El protocolo SSH

- Doctorado en Ciencias de la Computación – ESIDE -

Jorge García Ochoa de Aspuru
E-mail: shadow@bardok.net
URL: http://www.bardok.net

Índice de contenido

1. ¿Qué es SSH?....6
1.1 Algoritmos de cifrado....6
1.2 Autentificación....6
1.3 Protección proporcionada por SSH....7
1.4 Sistemas operativos....7
2. SSH1....9
2.1 El protocolo SSH1....9
2.2 Características del protocolo....9
2.3 Formato de los paquetes....10
2.4 Compresión de los paquetes....11
2.5 Cifrado de paquetes....11
2.6 Puerto TCP/IP y otras opciones....11
2.7 Identificación de la versión del protocolo....12
2.8 Intercambio de claves y autentificación del equipo servidor....12
2.9 El nombre de usuario....13
2.10 Fase de autentificación....13
Autentificación por RHOSTS....13
Autentificación por RHOSTS y RSA....13
Autentificación por RSA....14
Autentificación por password....14
2.11 Operaciones de preparación....14
2.12 Sesión interactiva e intercambio de datos....14
3. SSH2....15
3.1 Arquitectura....15
Claves de equipo....15
Extensibilidad....16
Negociación....16
Propiedades de seguridad....16
Consideraciones de seguridad....16
3.2 Protocolo de transporte....16
Establecimiento de la conexión....17
Utilización sobre TCP/IP....17
Intercambio de versión del protocolo....17
Formato de los paquetes....17
Compresión....18
Cifrado....18
Integridad de los datos....18
Métodos de intercambio de claves....18
Algoritmos de clave pública....19
Intercambio de claves....19
Negociación de algoritmos....19
Resultado del intercambio de claves....20
Uso de las claves....20
Reintercambio de claves....21
Petición de servicio....21
3.3 Protocolo de autentificación....21

Consideraciones del protocolo de autentificación....22
Peticiones de autentificación....22
Respuestas a peticiones de autentifiación....22
Método de autentificación "none"....22
Mensaje de aviso....23
Método de autentificación por clave pública....23
Método de autentificación por password....23
Autentificación basada en la máquina cliente....24
3.4 Protocolo de conexión....24
Peticiones globales....25
Mecanismo de canales....25
Apertura de un canal....25
Transferencia de datos....25
Cierre de un canal....26
Peticiones específicas del canal....26
Sesiones interactivas....27
4. Utilización de SSH....28
4.1 SSH1....28
ssh1....28
scp1....28
sshd1....29
ssh-agent1....29
ssh-add1....30
4.2 SSH2....30
ssh2....30
scp2....30
sshd2....30
ssh-agent2....31
ssh-add2....31
sftp2....31
4.3 Administración de SSH....31
4.4 Uso de autentificación con clave pública de usuario en SSH2....32
5. SSH y otras aplicaciones....34
5.1 Uso de SSH tras un firewall....34
5.2 Tunneling con SSH....34
6. Conclusiones....37
7. Referencias....38

Índice de tablas

Tabla 1.1. Versión de SSH soportada en servidores 8
Tabla 1.2. Versión de SSH soportada en clientes 8

Índice de ilustraciones

Ilustración 5.1. Tunneling con SSH 35

1. ¿Qué es SSH?

SSH, o "Secure Shell" es tanto una aplicación como un protocolo, que permite conectar dos ordenadores a través de una red, ejecutar comandos de manera remota y mover ficheros entre los mismos. Proporciona auntentificación fuerte y comunicaciones seguras sobre canales no seguros y pretende ser un reemplazo seguro para aplicaciones tradicionales no seguras, como telnet, rlogin, rsh y rcp. Su versión 2 (SSH2) proporciona también sftp, un reemplazo seguro para FTP.

Además de esto, SSH permite establecer conexiones seguras a servidores X-Windows, realizar conexiones TCP seguras y realizar acciones como sincronización de sistemas de ficheros (rsync) y copias de seguridad por red, de manera segura.

SSH permite, además, solventar problemas de seguridad que pueden derivarse del hecho de que usuarios tengan acceso como root a ordenadores de la red, o acceso al cable de comunicaciones, de modo que podrían interceptar contraseñas que se transmiten por la red. Esto no puede ocurrir con SSH, ya que SSH nunca envía texto plano, sino que la información siempre viaja cifrada.

Existen dos versiones de SSH (SSH1 y SSH2) con diferentes funcionalidades e incompatibles entre sí (SSH2 soluciona problemas de seguridad de SSH1). A pesar de estas mejoras, existen razones a favor del uso de una y otra versión:

- SSH1 tiene fallos estructurales que le hacen vulnerable a ciertos tipos de ataques
- SSH1 puede sufrir un ataque del tipo "Man In The Middle" (intercepción de las claves públicas en el intercambio de las mismas)
- SSH1 está soportado en más plataformas que SSH2
- SSH1 permite autentificación de tipo .rhosts (propuesta en SSH2)
- SSH1 permite métodos de autentificación más diversos (AFS, Kerberos, etc.)
- SSH1 tiene mejor rendimiento que SSH2

El uso de SSH está muy extendido, y cuenta con más de 2 millones de usuarios en más de 60 países.

1.1 Algoritmos de cifrado

SSH, dependiendo de su versión utiliza diferentes algoritmos de cifrado.

- SSH1: DES, 3DES, IDEA, Blowfish
- SSH2: 3DES, Blowfish, Twofish, Arcfour, Cast128-cbc

El cifrado utilizado para cuestiones de autentificación utiliza RSA para SSH1 y DSA para SSH2.

1.2 Autentificación

SSH permite autentificarse utilizando uno o varios de los siguiente métodos:

- Password
- Sistema de clave pública
- Kerberos (SSH1)
- Basado en el cliente (por relaciones de confianza en SSH1 y sistema de clave pública en SSH2)

Existen además parches que permiten realizar autentificación de otras maneras.

1.3 Protección proporcionada por SSH

SSH protege contra los siguientes tipos de ataques:

- IP Spoofing: un ordenador trata de hacerse pasar por otro ordenador (en el que confiamos) y nos envía paquetes "procedentes" del mismo. SSH es incluso capaz de proteger el sistema contra un ordenador de la propia red que se hace pasar por el router de conexión con el exterior.
- Enrutamiento de la IP de origen: un ordenador puede cambiar la IP de un paquete procedente de otro, para que parezca que viene desde un ordenador en el que se confía.
- DNS spoofing: un atacante compromete los registros del servicio de nombres.
- Intercepción de passwords y datos a través de la red.

- Manipulación de los datos en ordenadores intermediarios.
- Ataques basados en escuchar autentificación contra servidores X-Windows remotos.

SSH considera la red como un medio hostil en el que no se puede confiar. Un atacante, podrá forzar a que SSH se desconecte, pero no podrá descifrar o reproducir el tráfico, o introducirse en la conexión.

Todo esto, de todos modos, es sólo posible si se utiliza algún tipo de cifrado. SSH permite no usar cifrado (cifrado de tipo "none"), pero esta opción no debería utilizarse.

Hay que tener en cuenta que SSH no sirve de nada si el atacante consigue hacerse con el control de la máquina como root, ya que podrá hacer que SSH quede inutilizado. Si algún atacante tiene acceso al directorio "home" de un usuario de la máquina, la seguridad es ya inexistente.

1.4 Sistemas operativos

SSH tiene numerosas implementaciones de sus servidores para diversos sistemas operativos, y proporciona clientes para plataformas muy diversas, como puede observarse en las siguientes tablas, en las que se muestran las plataformas y la máxima versión soportada.

	Propietario	**Gratis – no comercial**	**Gratis – sin condiciones**	**Open Source**
OpenVMS	-	-	-	1.5
OS/2	-	1.5	ninguno	ninguno
UNIX	2.0	2.0	ninguno	2.0
Win32	-	2.0	ninguno	2.0

Tabla 1.1. Versión de SSH soportada en servidores

	Propietario	**Gratis – no comercial**	**Gratis – sin condiciones**	**Open Source**
Amiga OS	1.5	1.5	nin	ninguno
BeOS	-	1.5	ninguno	ninguno
Java	-	-	ninguno	1.5
Macintosh	2.0	-	1.5	ninguno
OpenVMS	-	-	-	1.5
OS/2	2.0	1.5	ninguno	ninguno
PalmOS	-	-	1.5	ninguno
UNIX	2.0	2.0	1.5	2.0
Win16	2.0	1.4	ninguno	ninguno
Win32	2.0	2.0	1.5	2.0
WinCE	1.5	ninguno	-	ninguno

Tabla 1.2. Versión de SSH soportada en clientes

2. SSH1

2.1 El protocolo SSH1

Como se ha comentado anteriormente, el protocolo SSH1:

- Evita ciertos fallos de seguridad.
- Proporciona diferentes métodos de autentificación (uso de autentificación del cliente utilizando “.rhosts” junto con RSA, o autentificación utilizando sólo RSA).
- Todas las comunicaciones se cifran de manera automática y transparente. Este cifrado también se utiliza para proteger la integridad.
- El forwarding de conexiones X11 proporciona sesiones X11 seguras.
- Cualquier puerto TCP/IP puede ser redireccionado a un canal cifrado en ambos sentidos.
- El cliente autentifica al servidor al comienzo de cada sesión mediante RSA, y el servidor autentifica al cliente mediante RSA antes de aceptar autentificación mediante ".rhosts" o "/etc/hosts.equiv".
- Un agente de autentificación, en la máquina del usuario local, puede utilizarse para gestionar las claves RSA del usuario.

El propósito consiste en crear un software sencillo de utilizar por los usuarios normales y a su vez, un protocolo seguro.

2.2 Características del protocolo

El software se compone de un programa servidor en una máquina servidora, y un programa cliente en la máquina cliente (junto con algunos programas auxiliares). Las máquinas se conectan a través de una red IP no segura.

La conexión siempre se inicia por parte del cliente. El servidor está escuchando un puerto determinado, esperando la conexión, y puede atender a varios clientes.

Cuando el cliente se conecta al servidor, el servidor acepta la conexión y responde enviando su identificación de versión, a lo que el cliente responde

enviando la suya. Esta información valida que la conexión se realizó en el puerto correcto, establece la versión del protocolo utilizada, y la versión del software utilizada por cada uno de ellos. Esta información no se envía cifrada, y es perfectamente legible. Si cualquier lado de la conexión no acepta o no entiende esta información, se cierra la conexión.

Una vez a terminado la fase de identificación, ambos lados comienzan una transmisión según un protocolo binario. El servidor envía su clave de equipo (la clave RSA del propio equipo), la clave del servidor (una clave RSA generada cada hora), y otras informaciones al cliente. El cliente genera una clave de sesión de 256 bits, la cifra utilizando ambas claves RSA y envía la clave cifrada, junto con el tipo de cifrado seleccionado al servidor. Ambos lados comienzan ahora a enviar datos cifrados mediante la clave generada y el algoritmo seleccionado. El servidor envía un mensaje de confirmación cifrado al cliente.

Tras eso, el cliente se autentifica utilizando uno de los métodos de autentificación soportados:

- basada en los ficheros ".rhosts" o "/etc/hosts.equiv", por defecto no permitida
- esta misma junto con autentificación del ordenador cliente basada en RSA
- autentificación RSA
- y autentificación por password

Tras una correcta autentificación, el cliente envía una serie de peticiones para comenzar la sesión, como establecer una sesión de consola remota, forwarding de X11 o de un puerto TCP/IP, ejecutar un comando, etc.

Al ejecutar una consola o un comando, la sesión entra en modo interactivo. En este modo, se envían datos en ambos sentidos, pueden abrirse nuevas conexiones, etc. Esta sesión generalmente termina cuando el servidor envía en código de terminación del programa ejecutado al cliente.

El protocolo reserva cierta información para permitir extenderlo en el futuro:

- Envío de la versión del protocolo
- El primer paquete enviado por ambas partes, contiene una serie de flags, que pueden utilizarse para acordar el uso de extensiones compatibles

- En las fases de autentificación y preparación de la sesión, el cliente realiza peticiones al servidor, de modo que si se envía una petición no permitida por el servidor, éste simplemente devuelve que la petición ha fallado. Esto permite añadir nuevos métodos de autentificación y operaciones de preparación. En cambio, la sesión interactiva, no permite la negociación de extensiones. Éstas deberían negociarse en las fases anteriores.

2.3 Formato de los paquetes

Tras el envío de los strings de identificación, ambas partes envían paquetes con el siguiente formato:

- Tamaño del paquete: 32 bits. Es el tamaño del paquete, sin incluir el campo tamaño, y el relleno. El máximo es de 262144 bytes.
- Relleno: 1 - 8 bytes de datos aleatorios (o ceros si no se utiliza cifrado). El tamaño del relleno es de (8 - (tamaño % 8)) bytes. El hecho de que este relleno sea aleatorio dificulta los ataques.
- Tipo de paquete: 8 bits. El valor 255 está reservado para futuras ampliaciones.
- Datos: con un tamaño igual al valor del campo "tamaño" menos 5.
- Bytes de chequeo: CRC con el polinomio 0xedb88320 del relleno, tipo de paquete y datos. Se calcula antes del cifrado.

El paquete puede cifrarse utilizando cualquiera de los algoritmos soportados. La longitud de la parte cifrada (relleno + tipo + datos + chequeo) es siempre múltiplo de 8 bytes. Típicamente, el cifrado se utiliza sobre todos los paquetes, como si fuesen una única secuencia.

El cifrado se activa cuando el cliente envía la clave de sesión. El algoritmo de cifrado es seleccionado por el cliente.

2.4 Compresión de los paquetes

Si la compresión está soportada, el tipo de paquete y los datos se comprimen utilizando el algoritmo gzip. En este caso, el tamaño del paquete indica el tamaño de los datos comprimidos, más 4 para el CRC. El relleno se calcula según los datos comprimidos, para que los datos cifrados sean múltiplo de 8 bytes.

2.5 Cifrado de paquetes

El protocolo soporta distintos métodos de cifrado. Durante la inicialización de la sesión, el servidor envía al cliente los métodos que soporta, y el cliente elige uno de estos métodos, y genera una clave aleatoria de 256 bits que también envía al servidor.

Todas las implementaciones deben de soportar el algoritmo 3DES, siendo los demás opcionales.

Para cifrar los datos, las partes cifradas de cada paquete se consideran un flujo de datos continuo, cuyo tamaño es siempre múltiplo de 8 bytes. Las partes cifradas de paquetes consecutivos se cifran como si se tratase de un buffer continuo de datos. Los datos se cifran independientemente en cada dirección.

2.6 Puerto TCP/IP y otras opciones

El puerto por defecto para el servidor es el 22.

El cliente puede conectarse desde cualquier puerto, pero si quiere utilizar autentificación mediante ".rhosts" o "/etc/hosts.equiv", debe conectarse a través de un puerto privilegiado (menor a 1024).

El campo IP "Tipo de Servicio" debería de ser IPTOS_LOWDELAY para conexiones interactivas, y IPTOS_THROUGHPUT para las no interactivas.

Se recomienda así mismo que se utilicen señales para comunicar que la conexión sigue activa, para detectar si alguno de los lados ha cortado la conexión de modo inusual.

2.7 Identificación de la versión del protocolo

Una vez que se ha abierto el socket, el servidor envía un string con el formato "SSH-<num_mayor_del_protocolo>.<num_menor_del_protocolo>-<versión>\n". Los dos primeros campos identifican la versión del protocolo, y

el último campo, la versión del software del servidor.

El cliente envía después su propia información. Si el servidor tiene un protocolo menor que el cliente, y éste puede emularlo, envía la versión menor. En caso contrario, envía su propia versión. El servidor compara la versión enviada por el cliente con la suya, y determina si pueden trabajar o no. El servidor decidirá desconectarse, o enviará el primer paquete binario, y ambas partes utilizarán después la versión del protocolo acordada.

2.8 Intercambio de claves y autentificación del equipo servidor

El primer mensaje enviado por el servidor envía la clave del equipo, la clave pública del servidor, los algoritmos de cifrado soportados, los métodos de autentificación soportados y las extensiones. También contiene un número aleatorio de 64 bits (cookie). Este paquete se envía sin cifrar.

Ambas partes calculan un identificador de sesión del siguiente modo:

session_id = MD5(hostkey || serverkey || cookie)

El cliente responde con un mensaje que contiene el tipo de cifrado elegido, una copia del cookie, y la clave de sesión de 256 bits cifrada con ambas claves del servidor. Este mensaje no se cifra.

Una vez que se ha enviado la clave de sesión, el cliente cifrará todos los paquetes salientes y descifrará todos los paquetes entrantes.

Una vez que el servidor recibe la clave, y activa el cifrado, envía al cliente un mensaje indicando el éxito de la operación.

La clave de equipo del servidor se recomienda que tenga 1024 bits, y la clave de servidor, 768 bits. La clave mínima no puede ser menor de 512 bits.

2.9 El nombre de usuario

El cliente envía al servidor un mensaje indicando el nombre de usuario con el que se quiere conectar.

El servidor valida que el usuario existe, si se necesita autentificación y responde con un mensaje de éxito (no se necesita autentificar el usuario) o fallo (se necesita autentificación, o el usuario no existe).

Si el usuario no existe, se devolverá fallo a todos los mensajes, excepto al mensaje de desconexión, al mensaje "ignore" y al mensaje "debug", pero seguirá escuchando al cliente, de modo que este no puede saber si el usuario existía o no.

2.10 Fase de autentificación

Si no se acepta el login automáticamente, el cliente pide al servidor distintos métodos de autentificación de manera aleatoria tantas veces como desee. Si la autentificación es correcta se devolverá un mensaje de éxito, y si falla, un mensaje de fallo.

Autentificación por RHOSTS

El cliente envía un mensaje con el nombre del usuario cliente. El servidor verifica este nombre contra los ficheros "/etc/hosts.equiv" y ".rhosts" (sistemas UNIX). La conexión debe de venir desde un puerto privilegiado.

Este tipo de verificación no suele activarse puesto que puede sufrir diversos ataques.

Autentificación por RHOSTS y RSA

El cliente no sólo envía el nombre de usuario, sino también su clave pública RSA. El servidor autentifica al cliente por el método RHOSTS, y si se verifica, comprueba si conoce su clave pública, y si la enviada es la misma almacenada en el servidor.

Si cualquiera de estos pasos no se cumple, se devolverá un mensaje de fallo. En caso contrario, se someterá al cliente a un desafío RSA con la clave pública

del cliente. El cliente deberá descifrar un mensaje cifrado con su clave pública, utilizando su clave privada, y enviará la respuesta al servidor, que verificará si se ha conseguido pasar el desafío o no.

Autentificación por RSA

En este caso, el cliente envía su clave pública, y si el servidor la admite, envía al cliente un desafío RSA.

En el caso de que el cliente lo supere, se permitirá el acceso del mismo.

Autentificación por password

El cliente envía al servidor el password del usuario, en texto plano (aunque lo normal será que viaje cifrado).

Si el password es correcto, se permitirá el acceso.

2.11 Operaciones de preparación

Una vez que el servidor ha admitido la conexión por parte del cliente, se negocian las opciones que van a utilizarse durante el intercambio de datos.

2.12 Sesión interactiva e intercambio de datos

Durante una sesión interactiva, los datos de salida el proceso en ejecución en el servidor se redireccionan a "stdin" o "stderr" en el cliente, y cualquier entrada disponible en "stdin" en el cliente se envía al programa en el servidor.

El envío de datos es asíncrono, y tanto el cliente como el servidor pueden enviar datos al mismo tiempo.

Cuando la aplicación termine de ejecutarse en el servidor, se enviará un mensaje con el estado de finalización del programa, y se cerrará la conexión.

Además, de esto, la conexión puede ser cerrada en cualquier momento por cualquiera de los dos ordenadores.

3. SSH2

SSH2 es un protocolo que permite servicios de red y conexiones seguras sobre una red insegura. Consta de tres componentes:

- Capa de protocolo de transporte: proporciona autentificación del servidor, confidencialidad e integridad y, opcionalmente, compresión. Generalmente se ejecutará sobre una conexión TCP/IP.
- Capa de protocolo de autentificación: autentifica al cliente contra el servidor. Se ejecuta sobre la capa del protocolo de transporte.
- Capa del protocolo de conexión: multiplexa el canal cifrado en múltiples canales lógicos. Se ejecuta sobre la capa de autentificación.

El cliente envía una petición de servicio una vez que se ha establecido una conexión segura en la capa de transporte, y una segunda petición tras la autentificación del usuario. Esto permitiría definir nuevos protocolos que coexistieran con los anteriormente definidos.

El protocolo de conexión proporciona canales que pueden utilizarse para diferentes propósitos, como abrir sesiones interactivas, redireccionar puertos TCP/IP (tunneling) y conexiones X11.

3.1 Arquitectura

Claves de equipo

Cada servidor debería de tener una clave de equipo. Es posible tener varias, con distintos algoritmos, e incluso varios equipos pueden compartir la misma clave. Si un equipo tiene claves, al menos debe de tener una clave utilizando el algoritmo obligatorio (DSS).

La clave se utiliza para verificar que el cliente está conectado al servidor correcto, para lo que el cliente debe conocer de antemano la clave pública del servidor.

Pueden utilizarse dos modelos:

- El cliente tiene una base de datos que asocia servidores con sus

correspondientes claves públicas.

- Las asociaciones vienen dadas por una autoridad de certificación.

El protocolo permite que, la primera vez que el cliente se conecte con un servidor, no se compruebe su clave pública, con lo que esta primera vez no sería necesaria, pero esto hace que la conexión sea vulnerable, por lo que no se recomienda, aunque a veces es necesaria esta opción.

Todas las implementaciones del protocolo deberían hacer todo lo posible por comprobar siempre esta clave.

Extensibilidad

El protocolo está diseñado de manera que pueda resultar fácilmente extensible, añadiendo nuevos algoritmos, protocolos, tipos de datos, etc.

Negociación

El protocolo permite negociar el cifrado, la integridad, el intercambio de claves, la compresión, y los algoritmos y formatos de las claves públicas. Los algoritmos de cifrado, integridad, clave pública y compresión pueden ser diferentes en cada sentido de la conexión.

Propiedades de seguridad

- Los algoritmos de integridad, cifrado y clave pública utilizados son conocidos y ampliamente probados.
- Los algoritmos utilizan claves lo suficientemente largas como para resistir durante décadas los más fuertes ataques de criptoanálisis.
- Como los algoritmos se negocian, si uno se ve comprometido es posible negociar el uso de otro sin cambiar el protocolo.

Consideraciones de seguridad

El protocolo de transporte proporciona un canal confidencial sobre una red no

segura. Efectúa la autentificación del servidor, el intercambio de claves, el cifrado y la protección de la integridad. También proporciona un identificativo de sesión único que será utilizado por los protocolos superiores.

El protocolo de autentificación proporciona mecanismos para autentificar al cliente contra el servidor.

El protocolo de conexión especifica un mecanismo para multiplexar diferentes flujos de datos (canales) sobre un canal de transporte confidencial y autentificado. También especifica canales para acceder a consolas interactivas, redireccionar protocolos al canal de transporte seguro, y acceder a subsistemas seguros en el servidor.

3.2 Protocolo de transporte

Este protocolo proporciona cifrado fuerte, autentificación del servidor y protección de la integridad de los datos. No autentifica al cliente, sino que permite hacer esto en un protocolo superior.

El protocolo de transporte negocia el método de intercambio de claves, el algoritmo de clave pública, el algoritmo de cifrado simétrico, el algoritmo de autentificación de mensajes y el algoritmo de hash. En el mejor de los casos, se necesitaran dos intercambios de mensajes entre el cliente y el servidor para esta negociación, y en el peor, tres.

Establecimiento de la conexión

Utilización sobre TCP/IP

Sobre TCP/IP, el servidor generalmente escucha conexiones en el puerto 22.

Intercambio de versión del protocolo

Cuando se establece la conexión, ambos lados deben enviar un string de identificación con el formato:

"SSH-versión_protocolo-versión_software comentarios\r\n"

Formato de los paquetes

Cada paquete tiene el siguiente formato:

```
uint32    tamaño paquete
byte      tamaño relleno
byte[n1]  datos; n1 = tamaño paquete - tamaño relleno – 1
byte[n2]  relleno aleatorio; n2 = tamaño relleno
byte[m]   mac (message authentication code); m = tamaño_mac
```

- Tamaño del paquete: el tamaño del paquete sin incluir el campo MAC ni el propio campo tamaño.
- Tamaño del relleno: el tamaño del relleno
- datos: los datos útiles del paquetes. Si la compresión está activada, está comprimido. Inicialmente, la compresión debe de estar desactivada ("NONE").
- Relleno aleatorio: relleno para que el tamaño de todo el paquete, excepto el campo mac, tenga un tamaño múltiplo de 8, o el tamaño del bloque de cifrado, lo que sea mayor. Tiene que tener, como mínimo, 4 bytes, y cómo máximo 255.
- mac: bytes para la autentificación del mensaje.

El tamaño mínimo del paquete es 16 bytes, y el tamaño máximo del paquete debe de poder enviar 32768 bytes de datos sin comprimir, con un tamaño total del paquete de 35000 bytes. Las implementaciones deberían de soportar paquetes mayores para tareas específicas.

Compresión

Si se ha negociado la compresión, el campo de datos estará comprimido según el algoritmo negociado. El campo del tamaño y la MAC se calcularán según los datos comprimidos. El cifrado se realizará tras la compresión.

La compresión debe de poder negociarse independientemente para cada sentido de la transmisión.

Es obligatorio no soportar compresión, y opcional soportar compresión con el algoritmo "zlib".

Cifrado

El algoritmo de cifrado se negociará durante el intercambio de claves. Los campos cifrados serán: tamaño del paquete, tamaño del relleno, datos y relleno.

Los datos enviados en una dirección deberían de ser tratados como un flujo continuo de datos, y las claves de los algoritmos deberían tener, al menos 128 bits.

Los algoritmos pueden ser diferentes en cada dirección de la comunicación.

El único algoritmo requerido es el 3des-cbc, aunque pueden implementarse una gran cantidad de algoritmos opcionales.

Es también posible especificar que el tráfico no sea cifrado, con lo que se perdería la confidencialidad, por lo que no se recomienda.

Integridad de los datos

En cada paquete se introduce un código de autentificación de mensaje (MAC) calculado a partir de una clave compartida por el cliente y el servidor, el número de secuencia del paquete, y el contenido del mismo.

El algoritmo es independiente en cada sentido de la conexión.

El único algoritmo que ha de implementarse de manera obligatoria es el algoritmo hmac-sha1.

Métodos de intercambio de claves

Especifican cómo se generan las claves de sesión, y cómo se autentifica el servidor.

El único algoritmo obligatorio es el Diffie-Hellman

Algoritmos de clave pública

El algoritmo requerido por toda implementación es el algoritmo DSS, aunque pueden implementarse muchos más.

El tipo de clave debe conocerse de antemano (por ejemplo, especificándolo en la negociación del algoritmo).

Intercambio de claves

Cada equipo envía una lista de los algoritmos que soporta. Cada una de las partes tiene un algoritmo preferido para cada una de las categorías, y se presupone que la mayoría de las implementaciones van a usar siempre el mismo algoritmo preferido. Cada parte puede suponer qué algoritmo está utilizando el otro equipo, y puede enviar un paquete de intercambio de claves según ese algoritmo si corresponde con el algoritmo predefinido.

Esta suposición es incorrecta si:

- El algoritmo de intercambio y/o el algoritmo de clave de equipo se presuponen de manera incorrecta (el algoritmo predefinido es distinto en el cliente y en el servidor)
- No pueden ponerse de acuerdo en cualquiera de los otros algoritmos

En caso contrario, la suposición se considera correcta, y el primer paquete enviado debe de ser gestionado como el primer paquete de intercambio de claves.

De todos modos, si la suposición es incorrecta, y se ha enviado algún paquete de intercambio de claves, estos paquetes serán ignorados, y el lado implicado deberá de enviar un paquete de inicio correcto.

La autentificación del servidor en el intercambio de claves puede ir implícita. Tras un intercambio de claves con autentificación implícita, el cliente debe esperar una respuesta a su petición de servicio antes de enviar más datos.

Negociación de algoritmos

El intercambio de claves comienza con un paquete enviado por ambas partes con la siguiente información:

```
byte                Código de comando
byte[16]            cookie
string              algoritmos de intercambio de claves
string              algoritmos de clave de servidor
string              algoritmos de cifrado cliente/servidor
string              algoritmos de cifrado servidor/cliente
string              algoritmos de compresión cliente/servidor
string              algoritmos de compresión servidor/cliente
string              algoritmos de mac cliente/servidor
string              algoritmos de mac servidor/cliente
string              lenguajes cliente/servidor
string              lenguajes servidor/cliente
boolean             el siguiente paquete es el primero de intercambio de claves
uint32              (reservado para extensiones)
```

El primer algoritmo de cada lista debe de ser el preferido por la parte (el que se presupondrá), y ninguna lista de algoritmos puede ser vacía.

Si ambas partes tienen el mismo algoritmo de intercambio de clave, se utilizará ese algoritmo. En caso contrario, se seguirá el siguiente algoritmo, iterando sobre los algoritmos soportados por el cliente. Se elegirá el primero que:

- esté soportado por el servidor
- si necesita cifrado, hay un algoritmo de cifrado de clave de host (clave del equipo servidor) en el servidor que también está soportada en el cliente
- si necesita una clave del equipo servidor con firma, este algoritmo de firma existe en el servidor, y en el cliente
- si ningún algoritmo satisface estas condiciones, ambos lados deben desconectarse

Tras en envío de este paquete, se produce el intercambio de claves según el algoritmo escogido.

Resultado del intercambio de claves

Este intercambio produce dos valores: una clave secreta K, y un valor de intercambio H. El valor H del primer intercambio de claves es también el identificativo de sesión, y se genera a partir de una función de HASH.

Uso de las claves

El intercambio de claves termina con el envío de un comando SSH_MSG_NEWKEYS. Este mensaje se envía utilizando las claves y algoritmos que se utilizaban hasta ese momento, y tras su envío, todos los demás paquetes que se envíen lo harán con estas nuevas claves y algoritmos.

Cuando este mensaje se recibe, los nuevos algoritmos y claves deberán utilizarse para recibir.

Tras el intercambio de claves, este es el único paquete válido (junto con la orden de desconexión, el paquete "ignore" y el paquete "debug"), para que ambos equipos confirmen que todo ha ido bien.

En el caso del intercambio de claves Diffie-Hellman, se calcula una clave secreta, compartida por ambos equipos, que no puede determinarse sino es por colaboración de ambos. El intercambio de claves se combina con una firma utilizando la clave del host del servidor para autentificar a éste.

Re-intercambio de claves

Si se recibe un mensaje de intercambio claves (y no se está realizando ya uno), ambas partes vuelven a negociar la clave.

Este intercambio se realiza utilizando el cifrado activo en ese momento. Los métodos de cifrado, compresión y MAC no se cambian hasta que termine la negociación, y se procesa de manera idéntica al intercambio inicial (excepto que el identificativo de sesión no cambia).

Se recomienda cambiar las claves después de la transferencia de un gigabyte, o una hora de conexión.

Tras el intercambio se puede continuar enviado datos.

Petición de servicio

Tras el intercambio inicial de claves, el cliente solicita un servicio, identificado por un nombre. Los servicios inicialmente implementados son el de autentificación y el de petición de conexión.

Si el servidor no acepta la petición, deberá desconectarse, y si lo acepta, deberá notificarlo al cliente.

El servicio puede tener acceso al identificador de sesión generado en el intercambio de claves.

3.3 Protocolo de autentificación

El protocolo de autentificación SSH es un protocolo de autentificación de propósito general que trabaja sobre el protocolo de la capa de transporte SSH. Presupone que los protocolos inferiores proporcionan integridad y confidencialidad de los datos.

Este protocolo recibe el identificador de sesión generado por el protocolo inferior. Este identificador de sesión es único para la misma, y adecuado para realizar firmas, de modo que pueda probarse la posesión de una clave privada. También necesita conocer si el protocolo inferior realmente proporciona confidencialidad.

Consideraciones del protocolo de autentificación

El servidor dirige la autentificación diciendo al cliente los métodos soportados. El cliente puede elegir cualquiera de ellos e ir probando en cualquier orden. De este modo el servidor tiene el control, pero cliente la suficiente flexibilidad para resultar cómo para el usuario.

Peticiones de autentificación

Todas las peticiones de autentificación deben de tener el siguiente formato:

```
byte                código de comando (SSH_MSG_USERAUTH_REQUEST)
string              nombre de usuario
string              nombre del servicio
string              método de autentificación
datos dependientes del método
```

Si el usuario no existe, el servidor puede desconectarse, o enviar un listado incorrecto de métodos de autentificación, pero nunca aceptar ninguno.

El resto de mensajes dependerán del método de autentificación escogido, que puede ser cambiado por el cliente en cualquier momento, con lo que el servidor deberá abandonar el proceso anterior, comenzar el nuevo.

Respuestas a peticiones de autentificación

Si el servidor rechaza la petición, enviará un mensaje de fallo con los métodos de autentificación portados. En caso de que la autentificación se acepte, se enviará un mensaje indicándolo.

Si la autentificación consta de varios pasos, se responderá a cada paso individual con un mensaje de fallo, que contendrá un campo "éxito parcial" a verdadero. El mensaje de aceptación sólo se enviará al final de todo el proceso.

Cualquier mensaje que no sea de autentificación enviado por el cliente una vez se haya aceptado su petición, deberá pasarse al servicio activo sobre este protocolo.

Método de autentificación "none"

Este método no debería estar activo para ningún usuario, pero en ciertos casos puede estarlo. El cliente puede solicitarlo, para recibir un mensaje de fallo con todos los métodos permitidos.

Si el servidor está configurado para aceptar el método "none" desde ese cliente, deberá enviar un mensaje de éxito. En caso contrario, un mensaje de fallo con los métodos aceptados, que no deben incluir nunca el método "none" (aunque se acepte).

Mensaje de aviso

En ciertas jurisdicciones, puede ser necesario enviar un mensaje de aviso al comenzar la sesión para obtener asistencia legal.

Este protocolo permite al servidor enviar un mensaje (como el mostrado en sistemas UNIX) al cliente, para que este lo muestre, tras cualquier autentificación que se halla llevado a cabo con éxito.

Método de autentificación por clave pública

Es el único método requerido en todas las implementaciones.

Con éste método, la autentificación se prueba si se posee una clave privada. Se envía una firma creada con la clave privada del usuario. El servidor comprueba que la clave es válida para el usuario, y que la firma también es válida. Si ambas condiciones se cumplen, la petición se acepta. En caso contrario, se rechaza.

El proceso es el siguiente:

El cliente envía al servidor la petición de autentificación con su clave pública. Si el servidor no acepta el método, envía un mensaje de fallo. En caso contrario, envía un mensaje de aceptación del método pedido.

Si se acepta, el cliente enviará un nuevo paquete de petición de autentificación de clave pública, pero incluyendo una firma creada con su clave privada, que firma todos los demás campos del mensaje, incluyendo el identificativo de sesión como campo firmado.

Cuando el servidor recibe este mensaje, comprueba que la clave proporcionada es correcta, y si lo es, comprueba también la firma. El servidor entonces responderá con un mensaje de éxito o fallo.

Método de autentificación por password

El cliente envía un mensaje de autentificación con su password, con el

formato:

```
byte      SSH_MSG_USERAUTH_REQUEST
string    user name
string    service
string    "password"
boolean   FALSE
string    plaintext password
```

Generalmente, el servidor responderá con éxito o fallo, pero en el caso de que el password haya caducado, enviará el siguiente mensaje:

```
byte      SSH_MSG_USERAUTH_PASSWD_CHANGEREQ
string    prompt
string    language tag
```

En este caso, el cliente puede continuar con un método de autentificación diferente, o pedir un nuevo password al usuario y volver a intentarlo, con el siguiente mensaje:

```
byte      SSH_MSG_USERAUTH_REQUEST
string    user name
string    service
string    "password"
boolean   TRUE
string    plaintext old password
string    plaintext new password
```

El servidor puede responder con un mensaje de éxito, fallo, o una nueva petición de cambio de clave, en el caso de que considere que el nuevo password no es válido.

Autentificación basada en la máquina cliente

Es un método de autentificación similar al "rhosts" o "hosts.equiv", pero más riguroso.

El método funciona del siguiente modo: el cliente envía una petición de autentificación, firmada con su clave privada de equipo, que el servidor comprueba con la clave pública del mismo. Una vez que se ha establecido la identidad el equipo cliente, la autorización se realiza basándose en el nombre de usuario y el nombre de la máquina.

La firma incluye todos los campos del paquete de petición de autentificación, y

el identificativo de sesión.

El servidor verificará que la clave pública pertenece al equipo indicado, que el usuario en ese equipo tiene permiso de conexión, y que la firma es válida para esa clave.

3.4 Protocolo de conexión

El protocolo de conexión SSH está diseñado para funcionar sobre los protocolos de transporte y autentificación de usuario SSH. Proporciona sesiones de conexión interactivas, ejecución remota de comandos, redirección de conexiones TCP/IP y redirección de conexiones X11.

Peticiones globales

Existen peticiones que afectan al estado del equipo remoto de manera "global", independientemente de cualquier canal. Un ejemplo sería comenzar la redirección de un determinado puerto TCP/IP.

El destinatario responderá a estas peticiones con un mensaje de éxito o fallo, y se indica en la petición que se espera una respuesta.

Mecanismo de canales

Todas las sesiones de terminal, conexiones redireccionadas, etc., con canales. Cualquier parte de la conexión puede abrir un canal, y los distintos canales se multiplexan sobre una sola conexión.

Los canales se identifican con un número en cada extremo, que puede ser distinto en cada parte. Cualquier mensaje perteneciente a un canal, incluye el número de canal en el extremo opuesto.

Los canales tienen control de flujo, y no se envían datos al canal hasta que se reciba un mensaje diciendo que existe espacio en la ventana de recepción.

Apertura de un canal

Cuando cualquier parte quiere abrir un nuevo canal, reserva un número local para el mismo, y envía un mensaje al otro extremo, que incluye el número local, el tipo de canal que quiere abrir y el tamaño inicial de la ventana.

El extremo remoto decide si puede abrir el canal, y responde con un mensaje de confirmación, que contiene el número que usará él para el canal, y el tamaño de la ventana, o bien con un mensaje de fallo, que contiene el motivo del fallo (un código, y un mensaje).

Transferencia de datos

El tamaño de la ventana indica cuántos bytes puede enviar el otro lado antes de recibir un mensaje de ajuste de ventana.

Tras recibir un mensaje de ajuste de ventana, el receptor del mismo podrá enviar más datos, según el nuevo tamaño de la misma.

La transferencia de datos se realiza con paquetes con el siguiente formato:

```
byte      SSH_MSG_CHANNEL_DATA
uint32    canal del receptor
string    datos
```

El máximo tamaño es el actual tamaño de la ventana de transmisión, y el tamaño de ésta se decrementará en el tamaño de los datos enviados.

En ciertos tipos de canales, además, se puede enviar un código que indica el tipo de datos que se están enviando:

```
byte      SSH_MSG_CHANNEL_EXTENDED_DATA
uint32    canal del receptor
uint32    tipo de datos
string    datos
```

Cierre de un canal

Cuando una parte no va a enviar más datos por un canal, debería enviar un

mensaje de fin de datos:

```
byte      SSH_MSG_CHANNEL_EOF
uint32    canal del receptor
```

Tras este mensaje, el canal continúa abierto, y se pueden enviar datos desde el otro extremo.

Cuando cualquier extremo de la conexión quiere cerrar el canal, manda un mensaje de cierre:

```
byte      SSH_MSG_CHANNEL_CLOSE
uint32    recipient_channel
```

Al recibir este mensaje, el otro extremo debe devolver un mensaje de cierre, a no ser que lo hubiera enviado antes.

Peticiones específicas del canal

Muchos canales tienen extensiones propias. Un ejemplo puede ser solicitar un pty (pseudo terminal) para una sesión interactiva. En ese caso, se enviará un comando de petición que será propio del canal.

Si no se espera petición, no se enviará. En caso contrario, se enviará un mensaje de éxito o fallo de la petición.

Sesiones interactivas

Una sesión es una ejecución remota de un programa. El programa puede ser una consola, una aplicación, un comando del sistema o un subsistema. Puede tener o no una terminal y puede requerir redirección de trafico sobre X11. Además, múltiples sesiones pueden estar activas a la vez.

Para realizar todas estas operaciones, se enviarán diferentes tipos de mensajes para apertura de canales y solicitud de opciones en los mismos.

4. Utilización de SSH

En este apartado se comentarán algunos de los usos más comunes de SSH. Se distinguirán los diferentes comandos de SSH1 y SSH2 y sus modos de uso.

4.1 SSH1

ssh1

Mediante este comando, es posible abrir una sesión remota. Para abrir esta sesión con el mismo usuario con el que estamos conectados a nuestro ordenador local, tan sólo es necesario escribir:

$ ssh ordenador.remoto.org

Para utilizar un nombre de usuario diferente:

$ ssh -l nombreusuario ordenador.remoto.org

Utilizar SSH1 para enviar un comando a otro ordenador, de manera segura:

$ ssh ordenador.remoto.org comando

También es posible especificar el tipo de cifrado, activar la compresión en la transmisión de datos, definir la localización de los ficheros de configuración, reenviar comandos, y muchas otras utilidades.

Por ejemplo, para utilizar ssh sin cifrado (como se comentó anteriormente, esta práctica no se recomienda) podríamos utilizar:

$ ssh -o cipher=none ordenador.remoto.org

scp1

El comando scp permite copiar ficheros de manera segura entre dos ordenadores. Para copiar un fichero desde el ordenador local al ordenador remoto:

$ scp dir/local/fichero usuario@host:/directorio/destino

Para copiar un fichero desde el ordenador remoto al ordenador local:

$ scp usuario@host:/directorio/origen/fichero dir/local

Para mantener los mismos atributos del fichero, será necesario utilizar la opción -p

$ scp -p usuario@host:/directorio/origen/fichero dir/local

Este comando puede utilizarse en conjunción con la mayor parte de los parámetros del comando "cp" existente en los sistemas Unix. Por ejemplo, la siguiente instrucción copiará de manera recursiva todo el contenido del directorio remoto en nuestro ordenador local:

$ scp -r usuario@host:/directorio/origen directorio/local

y para realizar esta copia utilizando compresión en el envío:

$ scp -r -C usuario@host:/directorio/origen directorio/local

sshd1

Este comando permite activar el daemon SSH1 en el ordenador, y generalmente es ejecutado por el usuario "root". Para su uso, tan sólo es necesario ejecutar:

sshd

Este comando puede recibir multitud de parámetros, para definir ficheros de configuración, puerto por defecto, etc.

ssh-agent1

Este comando permite cargar las claves públicas en memoria, para que puedan ser utilizadas. La idea de este comando es que sea ejecutado al comienzo de una sesión de login o una sesión X-Windows, de modo que todos los programas que se ejecuten en esas sesiones utilicen de manera automática el agente para autentificarse mediante RSA a los ordenadores remotos usando SSH.

Su uso es muy sencillo, aunque admite parámetros:

```
$ ssh-agent
```

Para abrir una sesión xterm:

```
$ ssh-agent xterm &
```

ssh-add1

Permite añadir una identidad (su clave RSA) al agente ssh. Para poder ejecutar este comando, es necesario que el proceso ssh-agent sea un antecesor del proceso que ejecuta ssh-add.

```
$ ssh-add identidad
```

También permite eliminar y gestionar las identidades dadas de alta.

4.2 SSH2

ssh2

El comando ssh2 es muy similar al comando ssh de SSH1, pero con opciones adicionales, como activar el forwarding de la autentificación o el tráfico de X-Windows.

```
$ ssh2 remote.example.org
```

scp2

El comando scp2, al igual que scp, permite copiar ficheros de manera segura, pero con funcionalidades añadidas, como mover ficheros en lugar de copiarlos, con el parámetro -u (similar al comando "smv" o "secure move").

```
$ scp2 -u -p user@host:/dir/for/file localdir/to/filelocation
```

sshd2

Similar al comando sshd, arranca el daemon de SSH2. Permite especificar parámetros de funcionamiento. Lo más habitual es ejecutarlo como el usuario "root".

```
# sshd2
```

ssh-agent2

Su utilización es similar a la utilización de su homólogo en SSH1:

```
$ ssh-agent2
```

ssh-add2

Permite gestionar las identidades del agente SSH2.

```
$ ssh-add2 identidad
```

sftp2

Este comando permite realizar transacciones ftp de manera segura, cifrando tanto la conexión de datos como la conexión de control.

$ sftp servidor_remoto usuario

El servidor remoto tiene que estar ejecutando el daemon sshd2.

4.3 Administración de SSH

El principal problema a la hora de administrar ssh es el mantenimiento de las claves públicas de los clientes. Para permitir a un cliente conectarse al servidor, y autentificarse como un equipo válido mediante clave pública, el servidor tiene que conocer de antemano la clave pública del cliente.

En el caso de SSH1, existen aplicaciones (como "make-ssh-known-hosts.pl", en la propia distribución de ssh, o "ssh-keyscan", una aplicación externa) que permiten recoger y actualizar estas claves, de modo automático en periodos de tiempo programados.

Pero cuando es necesario añadir un nuevo equipo, o una persona ha cambiado su clave, resulta necesario verificar de forma manual este cambio, o esta nueva clave, para asegurarnos que no ha ocurrido un ataque de tipo "Man In The Middle".

Con SSH2, el proceso es algo más tedioso, pero más seguro, ya que no permite que ocurran ataques de tipo "Man In The Middle".

El primer paso es generar el par de claves pública/privada en el cliente mediante el comando:

ssh-keygen2 -P /etc/ssh2/hostkey

El segundo paso consiste en copiar el fichero de clave pública del cliente, en el servidor, con el nombre:

/etc/ssh2/knownhosts/nombre_cliente.ssh-dss.pub

El nombre del equipo cliente debería de ser el nombre completamente cualificado (o fully qualified hostname, es decir, su nombre y dominio completos: cliente.dominio.org).

Tras esto, en el directorio "home" del usuario remoto con el que queremos acceder al servidor, es necesario crear un fichero con el nombre ".shosts", que contenga el nombre del ordenador cliente, y el nombre del usuario con el que nos conectaremos en dicho ordenador. Este fichero deberá de tener permisos sólo de lectura, y sólo para el usuario del ordenador remoto.

Finalmente, en el fichero de configuración de SSH2 de ambos ordenadores, debe de estar especificado que vamos a validar la conexión según la confianza en el equipo cliente. Este fichero (/etc/ssh2/sshd2_config en el servidor, y /etc/ssh2/ssh2_config en el cliente) debe de contener la opción "hostbased" dentro de la sección "AllowedAuthentications", por ejemplo:

AllowedAuthentications passwd, hostbased

4.4 Uso de autentificación con clave pública de usuario en SSH2

En el ordenador cliente, es necesario crear un par de claves pública/privada (utilizando ssh-keygen), y copiar la parte pública en el directorio $HOME/.ssh2/ del servidor.

Una vez hecho esto, es necesario incluir la siguiente línea en el fichero $HOME/.ssh/authorization del servidor:

key id_dsa_1024_a.pub # nombre del fichero con la clave pública

y en el fichero $HOME/.ssh/identification del servidor:

idkey id_dsa_1024_a # nombre del fichero con la clave privada

5. SSH y otras aplicaciones

5.1 Uso de SSH tras un firewall

Para poder utilizar SSH tras un firewall, tan sólo es necesario tener abierto un puerto en el mismo, y hacer que el daemon remoto SSH escuche el tráfico de ese puerto.

Generalmente se utiliza el puerto 22 (estándar de SSH) pero puede utilizarse cualquier otro puerto que esté permitido por el firewall (por ejemplo, el puerto 443 de SSL).

Para ello, en el servidor, tan sólo es necesario arrancar el daemon SSH en el puerto deseado:

```
# sshd -p 443
```

Y establecer la conexión desde el cliente abriendo una conexión a ese puerto:

```
$ ssh -p 443 host.remoto.org
```

Esto nos permitirá también redirigir el tráfico no seguro a través del firewall, mediante "tunneling", técnica que se comentará a continuación.

5.2 Tunneling con SSH

El tunneling con SSH permite enviar datos basados en protocolos no seguros a través de un canal SSH seguro. De este modo, protocolos como POP3, o FTP pueden utilizarse sin peligro de que datos como los passwords sean enviados como texto plano, e interceptados a través de la red.

La idea consiste en crear un túnel a través del que los datos viajarán de manera segura desde un extremo al otro. Por tomar un símil con la vida real: imaginemos que viajar en tren es más seguro que viajar en coche. Si queremos viajar en coche de manera más segura, podemos subir nuestro coche en un vagón para coches de un tren. Estaremos viajando con la

seguridad que nos proporciona el tren, en lugar de la seguridad del coche.

Este es el mismo proceso 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.

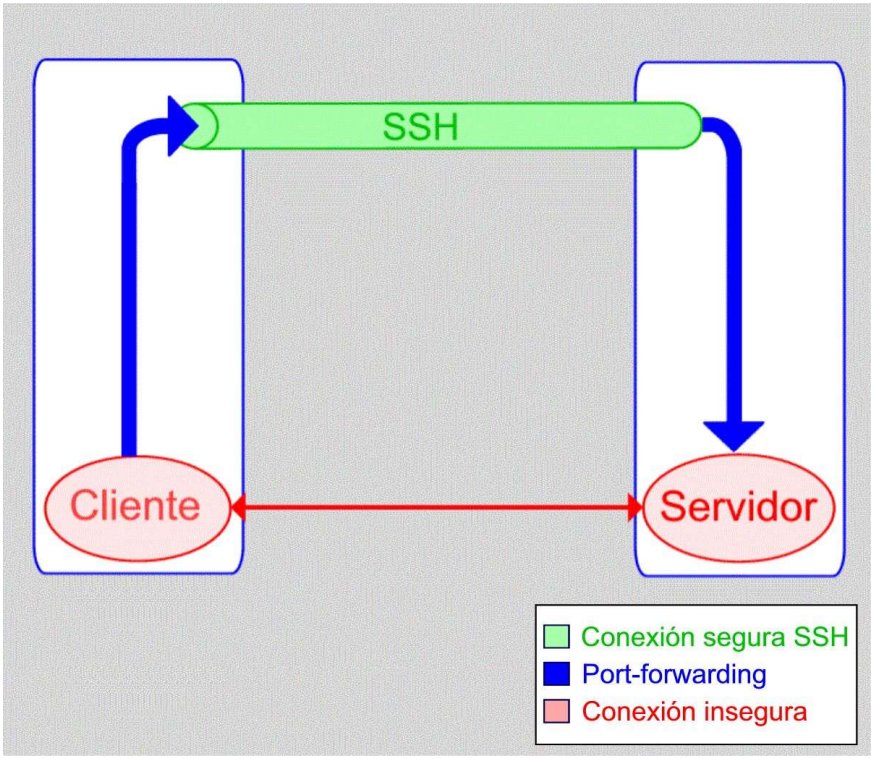

Ilustración 5.1. Tunneling con SSH

De modo práctico (utilizando los comando proporcionados por la implementación libre OpenSSH) la manera de hacer esto sería la siguiente:

Si queremos utilizar el puerto 10110 para crear una conexión segura al puerto 110 de una máquina remota, haremos:

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

Para poder utilizar puertos privilegiados en nuestra máquina, tendremos que realizar esta operación como root.

Algunos ejemplos de protocolos que podemos utilizar de este modo son:

- FTP (puerto 22): Es necesario tener en cuenta que FTP tiene dos conexiones distintas: una para comandos y otra para datos. Si no utilizamos el comando sftp, la manera habitual se centra en crear sólo un túnel seguro para el canal de datos. Es también necesario utilizar el modo pasivo en la conexión de datos para que el tunneling funcione correctamente, al igual que permitir que en el servidor, la conexión de datos y de control vengan de máquinas diferentes.
- SMTP (puerto 25): Para poder utilizar tunneling de SMTP, es necesario que el servidor de correo se acepte a sí mismo (localhost) dentro de su política de relay, puesto que recibirá las conexiones como si provinieran de la propia máquina.
- HTTP (puerto 80): Existen gran cantidad de páginas con secciones privadas, que envían nombres de usuario y contraseñas como texto plano a través de la red. Utilizando SSH se podría solucionar este aspecto de manera sencilla.
- POP3 (puerto 110): Es el candidato ideal, puesto que continuamente está enviando al servidor nuestro nombre de usuario y contraseña como texto plano para comprobar la existencia de nuevos mensajes.

6. Conclusiones

Los protocolos de la familia SSH, y sobre todo la segunda versión del mismo, permiten grandes posibilidades a la hora de conseguir conexiones seguras con ordenadores remotos, a través de redes no seguras.

El uso por parte de los usuarios no exige grandes conocimientos, y no dificulta la utilización de recursos remotos, ya que puede realizarse de manera totalmente transparente, por lo que el uso de los mismos debería extenderse, a nivel tanto de intranets como de extranets.

Una de las características más útiles, en mi opinión, es la capacidad de establecer conexiones seguras de protocolos no seguros. Esta capacidad nos permite continuar utilizando aplicaciones antiguas pero muy extendidas, sin comprometer la seguridad de los datos.

De todos modos, toda esta seguridad no sirve de nada en el caso de que los usuarios no sean conscientes de la necesidad de mantener la seguridad de los datos, puesto que si las contraseñas no son seguras, o el acceso a los ordenadores puede verse comprometido, permitiendo el acceso a las claves privadas, toda la seguridad permitida por SSH es totalmente inservible, puesto que una cadena es tan débil como el más débil de sus eslabones.

7. Referencias

- Garaizar Sagarminaga, Pablo. SSH Tunneling. http://130.206.100.150/docs/articulo.ssh.html, 2004
- Secure Shell FAQ, http://www.ayahuasca.net/ssh/, 2004
- Marín Illera, Álvaro. OpenSSH, http://www.e-ghost.deusto.es/docs/TutorialOpenSSH.html, 2004
- T. Ylonen. The SSH (Secure Shell) Remote Login Protocol, http://www.snailbook.com/docs/protocol-1.5.txt, 1995
- T. Ylonen. SSH Protocol Architecture, http://www.snailbook.com/docs/architecture.txt, 2003
- T. Ylonen. SSH Transport Layer Protocol, http://www.snailbook.com/docs/transport.txt, 2003
- T. Ylonen. SSH Authentication Protocol, http://www.snailbook.com/docs/userauth.txt, 2002
- T. Ylonen. SSH Connection Protocol, http://www.snailbook.com/docs/connection.txt, 2003
- SSH, http://es.wikipedia.org/wiki/SSH, 2004
- (Bitvise) SSH Port Forwarding, http://www.bitvise.com/port-forwarding.html, 2003