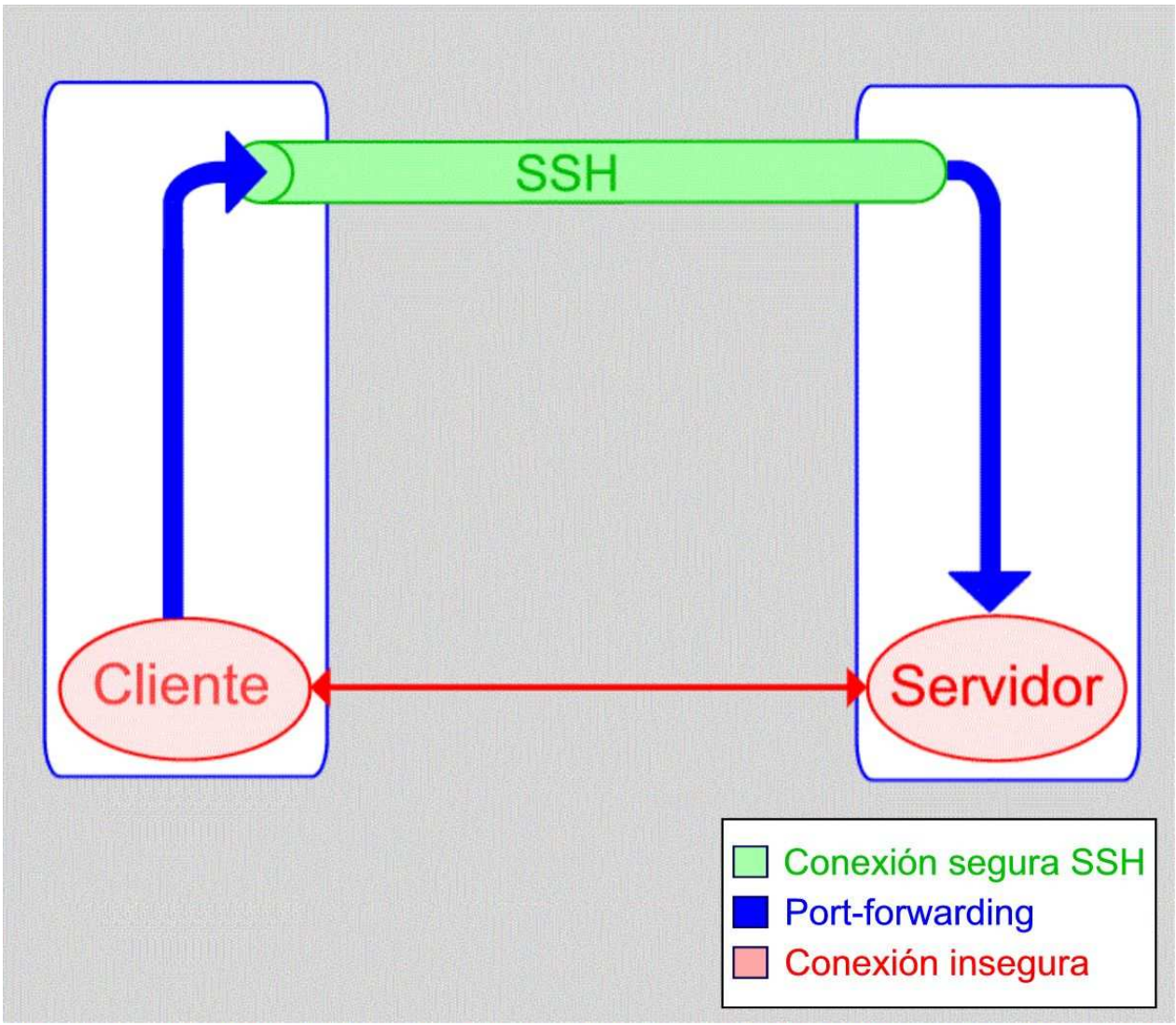

Figura 2. Proceso de tunneling SSH.

De la teoría a la práctica

Veamos cómo llevar esto a cabo. La clave está en usar la opción "-L" de SSH, que se encarga de todo el proceso de "port-forwarding". El siguiente ejemplo muestra cómo utilizar el puerto 10110 de nuestra máquina para establecer una comunicación segura con el puerto 110 del servidor popmail.correo.net:

```
$ ssh -P -L 10110:popmail.correo.net:110
```

El símbolo de dólar obviamente no pertenece al comando, indica que lo ha realizado un usuario sin privilegios de root. Esto es un punto importante: si queremos utilizar puertos privilegiados por debajo de 1024 ("well-known ports"), deberemos tener privilegios de root (por lo menos en un sistema con una configuración normal, alejado del "enfoque" de seguridad de Windows98 o similares). Si no disponemos de dichos privilegios, tendremos que optar por la alternativa de utilizar un puerto superior a 1024, como en

el ejemplo. Una manera fácil de no liarse con esto es sumar 10000 al número de puerto estándar, así el 110 sería el 10110 en nuestra máquina, el 10080 el 80, o el 16667 el 6667.

SSH da mucho más juego, éstas son algunas de las opciones que podremos incluir en nuestras configuraciones y scripts:

-1 ó -2 : fuerza a utilizar la versión 1 o la versión 2 de SSH. -f : pasa a segundo plano la ejecución de ssh (fork). Esta opción implica además la opción -n, en la que se impide leer de la entrada estándar. -N : impide ejecutar comandos remotos. Muy útil cuando lo único que se desea es "port-forwarding". -P : permite usar un puerto no privilegiado (>1024) para conexiones salientes. Muy usado cuando estamos tras un router o firewall que no permite conexiones a puertos privilegiados.

Para más información acerca de estas y otras opciones, ya sabéis, "man ssh" ;-)

Otros aspectos que deberemos tener en cuenta:

- Compatibilidad en cuanto a versiones de SSH. Bastantes clientes de SSH solamente soportan la versión 1, sin embargo esta versión es vulnerable a diversos ataques, por lo que es aconsejable utilizar la versión 2 convenientemente parcheada (openSSH-3.x.x con los parches para corregir el bug "hang-on-exit" o el "exit-delay"). Aquí se plantea la típica disyuntiva entre seguridad o compatibilidad.

- Para establecer una sesión SSH es necesario autenticación. Si utilizamos autenticación por passphrase (además de las correspondientes claves RSA o métodos similares), deberemos prever este hecho en nuestros scripts o nuestras sesiones interactivas o de lo contrario no podremos establecer el túnel SSH.

- Las reglas de protección en ipchains o iptables. Si tenemos configuradas reglas para bloquear las conexiones entrantes, necesitaremos establecer una excepción a esas reglas para que funcione el "port-forwarding" en local, con algo parecido a esto:

```
# iptables -A INPUT -s 127.0.0.1 -d 127.0.0.1 -i lo -j ACCEPT
# iptables -A OUTPUT -s 127.0.0.1 -d 127.0.0.1 -o lo -j ACCEPT
```

- Si vamos a utilizar puertos privilegiados en nuestra máquina local (con privilegios de root), deberemos tener en cuenta que esos puertos no se hayan empleado para conexiones entrantes. Si, por ejemplo, queremos hacer "port-forwarding" del puerto 110 para que nuestros usuarios puedan acceder de manera segura a su servidor de correo, deberemos tener cuidado ante un posible servidor POP3 escuchando el puerto 110 en nuestra máquina. Revisad vuestras configuraciones de los demonios que están escuchando puertos y en especial el fichero "/etc/inetd.conf".

- Si queremos establecer un túnel para cada conexión y destruirlo cuando ésta acabe, podemos usar una nueva opción incluida en el último parche (-S) para fijar un tiempo de espera o "delay" en el que se atenderán las conexiones SSH. Cuando ese tiempo expire, no se atenderán más conexiones y el comando finalizará cuando todas la conexiones hayan concluido (esto es importante, no termina cuando se consuma el tiempo fijado). Si disponemos de una versión antigua de SSH, podemos hacer esto mismo de la siguiente manera:

```
$ ssh -f -L 110:popmail.correo.net:110 usuario@popmail.correo.net sleep
30
```

esto sería equivalente a:

```
$ ssh -f -S 30 -L 110:popmail.correo.net:110 usuario@popmail.correo.net
```

Así estamos fijando un temporizador de 30 segundos para aceptar conexiones, finalizado el cual se esperará únicamente a que las conexiones ya establecidas terminen.

Infinitas posibilidades

Una vez entendido el método general para crear un túnel SSH, vamos a ver la multitud de aplicaciones posibles que tiene esta técnica:

Servicio	Puerto	Comentarios
FTP	21	Se han escrito ya varios textos acerca del SSH tunneling en servidores FTP ampliamente usados en nuestras máquinas Linux como puedan ser ProFTPd o wu-ftpd, si estás interesado en un demonio en particular, te recomendaría que buscaras información sobre él en particular. A nivel general, el SSH tunneling en FTP tiene algunos aspectos reseñables: ● Como bien sabemos, FTP tiene dos conexiones distintas: una para los comandos (control) y otra para la transmisión de ficheros (datos). La manera estándar de hacer un túnel SSH en una conexión FTP se centra únicamente en la protección de la conexión de control, es decir, del login/password de la sesión y de los comandos utilizados. Los ficheros se ç intercambiarán sin utilizar el túnel SSH. ● Es necesario usar el modo pasivo ("pasive mode") para la conexión de datos o de lo contrario el servidor FTP tratará de conectar con la máquina remota que se encuentre al otro lado del túnel SSH (típicamente la propia máquina servidora) y no con la máquina cliente que realiza la petición. ● Al usar el modo pasivo, la conexión de datos provendrá de una dirección IP distinta a la usada en la conexión de datos. Esto suele ser un problema para la gran mayoría de servidores FTP, ya que por defecto deniegan este hecho como medida de seguridad (para prevenir "hijacking" o "spoofing" en las conexiones). Deberemos, por lo tanto, configurar el servidor FTP para permitir esto. Debido a estos problemas de seguridad, y gracias a las nuevas prestaciones de la versión 2 de SSH, el SSH tunneling en conexiones FTP está siendo reemplazado por conexiones "sftp", que proporcionan protección tanto de datos como de comandos, sin necesidad de modificar el servidor FTP existente (realmente este último no es necesario, todo se realiza desde el servidor SSH).

Telnet	23	Realizar conexiones Telnet o rsh mediante túneles SSH no tiene ningún sentido, ya que SSH es ya el protocolo para shell remota de forma segura. En escenarios realmente extraños podría emplearse eventualmente para acceder, por ejemplo, desde un host de Internet a un servidor de una extranet que sólo tiene el puerto 23 abierto. En este caso podría establecerse una conexión SSH segura a un host en la frontera de la extranet con Internet, y de ahí realizar la comunicación al puerto 23 de forma insegura (suponiendo que el interior de la extranet no sea conflictivo).
SMTP	25	Ahora que a consecuencia del spam casi todos los servidores SMTP tienen unas políticas de "relay" muy severas, el SSH tunneling en conexiones SMTP puede aumentar la comodidad de muchos usuarios. La práctica totalidad de servidores SMTP tienen a ellos mismos (localhost) como "relay" aceptado, por lo que si establecemos un conexión segura mediante un túnel SSH a ese servidor y enviamos nuestro correo saliente por el túnel, el servidor lo considerará como proviniente de localhost a pesar de que nos encontremos en un host lejano, por lo que no denegará el envío.
HTTP	80	Muchos servidores web tienen secciones privadas que están protegidas por una contraseña (en Apache tenemos los típicos ficheros .htaccess que protegen directorios del servidor web, por ejemplo). La debilidad de este tipo de protección es que la contraseña viaja por la red en texto plano si no nos aseguramos de usar medios de encriptación como pueda ser una conexión median SSL al puerto 443 del servidor. Las advertencias de los navegadores no son en vano, y no deberían tomarse a la ligera dentro de una LAN en la que pueda haber sniffers de red. Un túnel SSH protegería de forma sencilla que esas contraseñas puedan ser espiadas.
POP3	110	Este protocolo (así como su antecesor POP2 en el puerto 109) es el candidato ideal a ser utilizado mediante un túnel SSH. Casi sin darnos cuenta, nuestros clientes de correo envían constantemente nuestro login y password al servidor de correo para sondear si hemos recibido nuevos mensajes. Todo este tráfico viaja en texto plano, por lo que puede ser capturado, analizado y la confidencialidad de nuestro correo podría verse afectada. Además, es posible que un eventual atacante que se hiciese con un usuario y clave de POP3, consiguiese entrar en servidor con ellos por otro protocolo. Además de esto, POP3 puede ser totalmente reemplazado por scripts que usan de forma nativa SSH, como scpmail, appendmail o loopmail.
NNTP	119	De la misma manera que sucede con POP3, nuestros clientes de noticias también se autentican automáticamente, si bien es cierto que la mayoría de servidores de news permiten acceso anónimo.

SMB	139	Las transferencias de ficheros mediante este protocolo típico de plataformas Microsoft también puede ser un objetivo de los túneles SSH. Existen varios problemas al respecto: ● La navegación dentro de directorios compartidos se realiza por UDP, por lo que no podremos realizarla a través del túnel SSH. ● NETBIOS siempre utiliza el puerto 139 y no puede ser reemplazado por otro, por lo que necesitaremos privilegios de root para utilizar ese puerto (<1024). ● Si queremos utilizar el puerto 139 para un túnel SSH, no podremos compartir ficheros de nuestra máquina al mismo tiempo. Deberemos elegir entre la conexión SSH y la conexión normal. Por todo ello, quizá convenga utilizar sftp y olvidarse de Samba para intercambios de ficheros.
IMAP	143, 220	IMAP admite el uso de túneles SSH de forma similar a POP3. Sin embargo, algunas de las posibilidades extra que ofrece IMAP con respecto a POP3, como pueda ser la organización de los mensajes en carpetas, pueden verse afectadas si no se utilizan los puertos originales para este protocolo (143 para IMAP2 o IMAP4 y 220 para IMAP3). Por ello, quizá necesitemos tener privilegios de root para hacer el port-forwarding para IMAP.
MySQL	3306	Nuestras conexiones remotas a Bases de Datos pueden también servirse de esta técnica. He elegido MySQL por ser bastante popular entre los programadores de entornos Linux, pero los túneles SSH pueden utilizarse para conectar con los múltiples tipos diferentes de Bases de Datos accesibles desde nuestros sistemas.
RPC (NIS, NFS...)	111	Holger Trapp programó un paquete denominado "sec_rpc" que utilizaba SSH para aumentar la seguridad de los protocolos basados en RPC (Remote Procedure Calls). Actualmente se ha mejorado bastante ese paquete y permite incluso túneles UDP con SSH versión 2. Este tipo de túneles son los que se usan para SNFS (Secure NFS), o NIS, protocolos históricamente inseguros, que se benefician de estos avances y consiguen perder su obsolescencia.
X11	6000	En ocasiones nos preocupamos por la seguridad de nuestras conexiones mediante shells remotas, pero descuidamos la protección de los datos enviados por terminales gráficas. Si usamos SSH tunneling en conexiones X11 evitaremos desagradables sorpresas. Para realizar el tunneling no es necesario fijar la variable de entorno "DISPLAY" de forma manual, el cliente ssh fija su valor automáticamente facilitándonos el proceso.

Tabla 1. Posibles aplicaciones de los túneles SSH.

Además de todas estas posibles aplicaciones, un protocolo nuevo empleado en redes inalámbricas se está beneficiando también de los túneles SSH. WEP es un protocolo de protección de datos en redes inalámbricas con un objetivo bastante modesto, proporcionar una protección similar a la de una conexión por cable (Wired Equivalent Protection). Recientemente, estudiantes de Berkeley han conseguido reventar el sistema criptográfico en el que se basa WEP y con el tiempo suficiente podría quebrantarse cualquier protección basada en WEP. Es por ello que los usuarios de redes de este tipo (wireless) está apostando por el fuerte cifrado que ofrece SSH para reforzar sus conexiones. Esto, en suma, deja una puerta abierta a la escalabilidad en este tipo de protocolos.

Antes de terminar, me gustaría comentaros dos curiosidades relacionadas con este tema. La primera es que gran parte de lo que hemos comentado en este artículo se puede utilizar en plataformas Microsoft gracias al empleo de la librería Cygwin, así que no tendremos que establecer configuraciones rocambolescas para utilizar SSH Tunneling en las máquinas con Sistemas Microsoft que podamos tener en nuestras redes. La segunda es que existen empresas que comercializan este servicio para todo tipo de protocolos. Lo que plantean es utilizar un conjunto de servidores que atienden conexiones ssh y las canalizan hacia los destinos reales. Realmente funcionan de intermediarios entre sus clientes y los servidores a los que quieren acceder sus clientes, proporcionando un canal seguro desde ellos a sus clientes, y "anonimizando" sus peticiones (todas las peticiones aparecerán como realizadas por los servidores de la empresa que proporciona este servicio y no por el cliente real).

En conclusión, el SSH Tunneling es una alternativa sencilla al empleo de protocolos de forma insegura, sin tener que hacer modificaciones en el propio protocolo. Además, esta técnica es aplicable a un amplio espectro de escenarios (como hemos podido comprobar) y es altamente escalable. Por todo ello recomiendo su empleo siempre que sea posible y aprovecho esta última línea para agradecer a los desarrolladores de OpenSSH todo el esfuerzo realizado ;-)

Este documento ha sido escrito por un miembro de e-GHOST, y su contenido es libre de ser reproducido en otros medios bajo las condiciones de la Licencia FDL (GNU Free Documentation License).