De viaje sin pasaporte

Fernando Cano

19 de marzo de 2018

En esta práctica nos familiarizaremos con las múltiples posibilidades del programa `ssh` (Secure Shell), que resuelve, en principio, muchos de los problemas de seguridad cuando nos conectamos a servicios remotos. Nuestro objetivo será conectarnos de forma segura y muy cómoda a diferentes hosts, entre ellos los de nuestros compañeros. Cuando decimos de forma segura nos referimos a que los datos que circulen entre los dos hosts lo hagan de forma cifrada. Y por otro lado cuando decimos de forma cómoda es porque vamos a evitar en lo posible andar introduciendo passwords para relizar las autenticaciones. Además de esto vamos a poder ejecutar el secure-copy *scp* casi como si fuera un simple *cp*. También vamos a poder ejecutar aplicaciones gráficas de forma remota y saltarnos cortafuegos y ...

Antes de nada crearemos un conjunto de cuentas en nuestra máquina para el resto de nuestros compañeros de prácticas. Los nombres de las cuentas los obtendremos de la base de datos que utilizamos en la práctica anterior. Y aprovecharemos para explicar, por encima, cómo es el almacenamiento de passwords en linux.

1. Creando usuarios linux

Partiendo de la plantilla, vamos a crear cuentas en nuestra máquina virtual para que todos los compañeros de nuestro grupo puedan conectarse a ella.

Para ello necesitamos disponer de todos sus UOs, cosa que realizaremos consultando una base de datos que contiene las listas de clase.

2. Generando el fichero con los UOs de nuestros compañeros

Para obtener los UOs de nuestros compañeros nos vamos a conectar a la base de datos ssi_bbdd.

Se supone que desde nuestro usuario *uo123456* tenemos permisos para hacer un select de la tabla de alumnos. En este momento ya podremos obtener los UOs de los alumnos del grupo PL-0X (el que corresponda) de esta asignatura.

Como ya has comprobado, el psql dispone de diversas utilidades para trabajar en modo texto. Consulta la ayuda y busca el modo de generar un fichero de texto de nombre *uo.txt* en el que cada línea contenga únicamente el UO de nuestros compañeros de grupo. Recuerda que los usuarios linux suelen ir en minúsculas.

Es importante que no incluya tu propio usuario, que ya tienes creado, pero que sí contenga el *uo123456*. Puedes realizarlo mediante una sola consulta que demuestre tus amplios conocimientos de SQL o si no te ves capaz, editando a mano el fichero *uo.txt*.

3. Creando cuentas en linux de forma masiva

Para crear nuestras cuentas en linux vamos a especificar para cada nuevo usuario su nombre (el uoxxxxxx correspondiente), un grupo (ssi20172018), una password (generada para cada usuario), la shell (bash) y el directorio home($\sim$), que en nuestro caso será */home/ssi20172018/uoxxxxxx*.

Cuando se crea ocasionalmente un usuario es cómodo utilizar el comando *adduser*, que nos pedirá algunos datos, de forma interactiva, para realizar la creación. En nuestro caso, para crear un conjunto de cuentas de forma masiva, en modo batch, vamos a utilizar el *useradd*.

En estos enlaces puedes consultar y ver algunos ejemplos sobre la gestión de usuarios y grupos:

- Manual Ubuntu

- Gestión de usuario y grupos en Ubuntu
- Ficheros password y shadow
- Más

Ahora tienes que crear un script de bash para crear una serie de usuarios partiendo de la siguiente plantilla que tienes en el fichero *plantilla_useradd1718* del directorio *useradd* del usuario alumnossi del host 156.35.163.123

Descarga el fichero, renómbralo como *useradd1718* y completa la orden **useradd**. Documéntalo de tal forma que te sea útil para el día del examen.

En la orden `useradd` el parámetro -m indica que se debe crear un directorio base para el usuario que estemos creando. Además permite copiar los ficheros que se encuentran en el directorio */etc/skel/* en el directorio del nuevo usuario.

Partiendo del propio *useradd1718*, crea el script *userdel1718*, para borrar los usuarios que hemos creado. Copia el bucle for del script anterior y mediante la orden *userdel* genera el script. El script eliminará el grupo y el directorio del mismo.

Para ver cómo funciona esto, crea algún fichero tipo README en it /etc/skel/ y verás que se copia al *home directory* de los nuevos usuarios.

Una vez creados los usuarios ya podrán hacer login en nuestra máquina. Para ello solo necesitan que les digas la clave que les ha sido asignada.

Sobre la generación de la password el siguiente texto obtenido del manual de PHP puede aclarar el funcionamiento de la función *crypt()* y los tipos de hash que se pueden utilizar:

- *CRYPT_STD_DES - Hash estándar basado en DES con un salt de dos caracteres del alfabeto "./0-9A-Za-z". Utilizar caracteres no válidos en el salt causará que crypt() falle.*
- *CRYPT_EXT_DES - Hash extendido basado en DES. El salt es un string de 9 caracteres que consiste en un guión bajo seguido de 4 bytes del conteo de iteraciones y 4 bytes del salt. Estos están codificados como caracteres imprimibles, 6 bits por carácter, por lo menos, el carácter significativo primero. Los valores del 0 al 63 son codificados como "./0-9A-Za-z". Utilizar caracteres no válidos en el salt causará que crypt() falle.*
- *CRYPT_MD5 - Hash MD5 con un salt de doce caracteres comenzando con 1*
- *CRYPT_BLOWFISH - Hash con Blowfish con un salt como sigue: "$2a$", "$2x$.º "$2y$", un parámetro de coste de dos dígitos, "$", y 22 caracteres del alfabeto "./0-9A-Za-z". Utilizar caracteres fuera de este rango en el salt causará que crypt() devuelva una cadena de longitud cero. El parámetro de coste de dos dígitos es el logaritmo en base 2 de la cuenta de la iteración del algoritmo hash basado en Blowfish subyacente, y debe estar en el rango 04-31; los valores fuera de este rango causarán que crypt() falle.*
- *CRYPT_SHA256 - Hash SHA-256 con un salt de dieciséis caracteres prefijado con 5. Si el string del salt inicia con 'rounds=<N>$', el valor numérico de N se utiliza para indicar cuántas veces el bucle del hash se debe ejecutar. El número de rondas por defecto es 5000, hay un mínimo de 1000 y un máximo de 999,999,999. Cualquier selección de N por fuera de este rango será truncada al límite más cercano.*
- *CRYPT_SHA512 - Hash SHA-512 con un salt de dieciséis caracteres prefijado con* 6*. Si el string del salt inicia con 'rounds=<N>$', el valor numérico de N se utiliza para indicar cuantas veces el bucle del hash se debe ejecutar. El número de rondas por defecto es 5000, hay un mínimo de 1000 y un máximo de 999,999,999. Cualquier selección de N por fuera de este rango será truncada al límite más cercano.*

4. Autenticación

SSH permite autenticarse con diferentes modelos, entre ellos vamos a trabajar con los basados en contraseñas y en criptografía de clave pública. Ya hemos utilizado el primer método, ahora para usar el de clave pública necesitaremos generar un par de claves.

Ejercicio 1 *Desde nuestra cuenta de usuario* `uo123456` *(el que proceda en cada caso), generaremos un par de claves con el comando* `ssh-keygen`*. Necesitarás especificar qué tipo de clave quieres con la opción* `-t`*. Usaremos una clave* `rsa`*.*

```
uo123456@us123456:~$ ssh-keygen -t rsa
```

Pregunta dónde quieres guardar las claves; deja las opciones por defecto que te ofrece.

Ejercicio 2 *Una vez generado tu par de claves, echa un vistazo a los ficheros que las contienen, que serán* `.ssh/id_rsa` *y* `.ssh/id_rsa.pub`*.*

La autenticación `ssh` por clave pública requiere que el lado remoto que te autentifica conozca tu clave pública. Por lo tanto, tendremos que exportar la información de `.ssh/id_rsa.pub` a los hosts donde queramos ser reconocidos. En nuestro caso será la máquina de uno de nuestros compañeros (por ejemplo el host 192.168.61.XX) en la cual tendremos habilitada una cuenta propia (uo123456) de la cual ya conoceremos la password, porque nuestro compañero nos los la ha transmitido por un medio seguro (boca-oreja).

Ejercicio 3 *Instala tu clave recién generada en el host remoto. Para ello, usamos el comando* `ssh-copy-id`*. Examina su página de manual con el comando* `man ssh-copy-id` *y efectúa la copia de la clave. El comando tendrá la forma*

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub uo123456@192.168.61.XX
```

Realmente lo hace el anterior comando es añadir nuestra clave pública a un fichero de claves autorizadas del usuario remoto. Este fichero de claves autorizadas es el *∼/.ssh/authorized_keys*. Esto significa que cualquiera que tenga posibilidad de añadir su clave pública a este fichero se podrá *loguear* como usuario remoto.

Otra forma de hacer esto mismo es añadiendo la clave pública en el fichero de claves /.ssh/authorized_keys mediante los siguientes comandos

```
$ scp ~/.ssh/id_rsa.pub uo123456@192.168.1.63:.ssh/
$ ssh uo123456@192.168.61.XX 'cat .ssh/id_rsa.pub>>.ssh/authorized_keys'
$ ssh uo123456@192.168.61.XX 'rm .ssh/id_rsa.pub'
```

Si observas estos dos últimos comandos entenderás porqué *ssh* es una secure-shell remota. En el segundo de ellos lo que se hace es que el usuario uo123456, ejecuta en el host remoto 192.168.61.XX el comando `cat .ssh/id_rsa.pub>>.ssh/authorized_keys`, es decir añadir el contenido del fichero *id_rsa.pub* al fichero *authorized_keys*. Además hay que tener en cuenta que toda esta información circula por la red lo hace de forma encriptada.

Ejercicio 4 *Comprueba que, en efecto, la clave ha sido enviada. Debería verse en el fichero* `.ssh/authorized_keys` *de tu cuenta en el host remoto.*

Si todo ha ido como debe, deberíamos ser capaces de hacer login remoto sin necesidad de usar nuestra contraseña. Naturalmente, tendremos que usar nuestra passphrase para liberar la password de la clave privada *id_rsa*, pero esa contraseña *solo* se utiliza localmente, *nunca* transita por la red.

5. Agentes

Tener que emplear nuestra clave privada cada vez que nos autenticamos con un host remoto implica usar su passphrase para desbloquearla. Esto es tedioso y poco conveniente si necesitamos abrir muchas sesiones. Es decir hemos conseguido un sistema que nos evita introducir una password para autenticarnos, pero ahora tenemos que introducir una passphrase para desbloquearla (no sería necesario si tuviéramos la clave privada desprotegida, cosa que no es recomendable). Por fortuna, existen programas (*agentes SSH*) en quienes podemos delegar nuestra autenticación mientras estemos en sesión. El agente permanece residente y se encarga de guardar una caché de claves privadas, de forma que, cuando necesites autenticarte, él mismo aporta la clave privada sin necesidad de que tú la desbloquees.

Dependiendo de si estamos trabajando en modo gráfico o en modo texto hay distintas soluciones. Ya que tenemos un entorno gráfico y nuestro usuario uo123456 es el propietario del entorno gráfico, este agente se ejecuta de forma automática y casi transparente para nosotros. La primera vez que se intente desbloquear la clave privada mediante la passphrase, se nos abrirá una ventana solicitando esa passphrase y preguntándonos si queremos que se guarde desbloqueada cada vez que iniciemos sesión. Esta es realmente la forma más cómoda de trabajar. La estudio de la utilización de agentes en modo texto lo dejaremos como algo opcional.

6. Ejecutando aplicaciones gráficas en un host remoto

Como `ssh` nos permite hacer tantas cosas, no iba a ser menos la posibilidad de ejecutar aplicaciones gráficas de forma remota y segura. Para hacer esto solo tenemos que añadir la opción **-X** a nuestro comando. Lo que realmente se consigue es que nuestro entorno gráfico local (X11) ejecute los gráficos de la aplicación remota. Evidentemente debemos ser los propietarios del display local.

Ejercicio 5 *Conéctate mediante `ssh` al el host de tu compañero con la opción **-X** y ejecuta una aplicación gráfica, por ejemplo zenmap.*

7. Putty

Ahora vamos a configurar Putty para establecer conexiones seguras.

Ejercicio 6 *Se trata de utilizar tu pareja de claves rsa creadas anteriormente. Para hacer esto necesitaremos realizar los siguientes pasos:*

1. *Convertir tu clave privada a un formato que entienda el Putty. Para ello necesitarás instalar el paquete putty-tools y ejecutar la orden `puttygen id_rsa -o id_rsa.ppk`*
2. *Copia el nuevo fichero para que Putty pueda utilizarlo para autenticarse.*
3. *Recuerda que tienes añadir tu clave pública a fichero authorized_keys para autorizar que tu cuenta linux permita conexiones desde hosts remotos, en este caso sería desde Windows.*
4. *Haz las operaciones necesarias para conectarte al host de tu compañero mediante clave pública.*
5. *Configura Putty para poder ejecutar aplicaciones gráficas. Para ello habilita el servidor X11 forwarding, define el display como localhost:0 y arranca el Xming. Suele resultar interesante habilitar la compresión del tráfico para el SSH.*
6. *Ahora ya puedes conectarte al ordenador de tu compañero y ejecutar cualquier aplicación gráficazenmap.*

8. Túneles

Por úĺtimo vamos a probar a establecer un túnel con SSH. Existen dos tipos de túneles, estáticos y dinámicos y dentro de los primeros los locales y los remotos, con importantes aplicaciones. La idea básica es utilizar una conexión segura entre un cliente y un servidor SSH para canalizar el tráfico de conexiones tcp. Alguna de sus aplicaciones es dar confidencialidad e integridad a conexiones que no contarían con ellas. Por otro lado permite saltarse ciertas restricciones impuestas por routers o cortafuegos.

Ejercicio 7 *Para realizar un sencilla prueba vamos a establecer un túnel estático local para conectarnos a nuestro servidor ritchie. Realiza los siguientes pasos:*

1. *Abre un terminal y ejecuta la siguiente orden:*
   `uo123456@us123456:~$ ssh -N -L 1234:localhost:5432 192.168.61.XXX`
   el parámetro -L indica que se trata de un túnel local, es decir un local port forwarding y lo que viene a establecer es que las peticiones al puerto 1234 de tu máquina se redireccionarán al puerto 5432 del host de tu compañero, es decir postgres. Con el -N se evita que se abra un sesión interactiva.
2. *Para el proceso con un Ctrl-Z y ejecuta el comando `bg` (background) para que se ejecute en un segundo plano.*
3. *Ejecuta la orden `psql -h localhost -p 1234 -U alumnossi ssi_bbdd` y deberías entrar conectarte a la base de datos ssi_bbdd del servidor postgres que está corriendo en el host 192.35.61.XXX. Realmente lo que estamos haciendo es establecer un canal seguro para una conexión a nuestro servidor de bases de datos mediante una conexión segura ssh.*
4. *Cierra la sesión con postgres y ejecuta el comando `fg` (foreground) para traer el proceso a un primer plano y poder pararlo con un Ctrl-C.*