WUOLAH

wiser

www.wuolah.com/student/wiser

161

SOLUCIÓN Práctica 2 - Sesión 1

SOLUCIÓN Práctica 2 - Sesión 1

3º Ingeniería de Servidores

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
UGR - Universidad de Granada

Ingeniería de Servidores
Práctica 2

Sesión 1

En esta práctica vamos a instalar el servicio `ssh` (secure shell) para la autenticación cliente - servidor y servidor - cliente. El que ejecuta el `sshd`, es el servidor, mientras que el que ejecuta el `ssh` es el cliente.

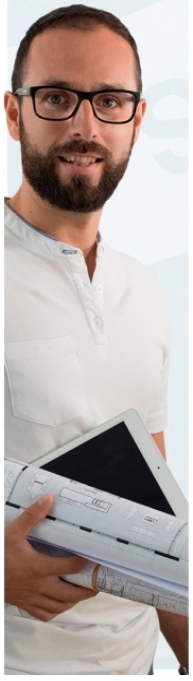
Antes de empezar la práctica, nos aseguraremos de que tal y como lo dejamos, Ubuntu tiene configurada la segunda tarjeta de red en host-only. Vamos a cargar la imagen de Ubuntu en la unidad óptica, pero indicaremos que no arrancaremos de allí; nos servirá para instalar paquetes adicionales a continuación. Hemos cambiado la IP del archivo `/etc/network/interfaces` de .105 a .101. También nos aseguraremos de que la red en CentOS está configurada de manera similar, tal y como hicimos en la sesión anterior.

1.- Usando Ubuntu como servidor y CentOS como cliente

Instalaremos en Ubuntu: `apt-get install openssh-server`.. Nos conectaremos con: `ssh <user>@192.168.56.101` al servidor. Nos avisará de que no se podría verificar que el servidor tenga una conexión segura, pues podrían darse ataques del tipo man-in-the-middle. Del lado del host, hay unos archivos `/etc/ssh/ssh_host_rsa_key.pub` y `/etc/ssh/ssh_host_rsa_key`. Cuando se establece la conexión, el servidor proporciona al cliente un fingerprint (una especie de hash, único), a partir del cual se puede obtener la clave pública, de modo que se pueda verificar la identidad del servidor; es decir: `fingerprint = hash[clave pública]` (podemos imaginar que resulta casi imposible sacar la clave pública a través del fingerprint).Una vez que se acepta la conexión, en `~/.ssh/known_hosts`, se guarda este registro, de modo que la próxima vez no habrá otro aviso.

Con `systemctl status sshd` podemos ver el estado del proceso que corre en el lado del servidor. Vemos que está escuchando en el **puerto 22**, que es el que está por defecto para las conexiones ssh. Podemos cambiar el puerto desde donde escucha `sshd` al **22022** Esto se puede hacer modificando el archivo `/etc/ssh/sshd_config`, o también se puede usar el programa **string editor** (`sed`; más concretamente `sed s/<expresión regular>/<texto a reemplazar>`): `sed s/'Port 22'/'Port 22022'/ -i /etc/ssh/sshd_config`. Ponemos esto a funcionar con `systemctl restart sshd`. En el cliente, nos conectaremos usando `ssh <user>@192.168.56.101 -p 22022`.

Para añadir más seguridad, en el servidor modificaremos el archivo `sshd_config` anteriormente mencionado, cambiando el flag `PermitRootLogin` a **no**. Esto evitará que con la cuenta de superusuario nos podamos conectar al servidor, de modo que haya que



conectarse con una cuenta normal y luego, si se desea, se tenga que cambiar a su. El hecho de necesitar dos contraseñas (la del usuario y la del root) complica la labor de un posible atacante.

No solamente podemos conectarnos con contraseña, sino que también usando la clave pública y privada del usuario, podríamos conectarnos al servidor. Vamos ahora a generar las claves pública y privada para darle la pública al servidor y la almacene. Usaremos `ssh-keygen`. Guardaremos la clave en el directorio que indica (`~/.ssh/id_rsa` (solo permisos de lectura y escritura para el usuario únicamente) y `~/.ssh/id_rsa.pub` lectura y escritura para el usuario y para los demás solo lectura). Elegiremos que no tenga contraseña. Cuando termina la generación, aparece una especie de imagen en el terminal para que nos sea más sencillo comparar las huellas en ambas partes (cliente y servidor). Ahora copiaremos la clave con `ssh-copy-id <user>@192.168.56.101 -p 22022`. Después de este proceso, en el servidor, justo en `~/.ssh/authorized_keys` se encontrará la clave pública del cliente que acabamos de copiar. Ahora, al recibir información, vendrá cifrada con nuestra clave pública y solo nosotros, con nuestra clave privada, podremos desencriptarla.

Es más, a partir de ahora podremos conectarnos al servidor sin necesidad de poner la clave. Para ello, también necesitaremos descomentar la línea del `/etc/ssh/sshd_config` y poner el `PasswordAuthentication` a **no**. El problema vendría si alguien se hace con nuestra clave privada. Nos ofrece el poder ponerle una contraseña, pero entonces nos la pediría al conectarnos (y esto es precisamente lo que queremos evitar). Nota: cuidado al conectarse al servidor, pues si estamos en root nos va a rechazar la conexión, ya que hemos ordenado en Ubuntu que no se acepten las conexiones del root.

Ubuntu tiene un cortafuegos llamado `ufw` (uncomplicated firewall). Con el comando `ufw status` vemos que está inactivo. Lo activamos con `ufw enable`. Tras ello, tendremos que añadir una excepción al puerto 22022 para que nos permita entrar, aunque el cortafuegos esté activo: `ufw allow 22022`.

2.- Usando CentOS como servidor y Ubuntu como cliente

En esta ocasión lo tenemos más fácil, pues ya tenemos `sshd` instalado. Directamente usamos `ssh <user>@192.168.56.110`. Tenemos que cambiar también la configuración de puerto con vi `/etc/ssh/ssh_config`, ponemos el puerto 22022 como antes, pero al restablecer el servicio sshd (`systemctl restart sshd`) nos devuelve un mensaje de error. Para resolver este inconveniente, necesitamos la ayuda del comando que la propia salida de error nos recomienda: `journalctl -xe`. Vemos que el proceso `polkitd` está causando esto; tenemos que darle permisos con `semanage`, que podemos instalar con `yum install policycoreutils-python`. Una vez listo, usamos `semanage port -l` para ver los puertos que hay, pero como hay tantos, obtendremos una pequeña ayuda de `grep: semanage port -l | grep 22` para localizar cuál es al que se le está limitando a solamente escuchar el puerto 22 (tenemos que indicar que sea el 22022 al que se escuche, tal y como queremos). Es `ssh_port_t`; por tanto, usaremos `semanage`

`port -a -t ssh_port_t -p tcp 22022`: donde **-a** significa añadir, **-t** especifica el tipo, y **tcp** es el tipo de puerto. Tras unos segundos, se cambiará la configuración. Comprobamos el funcionamiento con `restart` y `status` de `systemctl`.

No obstante, cuando intentamos conectarnos, no nos deja, porque por defecto en CentOS el cortafuegos está activo, así que tenemos que permitir las conexiones entrantes el puerto 22022 con `firewall-cmd --add-port 22022/tcp`. Sin embargo, esta orden sólo modifica las instancias en RAM, y la próxima vez que reiniciemos la máquina virtual se perderá esta configuración. Si añadimos `--permanent` también se hace un cambio en los ficheros de configuración del arranque, pero no lo hace en RAM, así que usaremos `firewall-cmd --reload` y se volverá a cargar la configuracićon actual (así ni tendremos que reiniciar la máquina, ni que ejecutar el comando sin `--permanent` otra vez). Cuidado, pues si no pusiéramos `--permanent` e hiciéramos tras ello `--reload`, se borraría la configuración, porque no se habría cargado en el archivo de configuración, al contrario de lo que se consigue con `--permanent`.

En resumen:

	Ubuntu Server 16	**CentOS 7**
`ssh daemon`	Instalación manual	Por defecto
`systemctl status`	`ssh / sshd`	`ssh`
`sshd_config`	línea de especificación del puerto sin comentar	línea de especificación del puerto comentada
Cortafuegos	`ufw` (uncomplicated firewall), desactivado por defecto	`firewall-cmd`, activado por defecto
Módulo SELinux	no instalado	instalado
Daemons	preparados (loaded) y activos (enabled)	parados y desactivados por defecto

Los **rootKit** son una especie de puerta trasera que puede ser tomada como malware. Hay softwares específicos que pueden detectar este tipo de software, como **`rkhunter`** (rootKit hunter) Con la opción `-c` hará un chequeo. Lo instalaremos en Ubuntu con `apt install rkhunter`. Si da error por no haber encontrado el cdrom, usaremos `mount /dev/sr0 /media/cdrom/`. Después, elegiremos “sin configuración” cuando nos pregunte. Pasaremos un escáner con `rkhunter -c` como hemos dicho.

El servicio **fail2ban** administra los fallos de conexión que ha podido haber y que han sido previamente almacenados en los logs y los lleva al cortafuegos, que con las tablas de IP y unas reglas, impiden a ciertas direcciones IP seguir intentando conectarse, llevadas hasta aquí por fail2ban, que indica cuándo sucede esto. Para la instalación tenemos que añadir el repositorio con `yum install epel-release` y, acto seguido, `yum install`

`fail2ban-all.noarch`. Al igual que hemos estado haciendo hasta ahora, podemos comprobar el estado de este servicio (daemon) con `systemctl status fail2ban`; al principio veremos que está parado y desactivado. Lo activamos con `systemctl start fail2ban`. Al ver su estado de nuevo, veremos que está activo **pero deshabilitado**, así que usaremos `systemctl enable fail2ban`. Con `fail2ban-client status` podemos ver las estadísticas según las acciones que ha llevado a cabo, las cuales pueden especificarse modificando los archivos en `/etc/fail2ban`.