

Secure Shell

David J. Horat
davidjesus@gmail.com
www.davidhorat.com

Enrique Fernández
efdezperdomo@yahoo.es

Índice

I. Introducción	2
II. Algoritmos de cifrado	3
RSA.....	4
<i>Generación de claves</i>	4
<i>Encriptación</i>	5
<i>Desencriptación</i>	5
<i>Ataques posibles</i>	5
III. Funcionamiento	7
Características	7
Inicio de una comunicación segura.....	7
Métodos de autenticación de usuarios.....	9
<i>Autenticación con contraseña</i>	9
<i>Autenticación con clave pública</i>	9
IV. Implementaciones	11
Software Abierto y gratuito	11
<i>Multiplataforma</i>	11
<i>Windows</i>	11
<i>Macintosh</i>	12
<i>Unix y derivados</i>	12
<i>Móviles</i>	12
Software comercial	12
<i>Windows</i>	12
<i>Móviles</i>	13
V. Uso de las herramientas cliente de OpenSSH	14
Sesión entre máquinas en UNIX.....	14
<i>Ejemplos</i>	15
Transferencia de archivos	15
<i>Ejemplos</i>	16
Accesos de Mecanismos de Confianza	17
“Forwarding” o “tunneling”.....	18
<i>Ejemplo de forwarding desde máquina local</i>	18
<i>Ejemplo de forwarding desde máquina remota</i>	19
VI. Preguntas más frecuentes	21
VII. Referencias	23
Libros	23
Webs	23

I. Introducción

Secure Shell, en adelante SSH, es el nombre de un protocolo y del programa que lo implementa. Este protocolo sirve para acceder a máquinas remotas a través de una red, de forma similar a como se hace con telnet. La diferencia principal es que SSH usa técnicas de cifrado que hacen que la información que viaja por el medio de comunicación vaya de manera no legible y ninguna tercera persona pueda descubrir el usuario y contraseña de la conexión ni lo que se escribe durante toda la sesión; aunque es posible atacar este tipo de sistemas por medio de ataques de REPLAY y manipular así la información entre destinos. Al igual que telnet, sólo permite conexiones tipo terminal de texto, aunque puede redirigir el tráfico de X para poder ejecutar programas gráficos si tenemos un Servidor X arrancado.

Además de la conexión a otras máquinas, SSH nos permite copiar datos de forma segura (tanto ficheros sueltos como simular sesiones FTP cifradas), gestionar claves RSA para no escribir claves al conectar a las máquinas y pasar los datos de cualquier otra aplicación por un canal seguro de SSH (esto sólo si tenemos acceso como administrador a ambas máquinas).

La primera versión del protocolo y el programa eran libres y los creó un sueco llamado Tatu Ylönen, pero su licencia fue cambiando y terminó apareciendo la compañía *SSH Communications Security*, que lo ofrecía gratuitamente para uso doméstico y académico, pero exigía el pago a otras empresas. En el año 1996, dos años después de que se creara la primera versión, se propuso una segunda versión, incompatible con la anterior, como borrador en la IETF¹ (*Internet Engineering Task Force*). En el año 2005 sigue como borrador con el nombre de *Secure Shell (secsh)*², lo que conocemos con el nombre de SSH-2.

Para entender la importancia del protocolo SSH, primero haremos un breve repaso de los algoritmos de cifrado actuales, especialmente en el RSA y luego entraremos en su funcionamiento.

¹ IETF (Internet Engineering Task Force): <http://www.ietf.org/home.html>

² Secure Shell Group: <http://www.ietf.org/html.charters/secsh-charter.html>

II. Algoritmos de cifrado

El fruto de las funciones criptográficas inventadas en las últimas tres décadas se observa a diario en el desarrollo de las firmas digitales, los certificados digitales, los sistemas de autenticación y el correo electrónico seguro. En la actualidad se emplean dos tipos de criptografía:

La criptografía simétrica, en la cual se usa la misma contraseña o llave para encriptar y para desencriptar la información. Entre los sistemas de criptografía simétrica, podemos mencionar Blowfish, IDEA (International Data Encryption Algorithm), FEAL (Fast Data Encipherment Algorithm), DES (Data Encryption Standard) y los más comunes que son el 3-DES, y el Rijndael o AES, adoptado en el año 2000.

El usar la misma llave para encriptar y para desencriptar es un problema a la hora de enviar datos, ya que el remitente debe enviar previamente la llave al destinatario para que éste pueda desencriptar la información, y debe hacerlo por un canal seguro. Por lo tanto la criptografía simétrica se emplea especialmente para almacenamiento seguro de datos (solamente una persona necesita la llave). Para envío de datos es preferible la criptografía asimétrica.

La criptografía asimétrica, que emplea un esquema de llave pública y llave privada. La información se encripta con la llave pública, y se desencripta con la llave privada. No presenta el problema de transmisión de la llave que tiene la criptografía simétrica, ya que la llave pública no sirve para desencriptar la información.

Los sistemas de criptografía asimétrica incluyen DH (Diffie & Hellman), ElGamal, DSA (Digital Signature Algorithm), Merkle-Hellman, Chor-Rivest, LUC, McEliece y RSA (Rivest, Shamir & Adleman) que es el más ampliamente usado y objeto de estudio de este trabajo.

Tanto la criptografía simétrica como la asimétrica basan su fortaleza en problemas matemáticos difíciles de resolver, como por ejemplo la factorización de números enteros grandes. Sin embargo y debido al avance en la potencia de computación, se estima hoy

en día que solamente ofrecen buena seguridad las llaves de 2048 bits en el caso de RSA y 256 bits para AES.

RSA

Este sistema de clave pública fué diseñado en 1977 por los profesores del MIT (Massachusetts Institute of Technology) Ronald R. Rivest, Adi Shamir y Leonard M. Adleman, de ahí las siglas con las que es conocido. Desde entonces, este algoritmo de cifrado se ha convertido en el prototipo de los de clave pública.

La seguridad de RSA radica en la dificultad de la factorización de números grandes: es fácil saber si un número es primo, pero es extremadamente difícil obtener la factorización en números primos de un entero elevado, debido no a la dificultad de los algoritmos existentes, sino al consumo de recursos físicos (memoria, necesidades hardware...incluso tiempo de ejecución) de tales algoritmos. Se ha demostrado que si n es el número de dígitos binarios de la entrada de cualquier algoritmo de factorización, el coste del algoritmo es $O(n^2)$, con un tiempo de ejecución perteneciente a la categoría de los llamados problemas intratables.

Generación de claves

Supongamos que Alicia desea permitir que Benito le mande un mensaje cifrado sobre un canal inseguro. Lo primero que ha de hacer es generar los pares de clave pública (n,e) y privada (n,d) .

1. Elegimos dos números muy grandes p y q (del orden de 300 cifras) que sean diferentes y totalmente independientes el uno del otro. Calculamos $n = p * q$.
2. Calculamos la función de totient de n : $totient(n) = (p-1)*(q-1)$
3. Elegimos un entero e , tal que $1 < e < totient(n)$ y que además sea coprimo con $totient(n)$.
4. Calculamos un d , tal que $d * e = 1 \text{ mod } totient(n)$.
5. La clave pública es (n,e) y la privada (n,d) .

Alicia le transmite su clave pública a Benito y conserva la clave privada en secreto. Los valores p y q son muy sensibles, ya que son la descomposición en factores primos de n y los que dieron lugar a e y d . Generalmente destruidos, aunque pueden conservarse en secreto, junto con la clave privada, para acelerar el proceso de descryptación usando el Teorema del Resto Chino³.

Encriptación

Ahora supongamos que Benito desea mandarle un mensaje a Alicia. Lo único que tendrá que hacer es consultar la clave pública de Alicia, dividir el mensaje que quiere enviarle, asignarle un alfabeto numérico a cada trozo y calcular para cada división: $c = n^e \bmod n$.

Descryptación

Con el mensaje que le ha llegado a Alicia, lo que tiene que hacer es dividirlo y usar su clave privada para calcular: $n = c^d \bmod n$.

Ataques posibles

Ataque a módulo común: Una posible implementación de RSA consiste en asignar el mismo módulo n a distintos usuarios, pero distintos valores para los exponentes e y d . Esto tiene el fallo de que si el mismo mensaje se cifra con distintos exponentes y el mismo módulo y ambos exponentes son primos entre sí (y generalmente lo serán), el texto en claro puede recuperarse sin ninguno de los exponentes privados⁴. Sea P el texto en claro. Sean e_1 y e_2 los exponentes públicos (claves de cifrado) y n el módulo. Los textos cifrados son: $C_1 = P^{e_1} \bmod n$ y $C_2 = P^{e_2} \bmod n$. El criptoanalista tiene acceso a C_1 , C_2 , e_1 , e_2 y n . Así es como recupera P : Al ser e_1 y e_2 primos entre si, existen enteros r y s tales que $r \cdot e_1 + s \cdot e_2 = 1$. Supongamos sin pérdida de generalidad que r es negativo (alguno de los dos ha de serlo) entonces puede calcularse el inverso de

C_1 y se tendrá que: $(C_1^{-r})^{e_2} \cdot C_2^s = P \bmod n$

3 Teorema del Resto Chino: http://en.wikipedia.org/wiki/Chinese_Remainder_Theorem

4 G.J. Simmons, "A 'Weak' Privacy Protocol Using the RSA Cryptosystem.", Cryptologia, v.7, n.2, Abr 1983

Ataque basado en un exponente público "bajo": Aunque un exponente bajo acelera el cifrado, también lo hace más inseguro. Si se encriptan $e(e+1)/2$ mensajes linealmente dependientes con diferentes claves públicas y el mismo valor de e hay un ataque contra el sistema⁵. Si los mensajes son idénticos entonces es suficiente con e mensajes. La solución más sencilla a este problema es añadir a los mensajes valores aleatorios independientes, PGP por ejemplo hace esto de modo que un mismo mensaje no se cifrará dos veces de la misma manera.

Ataque basado en un exponente privado "bajo": Otro ataque⁶, permite recuperar d cuando éste no supera un cuarto de n y e es menor que n . Esto ocurre raramente si e se elige al azar. El fundamento matemático del ataque es la representación de los números racionales como fracciones continuas finitas.

Ataque por iteración: Conocidos n , e y C un enemigo puede producir una sucesión de mensajes $C_1=Ce \bmod n$, $C_2=C_1e \bmod n$... $C_k=C_{k-1}e \bmod n$. Si existe un C_j tal que $C=C_j$ se deduce que $M=C_{j-1}$ ya que $C_{j-1}e=C_j=C$. Este ataque se vuelve impracticable si $p-1$ y $q-1$ contienen factores primos grandes.

5 J. Hastad, "On using RSA with Low Exponents in a Public Key Network", Advances in Cryptology, CRYPTO '85 Proceedings, Springer Verlag 1986

6 M.J. Wiener, "Cryptanalysis of Short RSA Secret Exponents", IEEE Transactions on Information Theory, v.36, n.3, May 1990

III. Funcionamiento

SSH es una aplicación cliente-servidor que utiliza el puerto 22 del protocolo TCP. La implementación más ampliamente usada de SSH es OpenSSH⁷, desarrollado por el equipo de OpenBSD⁸ a principios del año 1999. Este es el principal programa que usaremos para todas las pruebas. Más adelante tendremos una lista de clientes y servidores SSH libres, gratuitos y comerciales.

Características

Las principales características y garantías del protocolo SSH son:

- **Privacidad** de tus datos, mediante encriptación fuerte.
- **Integridad** de las comunicaciones, garantizando que no han sido alteradas.
- **Autenticación** mútua entre cliente y servidor.
- **Autorización**, es decir, control de acceso a las cuentas de usuario.
- **Forwarding** o **tunneling**, es decir, encriptación de otras sesiones basadas en TCP/IP.

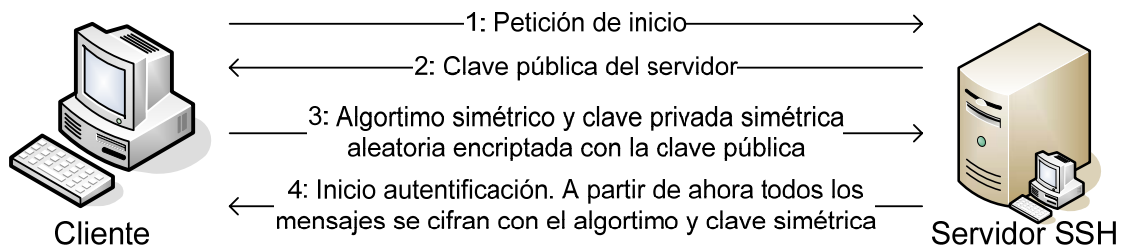
Inicio de una comunicación segura

Una de las principales desventajas del cifrado asimétrico es su lentitud, por lo que para mensajes de cierto tamaño se requiere bastante CPU. Por ello, para afrontar los problemas clásicos del paso de mensajes encriptados, SSH usa una aproximación combinada de ambos tipos de cifrado. Por un lado usa RSA como cifrado asimétrico, y por otro puede usar 3DES, Blowfish, IDEA o AES como cifrado simétrico entre otros.

⁷ OpenSSH: <http://www.openssh.org>

⁸ OpenBSD: <http://www.openbsd.org>

El proceso para iniciar una comunicación encriptada entre el cliente y el servidor es el siguiente:



1. El cliente abre una conexión TCP al puerto 22 del servidor.
2. El servidor comprueba que el cliente tiene permisos para iniciar un intento de conexión. El cliente y el servidor acuerdan la versión del protocolo a utilizar, de acuerdo a su configuración y capacidades.
3. El servidor posee un par de claves (pública y privada) del algoritmo RSA. El servidor envía su clave pública al cliente.
4. El cliente compara la clave pública recibida del servidor con la que tiene almacenada de accesos previos. En caso de ser un nuevo servidor o de que la clave pública haya cambiado, se le notificará al usuario para que la acepte. En caso de que sea distinta, existe un riesgo de que alguien, en el camino entre el servidor y el cliente, haya cambiado la clave pública por otra suya para interceptar todas las comunicaciones entre ambos. Este ataque se conoce con el nombre de “*man in the middle*”.
5. El cliente selecciona un algoritmo de cifrado simétrico (Blowfish, 3DES o AES entre otros) y genera una clave aleatoria, llamada clave compartida de sesión, para dicho algoritmo.
6. El cliente envía un mensaje con su selección de algoritmo de cifrado simétrico y la clave creada, todo ello encriptado con la clave pública del servidor usando el algoritmo RSA.

7. En adelante, toda comunicación cliente-servidor y viceversa se realizará usando el algoritmo simétrico y la clave compartida de sesión.
8. A partir de aquí se realiza la autenticación del usuario. Se pueden usar diferentes mecanismos que estudiaremos más adelante.
9. Finalmente se inicia la sesión.

Métodos de autenticación de usuarios

Existen varios métodos que pueden utilizarse para autenticar usuarios. Aunque son mutuamente excluyentes, tanto el cliente como el servidor pueden soportar varios de ellos. En el momento de la autenticación se aplicarán los métodos que ambos soporten, siguiendo un orden determinado. Analizaremos los dos métodos más importantes a los fines de este documento:

Autenticación con contraseña

Es el método más simple (y común) de autenticación. El cliente solicita al usuario el ingreso de una contraseña (password) y la misma es enviada al servidor, el cuál validará la misma utilizando los mecanismos configurados en el sistema (generalmente, utilizará la autenticación del sistema Unix para validar la contraseña contra el contenido del archivo */etc/shadow*, de la misma manera que lo hace el login).

Las desventajas de este sistema son varias:

- El usuario debe teclear su contraseña cada vez que se conecta al servidor.
- La contraseña del usuario es enviada hacia el servidor, por lo tanto hay que verificar que el servidor es el verdadero.

Autenticación con clave pública

Para poder llevarse a cabo, el usuario debe tener un par de claves pública/privada, y la clave pública debe estar almacenada en el servidor. Luego de establecida la

conexión, el servidor genera un número aleatorio llamado “desafío” (*challenge*) que es cifrado con la clave pública del usuario usando RSA o DSA, aunque en los ejemplos nos referiremos siempre a RSA. El texto cifrado es enviado al cliente, que debe descifrarlo con la clave privada correspondiente y devolverlo al servidor, demostrando de esta manera que el usuario es quien dice ser.

Las ventajas de este método respecto del anterior son:

- El usuario no debe teclear (ni recordar) ninguna contraseña.
- Ninguna contraseña secreta es enviada al servidor.

No debe confundirse el hecho de que el cliente y el servidor utilicen RSA para establecer la comunicación cifrada, con el hecho de que un usuario pueda utilizar el mismo algoritmo para autenticarse ante el servidor. Son dos etapas bien diferenciadas e independientes, aunque puede utilizarse el mismo algoritmo de cifrado en ambas.

IV. Implementaciones

Software Abierto y gratuito

Multiplataforma

- [PuTTY](#): Una suite de clientes para SSH, SFTP, SCP y telnet.
- [Ganymed SSH2](#): Una librería en Java de SSH-2.
- [JavaSSH](#): Un cliente de SSH hecho en Java.
- [JSch](#): Una librería hecha en Java para acceder a un servidor SSH como cliente de terminal, forwarding y transferencia de ficheros entre otros.
- [MindTerm](#): Una implementación en Java disponible gratuitamente para uso personal.
- [shtools](#): Incluye sstherm, un terminal ssh basado en Java accesible desde un navegador web.

Windows

- [WinSCP](#): Un cliente de SFTP y SCP abierto.
- [freeSSHD](#): Una servidor deSSH y Telnet con soporte para SFTP y *forwarding*.
- [OpenSSH para Windows](#).
- [SSHDOS](#).
- [Tunnelier](#): Cliente de SSH y SFTP gratuito para uso personal.

- [Whitehorn Secure Terminal](#): Cliente de SSH y Telnet gratuito.

Macintosh

- [MacSSH](#): Cliente terminal para Classic MacOS (PPC y 68k).

Unix y derivados

- [Lsh](#): Cliente y servidor SSH del proyecto GNU.
- [OpenSSH](#): Una implementación de SSH desarrollado por OpenBSD y ampliamente usado.
- [Dropbear](#): Cliente y servidor.
- [OSSH](#): Basado en la implementación original creador por Björn Grönvall.
- [libssh](#): Una librería de cliente y servidor.
- [libssh2](#): Otra librería de cliente y servidor.

Móviles

- [MidpSSH](#): Una implementación abierta para J2ME que puede ser usada en muchos móviles y dispositivos MIDP1 y MIDP2.

Software comercial

Windows

- [Private Shell](#): Soporte para SSH y SFTP.
- [PenguinNet](#): Cliente de SSH-1 y 2.

- [SFTPPlus](#): SFTP mejorado con funciones de automatización y auditoría. Multiplataforma.
- [Fortress](#): Cliente y servidor de SSH 1 y 2.
- [WinSSHD](#): Servidor SSH de Bitvise.
- [SSH Tectia Client](#): Cliente SSH de la empresa *SSH Communications Security*.
- [sshlib](#): Librería de SSH 2 en C++.
- [SecureCRT](#): Soporte para transferencia de ficheros ZMODEM y XMODEM.
- [ShellGuard](#): Cliente SSH 1 y 2. También soporta telnet y conexión directa.

Móviles

- [Pragma Handheld SSH-Telnet Client](#): Cliente SSH-2 para PocketPC y Windows CE.
- [Idokorro Mobile SSH](#): Cliente para RIM BlackBerry y dispositivos móviles.
- [Top Gun SSH](#): Cliente para Palm OS.

V. Uso de las herramientas cliente de OpenSSH

Sesión entre máquinas en UNIX

“ssh” permite mantener sesiones interactivas con una máquina remota de la forma como lo hace el comando telnet.

```
ssh [-a] [-c idea|blowfish|des|3des|arcfour|none] [-e esc] [-i identity_file] [-l  
login_name] [-n] [-k] [-V] [-o] [-p port] [-q] [-P] [-t] [-v] [-x] [-X] [-C] [-g]  
user@hostname:port [command]
```

Descripción de opciones relacionadas con el inicio de sesión:

- a Deshabilita el agente de autenticación.
- c *idea/des/3des/...* Selecciona el algoritmo de cifrado utilizado para cifrar la sesión.
- e Habilita el carácter de escape para una determinada sesión (default:~).
- i *identity-file* Selecciona el archivo donde leerán la llave de autenticación. Por default utiliza `~/.ssh/identity` dentro del home del usuario.
- l *login-name* Especifica la cuenta remota por medio de la cual se desea tener acceso.
- p *port* Puerto remoto de conexión.
- q Causa que todas las advertencias y mensajes de diagnóstico sean suprimidos, solo los mensajes de errores fatales son desplegados.
- v Causa que ssh despliegue mensajes de depuración acerca del proceso de conexión.
- x Deshabilita el reenvío (forwarding) de X11.
- C Compresión de todos los datos transmitidos durante la conexión.

Ejemplos

```
$ ssh -l micuenta maquina.remota
```

En este ejemplo utilizamos la opción `-l` para proporcionar el login con el que tendremos acceso a la máquina remota. En este caso la cuenta es `micuenta` y la máquina es `maquina.remota`.

```
$ ssh micuenta@maquina.remota
```

También podemos hacer uso del formato descrito arriba para entrar a una cuenta dentro de una máquina remota.

```
$ ssh maquina.remota
```

En caso de poseer el mismo nombre de cuenta en ambas máquinas (local y remota) es posible tener acceso a la máquina remota proporcionando solamente el nombre de la máquina.

Transferencia de archivos

Secure Shell proporciona una herramienta que permite realizar transferencia de archivos entre distintos hosts, haciendo uso de las características de `ssh`, este comando es `scp`, el cual cuenta con las siguientes opciones:

```
scp [-aAqrvBCL1] [-S path-to-ssh] [-o ssh-options] [-P port] [-c cipher] [-i identity]  
[[user@]host1:]filename1,... [[user@]host2:]filename2
```

Descripción de opciones:

- `-a` Habilita la estadística de transferencia de cada archivo que se transfiera.
- `-A` Deshabilita el mensaje de estadística de transferencia.
- `-c cipher` Selecciona el algoritmo de cifrado a utilizar en la transferencia de datos.
- `-i identity` Selecciona el archivo donde se tendrá acceso a la llave RSA.

-L Uso de puerto no privilegiado. Esta opción posee algunas restricciones como son la imposibilidad de usar rhosts o autenticación rsarhosts.

-l Forzar a scp a usar el comando scp1 por parte de host remoto, Esto puede ser necesario en el caso que el host remoto utilice scp2.

-o *ssh-options* Opciones que se pasarán al ssh.

-p Preserva las distintas fechas y horas de acceso, modificación y atributos del archivo original.

-q Deshabilita el despliegue de estadísticas de transferencia.

-r Copia recursiva de directorios completos.

-v Causa que scp y ssh desplieguen mensajes de estado relacionados con el proceso de conexión y transferencia de datos. Adecuado para depurar errores existentes en las conexiones realizadas.

Ejemplos

```
$ scp micuenta@máquina.remota:/tmp/archivo /copias
```

En este caso se copiará /tmp/archivo localizado en la máquina remota *maquina.remota* al directorio /copias en la máquina local. Se utilizó la cuenta micuenta para acceder al servidor.

```
$ scp /copias/archivo micuenta@máquina.remota:~/bck
```

En este caso se realizará la copia del archivo /copias/archivo localizado en la máquina local a la máquina remota *máquina.remota* colocando el archivo en el directorio bck del home de micuenta.

Accesos de mecanismos de confianza

Para realizar una conexión segura mediante mecanismos de confianza en SSH2 realice lo siguiente:

- 1.- Generar par de llaves dsa mediante el comando ssh-keygen.

```
Cliente> ssh-keygen -t rsa
Generating public/private dsa key pair.
Enter file in which to save the key (/home/usuario/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/usuario/.ssh/id_rsa.
Your public key has been saved in /home/usuario/.ssh/id_rsa.pub.
The key fingerprint is:
1a:84:34:9f:d1:97:76:03:c0:c7:be:12:b0:95:1e:37 usuario@Cliente
```

Se generan dos archivos en el directorio ~/.ssh/.

```
id_dsa.pub(publica)
id_dsa (privada)
```

2.- Exportar id_rsa.pub al servidor al que se realizará la conexión, e incluirlo su contenido en el archivo ~/.ssh/authorized_keys2.

```
Servidor>cat id_dsa.pub.Cliente >> ~/.ssh/authorized_keys2
```

Quedando algo así:

```
Servidor> cat ~/.ssh/authorized_keys2
ssh-rsa
mslHXihCHvYNADHKvFPSESKHUUpwF0TbpmvMesSGifqhQftn8BOU=usuario@Cli
ente
```

3.- Probar la conexión.

```
Cliente> ssh cuenta@Servidor
```

“Forwarding” o “tunneling”

Usaremos de nuevo la utilidad “ssh” que vimos antes, pero con otras opciones.

```
ssh [-f] [-N] [-L port:host:hostport] [-R port:host:hostport] hostname [command]
```

-f Hace que el proceso ssh vaya a segundo plano una vez autenticado.

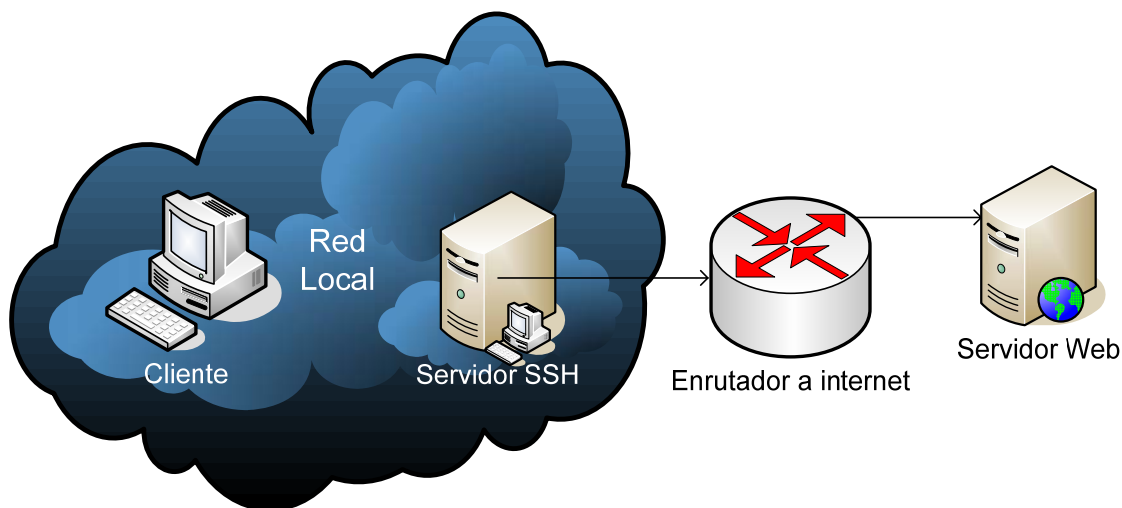
-N No ejecuta ningún comando en el servidor remoto, es decir, no permite usar la consola remota.

-L port:host:hostport Crea una conexión desde la máquina local donde se ejecuta el cliente en el puerto *port* al servidor *host*, puerto *hostport*.

-R port:host:hostport Crea una conexión desde la máquina remota donde se ejecuta el servidor en el puerto *port* al servidor *host*, puerto *hostport*.

Ejemplo de *forwarding* desde máquina local

Imaginemos una red local, en donde la única máquina que tiene acceso a internet es el servidor de SSH y nosotros queremos, desde una máquina normal de dicha red, conectarnos a un servidor web. Además sabemos que el servidor de SSH sólo tiene abierto el puerto 22, es decir, el del propio servicio SSH y no va a permitir abrir ninguno más para escucha.



En este caso, tenemos que conseguir que los paquetes vayan al servidor SSH y que éste los redirija al servidor web. Para ello usaremos el cliente SSH para crear una conexión en nuestra máquina en algún puerto, por ejemplo el 8080, y que redirija el tráfico al servidor SSH y luego que éste lo redirija al servidor web. Los paquetes se moverían de la siguiente manera:

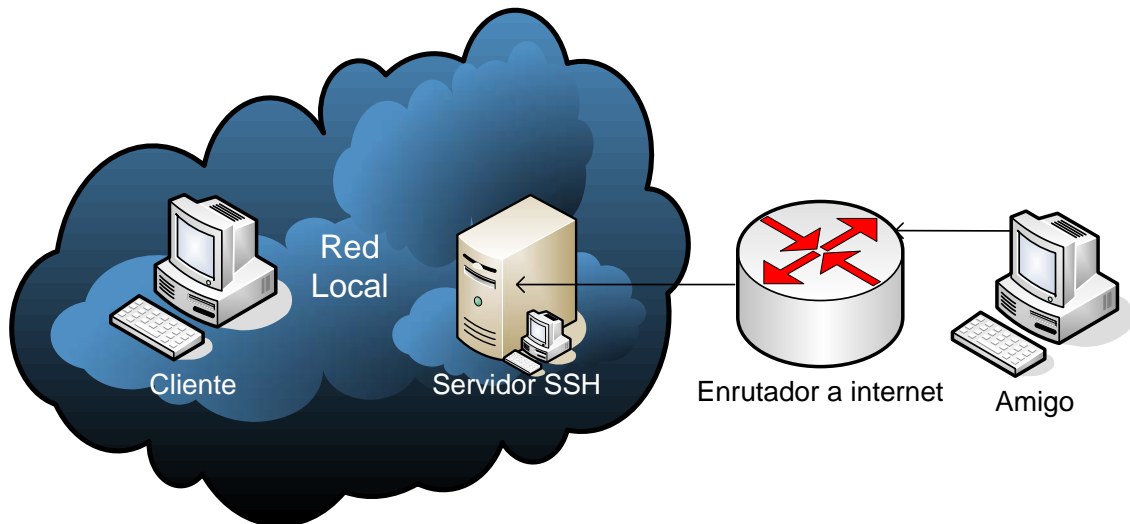
```
Navegador web -http-> http://localhost:8080  
localhost:8080 -ssh-> servidor_ssh:22  
servidor_ssh:algún_puerto_alto -http-> http://dondequeriamosir.com:80
```

Para conseguir que nuestro cliente ssh abra un puerto en nuestra máquina local, el 8080, y que redirija el tráfico de ese puerto al servidor ssh para que lo mande al destino y puerto que queramos (dondequeriamosir.com, 80), usaremos la línea:

```
ssh -N -f -L 8080:dondequeriamosir.com:80 usuario@servidor_ssh
```

Ejemplo de *forwarding* desde máquina remota

Imaginemos ahora que seguimos detrás de nuestra red local y no tenemos acceso a internet, o, como mínimo, nadie tiene acceso desde fuera a nuestro ordenador. Sin embargo hay un amigo que desea conectarse desde internet a nuestro ordenador, por ejemplo a nuestro FTP.



En este caso, lo que tendremos que hacer es que el servidor SSH abra un puerto en su máquina, por ejemplo el 8000, y que redirija todo el tráfico que le entre a dicho puerto hacia nuestro ordenador en el puerto 21, el del FTP. Para ello ejecutaremos:

```
ssh -N -f -R 8000:localhost:80 usuario@servidor_ssh
```

En este caso, localhost se refiere con respecto al servidor de SSH, ya que hemos usado la opción “-R”. En caso de haber usado “-L”, como anteriormente, localhost se hubiese referido con respecto a la máquina cliente.

VI. Preguntas más frecuentes

1. ¿Puedo hacer copias de seguridad sobre SSH?

Si. SSH permite la transferencia de archivos y reemplaza a rsh, por lo que los scripts para copias de seguridad se pueden reutilizar con un cambio mínimo.

2. ¿Puedo usar SSH para comunicarme a través de un cortafuegos?

Si. Puedes usar *forwarding* para eso.

3. ¿Puedo usar SSH para conectar dos subredes por internet?

Si. Puedes ejecutar PPP sobre una conexión SSH, tanto para conexiones [TCP](#) como para conexiones [UDP](#). No obstante, una solución para conectar dos subredes de forma segura es usar redes privadas virtuales⁹ o VPN.

4. ¿Puedo usar SSH para securizar servicios basados en UDP, como NFS o NIS?

[Existe una solución para servicios basados en RPC](#), como por ejemplo NIS.

5. ¿Puedo usar SSH para securizar servicios TCP, como FTP o POP?

Si, pero hay que tener en cuenta que la comunicación del servidor SSH al propio servidor del servicio que usemos no estará encriptada, por lo que deberá ser segura por algún otro método.

Otra posibilidad es que el propio servicio haya sido mejorado para implementarlo, como por ejemplo [esta implementación del servicio POP](#).

6. ¿Puedo usar SSH a través de un PROXY o SOCKS?

Si, ya que es una conexión estándar TCP.

⁹ Red Privada Virtual o VPN: http://es.wikipedia.org/wiki/Red_privada_virtual

7. ¿SSH 2 es compatible con SSH 1?

No. SSH 1 y SSH 2 son protocolos totalmente diferentes, tanto a la hora de encriptar los paquetes como de autenticar los sistemas. SSH2 es una reescritura completa del protocolo SSH1 con mejoras en seguridad, rendimiento y portabilidad.

8. ¿Dónde están los archivos de configuración del OpenSSH?

En muchas distribuciones GNU/Linux las podemos encontrar en el directorio “/etc/ssh”.

VII. Referencias

Libros

1. Computer networks / Andrew S. Tanenbaum. -- 4th. ed. -- Englewood Cliffs (New Jersey) : Prentice Hall, 2003.
2. Análisis e implementación del RSA. David J. Horat, Jorge L. Cañizales. Criptografía, U.L.P.G.C, 2004.
3. Introducción a Secure Shell. Javier Smaldone. – Versión 0.2 – <http://www.smaldone.com.ar>
4. SSH, The Secure Shell: The Definitive Guide. Daniel J. Barrett, Richard E. Silverman. Ediciones O`Reilly, Mayo 2005.

Webs

1. Enciclopedia Libre en varios idiomas, incluyendo inglés y español. <http://www.wikipedia.org>
2. Criptotaller del cifrado RSA en matematicas.net. <http://www.matematicas.net/paraiso/cripto.php?id=rsa1>
3. MundoCripto: RSA. <http://webs.ono.com/usr005/jsuarez/rsa1.html>
4. Preguntas más frecuentes de SSH en employees.org. <http://www.employees.org/~satch/ssh/faq/ssh-faq.html>
5. Real Programmers - Tunneling. <http://realprogrammers.com/>