Qué

Se ha de instalar y configurar un servidor SSH en Ubuntu 20.04 y justificar por qué se ha elegido sobre otras opciones. Además, hay que configurarlo para permitir acceso a él mediante un par de claves y que, en caso de que no exista la clave pública del cliente en el servidor o no sea correcta que se de un intento para introducir la contraseña.

Para qué

El objetivo de esta memoria es aprender a configurar un servidor SSH y cambiar su configuración de modo que los clientes puedan acceder de forma segura mediante su propio par de claves si se ha copiado su clave pública en el servidor para tener un servidor SSH que sea tanto seguro como conveniente.

Cómo

Instalación

El primer paso es instalar el servidor SSH, en este caso OpenSSH. Elijo este porque es el servidor SSH de código abierto más utilizado. Esto tiene dos implicaciones: Primero, hay más documentación escrita para él que para otras alternativas, lo que facilita encontrar información. Segundo, como es de código abierto y tiene mucha gente usándolo, hay mucha gente encontrando bugs y problemas y, sobretodo, aportando formas de arreglarlos.

Para instalar el el servidor OpenSSH, se ha de ejecutar:

sudo apt install openssh-server

Normalmente también se instala el cliente SSH de OpenSSH, para poder probar el funcionamiento de éste servidor:

sudo apt install openssh-client

Ahora toca comprobar que el servicio esté habilitado. Para ello uso systemctl de esta forma:

systemctl status sshd

Hay que fijarse en dos cosas: Primero, si está habilitado y si esta activo. Deberían estarlo por defecto al instalarlo, pero por si acaso no se da alguno de los casos anteriores, se debe iniciar el servicio para que funcione en la sesión actual y habilitar el servicio para que se inicie automáticamente al encender el servidor. Para ello se ejecutarían estos dos comandos, respectivamente:

sudo systemctl start sshd

sudo systemctl enable sshd

Ahora, para poder restaurar la configuración por defecto en caso de que algo salga mal, hago una copia del archivo de configuración de openssh-server, /etc/sshd_config:

sudo cp /etc/ssh/sshd_config{,.bck}

Por último, se debería hacer una copia de hacer una copia de seguridad de las claves públicas y privadas del servidor, para evitar tener que volver a configurar todos los clientes a los que se va a conectar el servidor:

*sudo cp /etc/ssh/ssh_host*key* <ruta_destino>*

Donde *<ruta_destino>* debería ser la ruta a un dispositivo de almacenamiento externo en el que hacer la copia. Es importante saber que al copiarla de nuevo a un servidor, se le debería dar permisos de nuevo, ya que la mayoría de sistemas de archivos que usan los dispositivos de almacenamiento externos no conservan los permisos. Para restaurarlos, se deberían ejecutar estos tres comandos, desde el directorio /etc/ssh:

*sudo chown root:root ssh_host*key**
*sudo chmod 644 ssh_host*key*.pub*
*sudo chmod 600 ssh_host*key*

Se hace de esta forma para evitar que las claves privadas (las que no están sucedidas por '.pub') sean visibles por alguien que no sea su dueño, root.

Configurar el servidor para acceder por par de claves

Para permitir el acceso al servidor usando usando un par de clave en vez de contraseñas, se ha de comprobar que la opción '*PubkeyAuthentication'* de */etc/sshd_config*, el archivo de configuración de openssh-server, esté comentada o tenga como valor *yes*, que es el valor por defecto. También hay que prestarle atención a la opción AuthorizedKeysFile, que define el archivo en el que estarán guardadas las claves públicas de los clientes en los usuarios del servidor. Por defecto hay dos archivos, yo sólo he dejado el primero, '.ssh/authorized_keys'.

Tras modificar el archivo de configuración se tiene que reiniciar el servicio para aplicar los cambios:

sudo systemctl restart sshd

Para que el cliente pueda acceder al servidor usando autentificación por par de claves, primero tiene que generarlas con este comando:

ssh-keygen -t rsa

Es un comando interactivo que pedirá la ruta de destino del par de claves y la contraseña de la clave. Por defecto las claves, si se elige el algoritmo RSA, serán *~/.ssh/id_rsa* (privada) y *~/.ssh/id_rsa.pub* (pública).

Tras eso, se tiene que añadir la clave pública del cliente a '.ssh/authorized_keys' en uno de los usuarios del servidor. Se puede hacer de dos formas:

Sobreescribiendo 'authorized_keys' con la clave pública del cliente:
scp .ssh/ usuario@servidor:~/.ssh/authorized_keys

Añadiendo la clave pública a una nueva línea en authorized_keys con ssh-copy-id (opción recomendada y la que he usado yo):
ssh-copy-id usuario@servidor

En ambos casos se habrá de escribir la contraseña del usuario al que se intenta conectar.

Ahora el cliente puede acceder usando ‘ssh usuario@servidor’ y no se le pedirá contraseña, se iniciará sesión automáticamente.

Configurar el servidor para que si la clave del cliente no existe en el servidor o si es incorrecta, el usuario pueda poner la clave una sola vez.

Primero se comprueba que la opción ‘PasswordAuthentication’ este comentada o en *yes* dentro del fichero de configuración de sshd.

Después se limita el número intentos de acceso por contraseña posibles a uno escribiendo ‘*MaxAuthTries 2*’ en /etc/sshd_config y se reinicia el servicio sshd. MaxAuthTries no define una cantidad máxima de intentos por contraseña (esa opción no existe, no está siquiera en el manual de sshd_config), sino un máximo de intentos de autentificación de cualquier tipo. Se pone el número de intentos en 2 porque primero intenta acceder por clave, y si funciona el usuario puede acceder, pero si falla resta un intento.

Con esto, si existe un par de claves en el cliente que está obsoleto, se ha cambiado o no está registrado en el servidor, falla la autentificación por clave, lo cual ya cuenta como un intento, dejando un intento restante.

El problema es que esto no tiene en cuenta dos casos: El cliente puede tener 0 pares de claves o puede tener múltiples pares no registrados en el servidor.

En caso de que tenga 0 pares de claves, el comando ssh salta directamente a autentificarse por contraseña al servidor por lo que aún tendremos dos intentos para poner la contraseña.

En caso de que el usuario tenga múltiples pares de claves no válidos, si el comando ssh trata de autentificarse con dos de ellos antes que con un par de claves que si que esté registrado en el servidor, se alcanzará el máximo de intentos de autentificación y se saldrá del comando con un error.

La RFC de SSH-USERAUTH, uno de los tres componentes principales listados en la introducción de la RFC de la Arquitectura del protocolo SSH, menciona lo siguiente en el punto 4:

> *El servidor dirige la autentificación diciéndole al cliente qué métodos de autentificación puede usar para continuar el intercambio en un momento dado. El cliente tiene la libertad de elegir cualquiera de los métodos listados por el servidor en cualquier orden. Esto le da al servidor control completo de proceso de autentificación si lo desea, pero también le da al cliente suficiente flexibilidad como para elegir los métodos más convenientes para el usuario cuando múltiples métodos son ofrecidos por el servidor.*

Es decir, en la forma en la que esta diseñado el protocolo de autentificación de SSH, el servidor es libre de restringir las formas de autentificarse y el máximo de intentos, pero el cliente es libre de elegir cómo quiere autentificarse de entre los métodos disponibles y qué pares de claves quiere usar.

Por tanto, la responsabilidad en caso de que se deniegue el acceso por tener demasiadas claves invalidas es del usuario, que deberá cambiar su configuración o usar el argumento ‘-o PubkeyAuthentication=no’, entre otras opciones.

Además, el punto 7 del mismo RFC menciona que solo se puede responder a peticiones de autentificación por clave pública mediante los mensaje SSH_MSG_USERAUTH_FAILURE (error de autentificación), el mensaje SSH_MSG_USERAUTH_PK_OK (la clave es correcta) o, en casos más raros, con el algoritmo o *blob* de la clave publica de la petición de autentificación, lo cual indica que, por diseño, los pares de claves incorrectos que se traten de usar serán contados como fallos de autentificación y reducirán los intentos restantes.

En resumen, dejo 'MaxAuthTries' en 2 porque a pesar de que no funciona como especifica la práctica, es lo más similar dentro de las limitaciones que existen por cómo está diseñado el protocolo de autentificación de SSH.

Bibliografía

https://www.rfc-editor.org/rfc/rfc4251

https://www.rfc-editor.org/rfc/rfc4252

https://manpages.ubuntu.com/manpages/jammy/en/man5/sshd_config.5.html

https://manpages.ubuntu.com/manpages/jammy/en/man8/sshd.8.html

https://unix.stackexchange.com/questions/574366/client-still-can-ssh-login-without-password-on-server-after-generating-new-keys

Configurar el servidor para que, al borrar la clave en el cliente, no pueda entrar sin contraseña

El caso de que pueda entrar sin contraseña al borrar la clave sólo se da bajo unas circunstancias específicas. Si el cliente tiene un par de claves que está registrado en el servidor y se conecta a él a través de ssh, posteriormente cierra la conexión y, finalmente, borra su par de claves, podrá seguir entrando al servidor sin que éste le pida contraseña hasta que el usuario cierre su sesión.

La cosa está en que esto no tiene que ver con el servidor, y difícilmente se puede considerar un fallo de seguridad.

Lo que está sucediendo aquí realmente es que tras iniciar una conexión ssh usando un par de claves o, más específicamente, al usar un par de claves por primera vez en la sesión, una copia de la clave privada se guarda en RAM en forma de proceso en el propio cliente por el Identity Agent de OpenSSH, ssh-agent.

La utilidad de ssh-agent es no tener que introducir contraseñas de claves privadas cada vez que se usan y no indicar constantemente dónde está ubicado un par de claves en caso de que esté en un directorio fuera de lo normal, pues se guarda una copia de la clave privada en RAM que no tiene la contraseña y no depende de una ruta concreta. Además, estas copias no se pueden sacar o ver de ningún modo, así que se pueden tener pares de claves que no estén guardadas en disco, para una mayor seguridad. Aunque las claves guardadas en la RAM no se puedan leer, se puede usar el ssh-agent para generar un hash de la clave para comprobar si es correcta, para generar la clave pública correspondiente e incluso para encriptar y desencriptar mensajes.

Ssh-agent no actúa automáticamente de por sí, se tiene que iniciar el servicio con 'eval `ssh-agent`' y se tienen que añadir claves con ssh-add. Pero aquí es donde entra en juego la otra pieza de este

rompecabezas, gnome-keyring, también conocido como el gestor de claves y contraseñas de Gnome (el entorno de escritorio de Ubuntu) y su servicio, gnome-keyring-daemon.

Este servicio se encarga de mostrar automáticamente un prompt gráfico para que el usuario introduzca contraseñas de pares de claves en el momento de usarlas en caso de que la tengan y, algo crucial para este caso, de guardar copias de estas claves usando ssh-agent automáticamente al usarlas por primera vez en la sesión. Es por esto por lo que una de las condiciones para que el cliente pueda entrar sin contraseña al servidor sin borrar su clave sea haber iniciado una conexión ssh en esa sesión, porque al hacerlo se esta usando el par de claves, y al usar el par de claves gnome-keyring-daemon lo guarda en RAM usando ssh-agent.

Ahora bien, para saber por qué se conecta al servidor sin tener una clave física primero se tiene que tener en cuenta como funciona la autentificación por par de claves.

Esto es lo que tiene que decir el segundo párrafo del punto 7 de la RFC de SSH-USERAUTH:

> *Con este método [Autentificación por clave pública] la posesión de una clave privada sirve como autentificación. Este método funciona mandando una firma creada con la clave privada del usuario. El servidor TIENE que comprobar que la firma es un autentificador válido para el usuario, y TIENE que comprobar que la firma es válida. Si ambos casos son ciertos, la petición de autentificación TIENE que ser aceptada, de otro modo, TIENE que ser rechazada. Nótese que el servidor PUEDE requerir autentificaciones adicionales tras una autentificación exitosa.*

La forma que tiene el servidor de validar la firma es con la clave pública que tiene almacenada del par de claves del cliente. Con la configuración por defecto, el comando ssh usa ssh-agent (recordemos que es una parte importante de OpenSSH) para enviar la firma usando la clave privada que hay almacenada en RAM, Esta clave es idéntica a la que estaba guardada, por lo que cumple con ambos requisitos igual que lo haría esa: Es un identificador válido para el usuario y la firma se puede desencriptar con la clave pública correspondiente que está en el servidor, por tanto TIENE que aceptarla, si no estaría contradiciendo directamente las RFCs.

Bibliografía:

https://www.rfc-editor.org/rfc/rfc4252#section-7

https://smallstep.com/blog/ssh-agent-explained/

https://manpages.ubuntu.com/manpages/jammy/en/man1/ssh-agent.1.html

https://manpages.ubuntu.com/manpages/jammy/en/man1/ssh-add.1.html

https://docstore.mik.ua/orelly/networking_2ndEd/ssh/ch02_05.htm

Conclusión

Durante esta práctica he aprendido mucho sobre el funcionamiento interno de SSH, pero no tengo claro si es porque la práctica está muy bien pensada o muy mal explicada.

He tenido que leer toneladas de documentación. Eso es didáctico, sí, pero he tenido que hacerlo porque las preguntas no estaban claras y en clase las dudas no se han aclarado.

Profundizando más en el tema de las preguntas, son demasiado escuetas, obtusas y ambiguas. Por ejemplo, la 3 dice:

> *Si no existe o es incorrecta que permita el acceso por contraseña con un único intento*

Pero no se deja claro dónde no tiene que existir, si en el cliente o en el servidor. Eso cambia completamente el foco de la pregunta, y como no se cuál es correcta, tengo que tener en cuenta ambos casos a la hora de redactar y lo que escribo acaba siendo pesado de leer y un tanto redundante.

Por otro lado, la pregunta 4 se ha discutido en clase, y se ha enmarcado como ‘Un problema de openssh-server en Ubuntu’, pero el servidor no tiene que ver, esa forma de funcionar es intencional así que no es realmente un problema, y ni siquiera es cosa de Ubuntu, sino cosa de Gnome (Cualquier distribución con Gnome tiene el mismo problema y Ubuntu sin Gnome no lo tiene, lo he probado), y al final todo se vuelve confuso.

Dicho todo esto, realmente he aprendido un montón, y me encanta aprender, así que aunque mientras escribo estos párrafos mi impresión de esta práctica no es muy positiva, pasados unos días la recordaré con mejores ojos viendo todo lo que he aprendido.