

CONTENIDO

| | | |
|---|----|----|
| 1 | 4 | |
| 2 | 4 | |
| 3 | 4 | |
| 3.1 | 4 | |
| 3.2 | 5 | |
| 3.2.1 | 6 | |
| 3.2.2 | 6 | |
| 3.2.3 | 7 | |
| 3.3 | 8 | |
| 3.3.1 | 9 | |
| 3.3.2 | 9 | |
| 3.3.3 | 9 | |
| 3.3.4 | 9 | |
| 3.4 | 9 | |
| 3.4.1 | 9 | |
| 3.4.2 | 9 | |
| 4 | 11 | |
| 4.1 | 12 | |
| 5 | 14 | |
| 5.1 | 14 | |
| 5.2 | 16 | |
| 5.3 | 17 | |
| 5.4 | 21 | |
| 5.5 | 23 | |
| 5.6 | 25 | |
| 5.7 | 26 | |
| 6 | 28 | |
| 6.1 | 28 | |
| 6.2 | 31 | |
| 7 | 33 | |
| 8 | 35 | |
| 9 | 36 | |
| Anexo I – Activar el cliente OpenSSH nativo en Windows 10 | | 36 |
| Anexo II – Habilitar SSH-Agent en Windows 10 | | 37 |
| Anexo III – SSH utilizando Java | | 37 |

| | |
|--|----|
| Anexo IV – Transferencia de ficheros mediante SFTP en Java | 39 |
| Anexo V – Instalación de 2FA para SSH | 40 |
| Anexo VI – Instalación y uso de Nmap | 44 |

1 OBJETIVOS

- Conocer qué es SSH
- Entender cómo funciona SSH y los principios criptográficos sobre los que se desarrolla
- Saber utilizar las funcionalidades principales de SSH
- Aprender a solventar cualquier problema que surja cuando se utilice esta herramienta
- Comprender y saber modificar la configuración del cliente y servidor
- Revisar algunas funcionalidades avanzadas de SSH

2 CASOS PRÁCTICOS

En el presente documento se explicarán ejemplos prácticos concretos sobre los siguientes temas:

- Uso básico de SSH
- Generación y gestión de pares de claves
- Acceso remoto con clave pública
- Uso de frase de paso (*passphrase*) para proteger una clave privada
- Uso de *SSH-Agent*
- Configuración del cliente y servidor SSH para múltiples entornos
- Uso de la funcionalidad de *forwarding*
- Túneles SSH

3 INTRODUCCIÓN

En este apartado se va a explicar brevemente algunos conceptos básicos que todo aquel que quiera utilizar SSH debería saber. No es requisito indispensable para poder utilizar la herramienta, pero nunca está de más saber el origen, las bases teóricas y cómo funcionan internamente las herramientas que utilizamos en Informática. Esta revisión nos proporcionará un punto de partida muy bueno para, en los apartados siguientes, empezar a utilizar y dominar SSH.

3.1 ¿QUÉ ES SSH?

Los orígenes de SSH provienen de otras aplicaciones anteriores, en concreto **Telnet**, **RLogin** y **RSH**. Estas herramientas concretamente permitían **abrir una Shell de forma remota**, es decir la conexión remota a otro equipo y manejarlo desde línea de comandos. La más importante de todas ellas fue Telnet, la cual se desarrolló en el año 1969 y fue fundamental en el crecimiento de la utilización de redes de ordenadores. Estos protocolos tenían dos inconvenientes importantes desde el punto de vista de la seguridad:

- **La conexión no era cifrada**, lo que significa que cualquier persona que tuviera acceso a la conexión entre dos máquinas podría ver qué se está haciendo con este equipo remoto.
- **La autenticación tampoco era cifrada**, es decir los datos de acceso al equipo remoto (típicamente usuario y contraseña) se enviaban en texto claro, sin cifrar. Esto significa que cualquier atacante que pueda tener acceso a la conexión (lo que se conoce como *sniffing* de la red) podría obtener las credenciales para acceder al equipo que se quiere gestionar de forma remota.

Con lo cual estos problemas ocasionan que el uso de estas herramientas sólo sea viable en entornos relativamente seguros (redes privadas, debidamente fortificadas y sin acceso a Internet). Estas herramientas fueron diseñadas de esta forma porque en los años en los que surgieron **la preocupación por la seguridad en las redes de ordenadores no era la misma que existe actualmente**, la cual ha ido incrementando conforme a las necesidades de los usuarios de Internet.

La solución a estos problemas por tanto era obvia, **cifrar por completo tanto el proceso de comunicación como el de autenticación**. De esta forma se garantiza que al autenticarnos en un equipo remoto nadie va a tener acceso a las credenciales que utilizemos para hacerlo, y además tampoco nadie en ningún momento va a poder ver lo que se está haciendo en la conexión. De esta forma **surge la herramienta SSH**, que nace para cubrir las necesidades de seguridad que hemos comentado.

SSH fue creada en **1995** por **Tatu Ylonen** quien fundó una empresa llamada SSH Communications Security (ssh.com). Sin embargo, cuando realmente tiene éxito SSH es cuando se desarrolla en 1999 la versión open source con prácticamente la misma funcionalidad que dicho software denominada **OpenSSH**. Supuso una revolución en el uso y administración de sistemas remotos por eliminar estas carencias de seguridad tan importantes. A partir de ahora en este documento siempre que hablemos de SSH nos estaremos refiriendo directamente a OpenSSH, que es con diferencia la implantación más extendida de esta herramienta.

OpenSSH está desarrollado por la comunidad OpenBSD¹. Fue desarrollado en el lenguaje C bajo licencia BSD de dominio público. Se apoya en la librería LibreSSL (Fork de OpenSSL tras el escándalo de la vulnerabilidad *Heartbleed* de dicha herramienta). Es utilizado ampliamente en BSD, GNU/Linux y UNIX.

Los componentes principales de OpenSSH son:

- *ssh*
- *scp*
- *sftp*
- *ssh-keygen*
- *ssh-agent*
- *sshd*
- *ssh-keyscan*

Versiones:

- **SSH-1**
- **SSH-2** incompatible con SSH1 (tanto cliente como servidor deben usar SSH-2). Incluye mejoras importantes. Actualmente ha reemplazado totalmente a SSH-1.
 - *Diffie-Hellman* para intercambiar claves
 - Verificación de integridad mediante *Message Authentication Code* (MAC)
 - Múltiples sesiones en una conexión (por defecto 10)

3.2 FUNDAMENTOS DE CRIPTOGRAFÍA

En esta sección vamos a ver una breve introducción a la criptografía para que algunos conceptos que vamos a usar en el resto del documento no sean desconocidos para aquellos lectores que no conozcan las bases de la técnica de la criptografía sobre la cual se asienta SSH como muchas otras herramientas que buscan proteger la confidencialidad y la integridad en las Tecnologías de la Información y la Comunicación.

La **criptografía** se puede definir como “*Ciencia que hace uso de métodos y herramientas matemáticas con el objetivo principal de **cifrar**, y por tanto proteger, un mensaje o archivo por medio de un algoritmo,*

¹ OpenBSD es un proyecto que sustenta un sistema operativo basado en BSD (tipo Unix) que desarrolla software caracterizado por ser abierto y muy robusto en cuanto a su seguridad.

usando para ello una o más **claves**², con lo que se logra en algunos casos la confidencialidad, en otros la autenticidad, o bien ambas simultáneamente”. En el contexto de las Tecnologías de la Información y la Comunicación (TIC), nos permite ocultar cierta información para mantenerla segura y mediante una serie de mecanismos proceder en cualquier momento a recuperar el mensaje original.

3.2.1 CRIPTOGRAFÍA DE CLAVE SIMÉTRICA

Pertenecen a este tipo de criptografía todas aquellas técnicas que nos **permitan cifrar y descifrar un mensaje mediante el uso de una única clave secreta**, que será la misma que se utilice tanto para cifrar como para descifrar la información. Es una técnica muy utilizada hoy en día, pero presenta una limitación importante: cuando queremos utilizarlo en redes de ordenadores, si queremos cifrar la comunicación entre nosotros y un receptor remoto mediante criptografía simétrica no disponemos, a priori, de un mecanismo para mandarle la clave de descifrado de una forma segura ya que si la mandamos por la red cualquier atacante que tenga acceso a la conexión entre ambas máquinas podría interceptar la clave (esta limitación se conoce habitualmente como **problema de distribución de claves**). Esto sin embargo no sería problema si lo que queremos es cifrar algo en un equipo, pero no deseamos comunicarlo a otro ordenador, sino que nosotros mismos nos encargáramos posteriormente de descifrarlo (por ejemplo, para guardar de forma segura documentos mientras no estamos presentes), o siempre y cuando podamos establecer una comunicación segura previa para intercambiar esta clave con el destinatario.

Algoritmos que utilizan criptografía simétrica:

- DES
- 3DES
- IDEA
- Blowfish
- CAST5
- AES (Rijndael) □ El que más se utiliza hoy en día

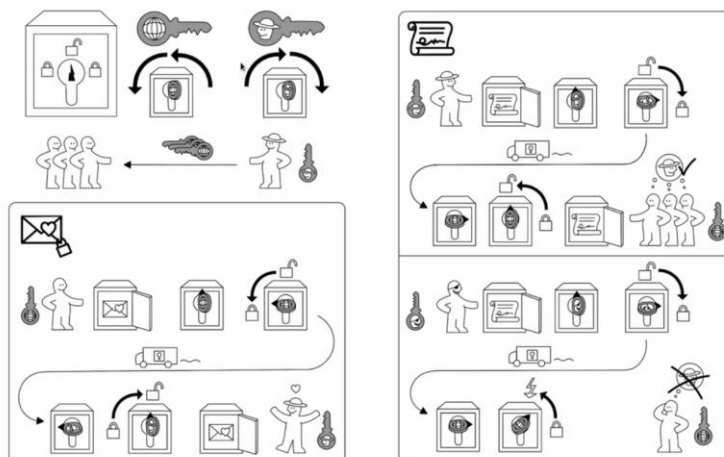
3.2.2 CRIPTOGRAFÍA DE CLAVE ASIMÉTRICA

La criptografía de clave asimétrica o de clave pública **utiliza un par de claves (pública/privada)** que cuyo su nombre indica una de ellas **se puede distribuir de forma pública** para que cualquiera la pueda utilizar mientras que la denominada clave privada **se mantendrá en secreto** por el usuario que origina el mensaje. De esta forma si un usuario (emisor) envía un mensaje cifrado a otro (receptor) con su clave privada, cualquiera que disponga de su clave pública podrá descifrar el mensaje, y al contrario. Si se cifra un mensaje con una clave pública sólo el usuario que disponga la clave privada asociada a dicha clave pública podrá descifrar el contenido del mensaje. Estos dos usos permiten la utilización de técnicas de criptografía asimétrica en múltiples tecnologías en Internet hoy en día para, fundamentalmente, cifrar las comunicaciones y garantizar que un usuario fue el creador de un mensaje (lo que se suele denominar en seguridad informática como “no-repudio”).

² En castellano cifrar y encriptar son sinónimos, por lo que se puede encontrar en las fuentes bibliográficas en ambas formas, lo mismo ocurre con descifrar y desencriptar. En este documento se va a utilizar por norma general los términos cifrar/descifrar.

PUBLIC KEY KRÜPTO

idea-instructions.com/public-key/
v1.0, CC BY-NC-SA 4.0 **IDEA**



Esquema conceptual (tipo IKEA) del funcionamiento de la criptografía de clave pública

La limitación de estas técnicas es **la complejidad de los algoritmos**, lo que ocasiona que sea mucho más costoso computacionalmente en comparación con las técnicas de criptografía simétrica:

- Factorización de números primos
 - RSA
 - DSA
- Curvas elípticas (para garantizar el mismo nivel de fiabilidad computacionalmente es menos costoso)
 - ECDSA
 - Ed25519

3.2.3 FUNCIONES HASH

Las **funciones hash** o **funciones resumen** son un tipo de operaciones matemáticas cuya principal característica es que son **unidireccionales**, es decir cuando se le pasa una función hash a un mensaje determinado termina produciendo un mensaje, siempre del mismo tamaño, y a partir del cual no puede obtenerse (podríamos decir descifrarse) el mensaje que lo ha originado. Podríamos decir que es una técnica de cifrado cuyos resultados no pueden ser descifrados, es decir **no existe una función inversa que nos devuelva el mensaje original**. La utilidad de este tipo de técnicas es la de garantizar la **integridad** de los mensajes, es decir verificar si unos datos son correctos o se han manipulado de alguna forma desde que se originaron.

Por ejemplo, si un usuario envía un mensaje con su hash correspondiente, el receptor podrá generar el hash con el mismo método o función de hash que se haya utilizado en el origen y deberá obtener el mismo mensaje que le haya llegado, de lo contrario puede concluirse que el mensaje se ha modificado durante la comunicación.

Algoritmos de hash habituales:

- CRC
- MD5 □ Considerado hoy en día no seguro, por lo que no se utiliza.
- Whirpool
- SHA-1
- SHA-256

- SHA-512
- SHA-3

A día de hoy las funciones **SHA-1**, **SHA-256** y **SHA-512** son los que se suelen utilizar, aunque está en desarrollo un futuro SHA-3.

3.3 FORMAS DE AUTENTICACIÓN EN SSH

3.3.1 CONTRASEÑA

El método básico para autenticarnos en un equipo remoto mediante una conexión SSH es la de **introducir manualmente el usuario y la contraseña**. Es el método más común que todo el mundo suele conocer, por su sencillez, pero a la hora de implementar tareas automatizables resulta ineficaz y hace necesario utilizar otros métodos de autenticación.

3.3.2 CLAVE PÚBLICA

Posiblemente sea el método más natural de utilizar SSH. Mediante este mecanismo el usuario generará un par de claves pública/privada. La privada se mantiene en secreto y la pública se almacena en el equipo remoto al que se quiere acceder. Después el usuario mediante SSH realiza la conexión, y **se verificará mediante la clave privada que es el propietario legítimo de acceder, por lo que no se necesitará introducir contraseña alguna**. Esto permite conectarse y administrar equipos remotos de una forma más ágil, rápida, segura e incluso permite la automatización de conexión entre equipos pues no requiere introducir ningún dato adicional para establecer la conexión.

3.3.3 KERBEROS

Este método es menos conocido y específico para entornos corporativos. Para llevar a cabo la autenticación por este método se necesita la intermediación de un servidor Kerberos configurado. No se verá con detalle en este documento.

3.3.4 GSSAPI

Son bibliotecas que permiten configurar un método de configuración personalizado a cualquier desarrollador que lo necesite. Tampoco se revisará este tipo de autenticación en este documento.

3.4 CÓMO FUNCIONA SSH

En esta sección, brevemente y sin mucho detalle, se va a explicar cómo se establece una conexión entre dos máquinas (cliente y servidor) mediante SSH. Este proceso habitualmente se divide en dos fases diferenciadas denominadas **negociación y autenticación**.

3.4.1 FASE 1: NEGOCIACIÓN

En esta primera fase el cliente establece la conexión con el servidor, respondiéndole este con la **versión de SSH que está ejecutando, junto con información adicional** sobre qué algoritmos específicamente deben utilizar para poder entenderse entre ellos.

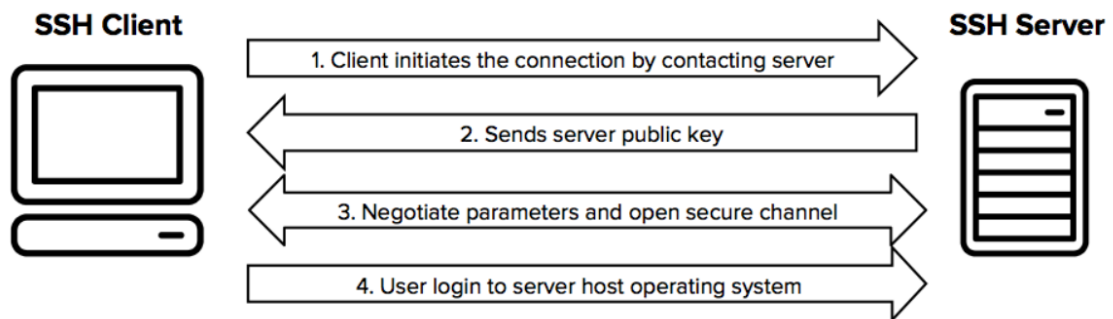
Acto seguido, **el servidor envía su clave pública al cliente**, quien se encargará en este caso de verificar si dispone de dicha clave entre las que tiene guardadas. Cuando se lleva a cabo esta verificación **se procede a negociar** (de ahí el nombre de esta fase) **qué algoritmos y semilla van a utilizar para el cifrado** de la comunicación. Esta negociación ya se produce de forma cifrada utilizando la clave pública del servidor.

A partir del algoritmo y la semilla escogida, ambos interlocutores (cliente y servidor) **generan a la vez las claves de sesión y la intercambian** para verificar que la otra parte ha obtenido el mismo resultado. A partir de este momento comienza lo que es la conexión en sí.

3.4.2 FASE 2: AUTENTICACIÓN

A partir del punto en el que se ha establecido una clave de sesión entre cliente y servidor llega el momento de que **el cliente se autentique** (inicie sesión) en el servidor SSH. Para esto, el servidor en primer lugar le ofrece al cliente los **métodos de autenticación** de los que dispone, pudiendo elegir el cliente cualquiera

de ellos para llevar a cabo su autenticación. Si el cliente logra realizar la autenticación de forma satisfactoria, por ejemplo, introduciendo correctamente el usuario y contraseña apropiados, **se abrirá así una sesión en el equipo remoto**, teniendo el cliente a partir de entonces la capacidad de utilizarlo y administrarlo como si estuviera presente físicamente.



4 INSTALACIÓN Y CONFIGURACIÓN INICIAL

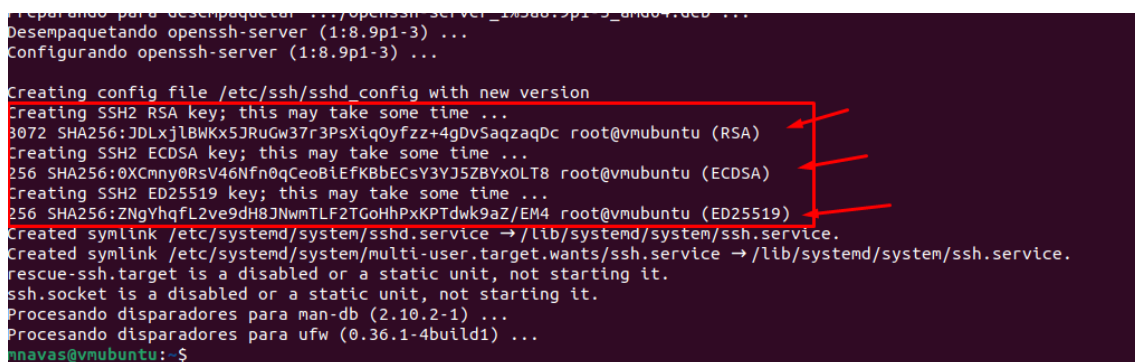
Como comentamos al principio del apartado anterior, hasta ahora solo hemos conocido la teoría de cómo funciona SSH y los fundamentos criptográficos sobre los que se ha desarrollado. A partir de ahora vamos a poner en práctica y aprender cómo se utiliza SSH. Para ello la mejor forma de hacerlo es mediante ejemplos, mediante prácticas. Para esto, lo primero que necesitaremos será un entorno de pruebas. Como se puede suponer, para hacer uso de SSH requerimos de un conjunto de al menos dos máquinas. Estas pueden ser físicas, virtuales utilizando cualquier herramienta de virtualización que se desee utilizar (VirtualBox, VMWare, Docker, ...), o incluso instancias de máquinas en la nube (AWS, Azure, etc). Como se puede observar las opciones son muy variadas, pero sea como fuese vamos a necesitar varias máquinas y algún sistema donde podamos instalar la herramienta de servidor SSH. En principio para instalar SSH utilizaremos cualquier sistema operativo tipo Unix (es decir, alguna variante de BSD, de GNU/Linux o de MacOS). Para el objetivo de este documento se recomienda instalar una máquina virtual con cualquier distribución de Linux, por ejemplo Ubuntu, donde en nuestro caso y para los ejemplos expuestos a partir de aquí se instalará el servidor OpenSSH. Por supuesto se podrá utilizar cualquier cliente SSH (existen clientes disponibles en todos los SSOO tanto Linux, MacOS como Windows) para realizar la conexión a dicho servidor.

A continuación se va a explicar cómo proceder a la instalación del servidor SSH en una máquina Ubuntu con un usuario con privilegios de *root*. El proceso puede variar ligeramente dependiendo del sistema operativo que se utilice.

El comando que debemos utilizar para instalar el servidor SSH es el siguiente (en un *prompt* con permisos de superusuario):

```
apt install openssh-server
```

Tras finalizar la instalación nos generará automáticamente varios pares de claves (pública/privada) como se puede observar en la siguiente captura.



```
Preparando para desempaquetar .../openssh-server_1:8.9p1-3_amd64.deb ...
Desempaquetando openssh-server (1:8.9p1-3) ...
Configurando openssh-server (1:8.9p1-3) ...
Creating config file /etc/ssh/sshd_config with new version
Creating SSH2 RSA key; this may take some time ...
3072 SHA256:JDLxjlBWKx5JRuGw37r3PsXiq0yfzz+4gDvSaqaqDc root@vmubuntu (RSA)
Creating SSH2 ECDSA key; this may take some time ...
256 SHA256:0XCmny0RsV46Nfn0qCeoBiEfKBbECsY3YJ5ZBYxOLT8 root@vmubuntu (ECDSA)
Creating SSH2 ED25519 key; this may take some time ...
256 SHA256:ZNgyHqfL2ve9dH8JNwmTLF2TGoHhPxKPTdWk9aZ/EM4 root@vmubuntu (ED25519)
Created symlink /etc/systemd/system/sshd.service -> /lib/systemd/system/ssh.service.
Created symlink /etc/systemd/system/multi-user.target.wants/ssh.service -> /lib/systemd/system/ssh.service.
rescue-ssh.target is a disabled or a static unit, not starting it.
ssh.socket is a disabled or a static unit, not starting it.
Procesando disparadores para man-db (2.10.2-1) ...
Procesando disparadores para ufw (0.36.1-4build1) ...
nnavas@vmubuntu:~$
```

Para verificar que el servidor se ha instalado correctamente podremos comprobarlo mediante el siguiente comando, que nos devolverá el estado actual del servicio SSH:

```
systemctl status ssh
```

```

mnavas@vmubuntu:~$ sudo systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2022-04-25 22:06:46 CEST; 1min 39s ago
     Docs: man:ssh(8)
           man:ssh_config(5)
   Main PID: 5281 (sshd)
     Tasks: 1 (limit: 9419)
    Memory: 1.7M
       CPU: 10ms
    CGroup: /system.slice/ssh.service
            └─5281 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

abr 25 22:06:46 vmubuntu systemd[1]: Starting OpenBSD Secure Shell server...
abr 25 22:06:46 vmubuntu sshd[5281]: Server listening on 0.0.0.0 port 22.
abr 25 22:06:46 vmubuntu sshd[5281]: Server listening on :: port 22.
abr 25 22:06:46 vmubuntu systemd[1]: Started OpenBSD Secure Shell server.
mnavas@vmubuntu:~$

```

Como se puede ver, con tan solo un comando hemos instalado y activado el servicio SSH en nuestra máquina. A partir de entonces desde cualquier otra máquina con un cliente SSH instalado seremos capaces de acceder de forma remota a este equipo.

Un paso que podría ser necesario es el de permitir SSH a través del *firewall* del sistema operativo. Ubuntu como todas las distribuciones Linux basadas en Debian incorpora una utilidad de firewall denominada *UncomplicatedFirewall (UFW)*, que es una interfaz para *iptables* que a su vez gestiona las reglas de la red. Si este *firewall* se encuentra activo puede que impida las conexiones a nuestro servidor SSH. Para configurar UFW de modo que permita siempre las conexiones mediante este protocolo se debe ejecutar el comando:

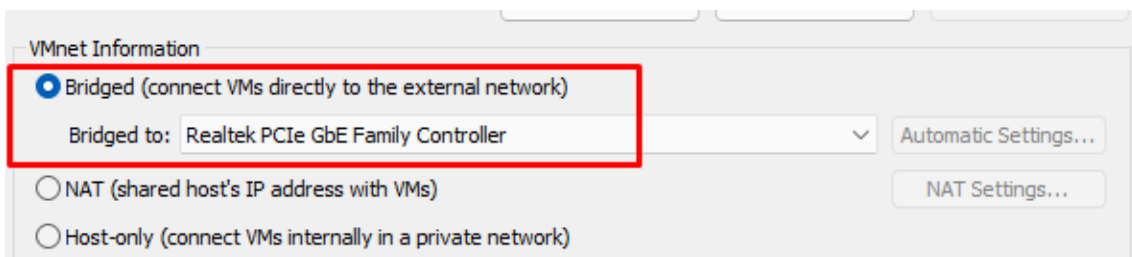
```
ufw allow ssh
```

```

mnavas@vmubuntu:~$ sudo ufw allow ssh
Reglas actualizadas
Reglas actualizadas (v6)

```

Por último en cuanto a configuración, en nuestro caso con VMWare es importante establecer el modo de red *bridge* y seleccionar el controlador de red cableada o por WiFi, según como esté conectada vuestra máquina host a la red.

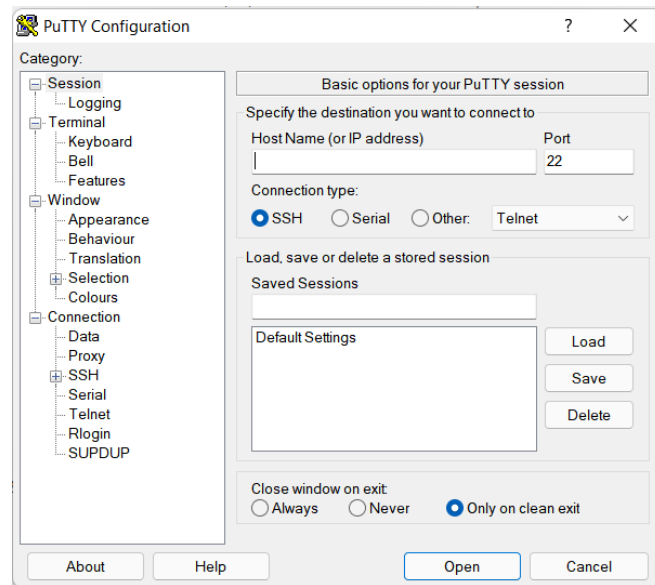


En los siguientes apartados veremos las distintas formas de utilizar SSH así como las posibles configuraciones que podremos implementar.

4.1 USO EN SISTEMAS OPERATIVOS WINDOWS

Hasta hace poco Windows no contaba con una versión nativa de OpenSSH, lo cual era incomprensible dada la importancia de esta herramienta. Debido a esto antes debíamos recurrir a herramientas de

terceros, de las cuales sin duda la más conocida era Putty. Esta herramienta no es más que un cliente de OpenSSH y otros protocolos como los antes mencionados Telnet o RLogin.



Por suerte a partir de Windows 10 ya podemos utilizar SSH de forma nativa integrada en el propio sistema operativo, sin tener que recurrir a herramientas externas. Sin embargo, dependiendo la versión de Windows 10 puede ser necesario que se habilite de forma manual el cliente o servicio de SSH e incluso alguna característica importante para SSH como puede ser el SSH-Agent. Para saber cómo activar esto en Windows 10 u 11 puede revisar los anexos I y II de este documento.

5 UTILIZACIÓN DE SSH

5.1 AUTENTICACIÓN MEDIANTE USUARIO Y CONTRASEÑA

Una vez preparado nuestro entorno de pruebas comenzamos con los métodos de autenticación disponibles en SSH. Y en primer lugar, como no, comenzamos con el método que todo el mundo suele conocer, el de iniciar sesión con un usuario y una contraseña. Es decir, deseamos **abrir una Shell remota en un equipo y para conectarnos necesitamos conocer un nombre de usuario y una contraseña**.

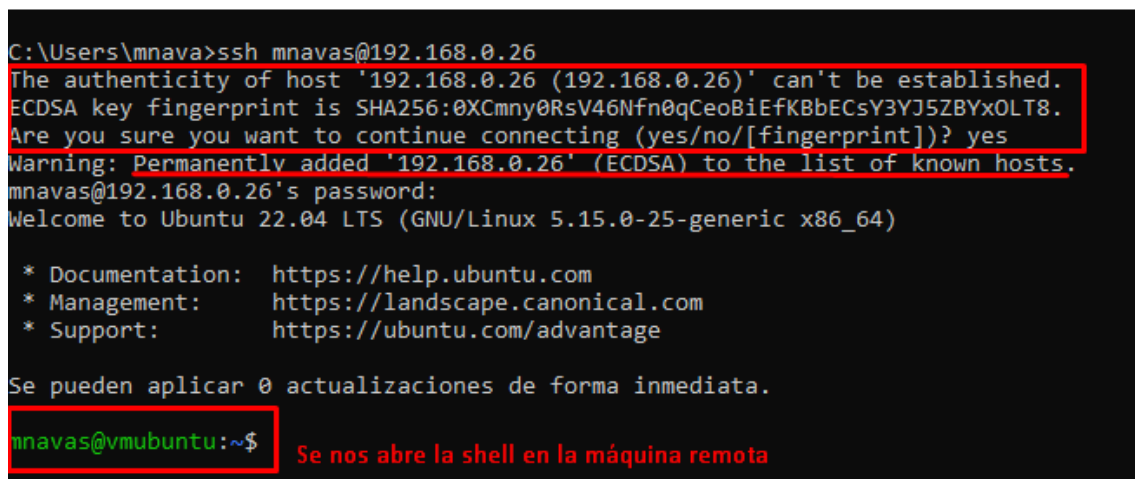
Para llevar a cabo la conexión SSH mediante este método usaremos el siguiente comando:

```
ssh <usuario>@<direccion-ip> -p <puerto>
```

Donde:

- **Usuario** es el nombre de usuario con el que queremos conectarnos al equipo remoto.
- **Dirección ip** es la dirección del servidor SSH con el que vamos a establecer la conexión.
- **Puerto** es el puerto en el que debe estar en escucha el servicio SSH en la máquina remota, que por defecto y si no se indica nada en el comando es el 22.

A continuación se muestra un ejemplo de conexión SSH mediante el método de autenticación por usuario y contraseña:



```
C:\Users\mnava>ssh mnavas@192.168.0.26
The authenticity of host '192.168.0.26 (192.168.0.26)' can't be established.
ECDSA key fingerprint is SHA256:0XCmny0RsV46Nfn0qCeoBiEfKBbECsY3YJ5ZBYx0LT8.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.0.26' (ECDSA) to the list of known hosts.
mnavas@192.168.0.26's password:
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-25-generic x86_64)

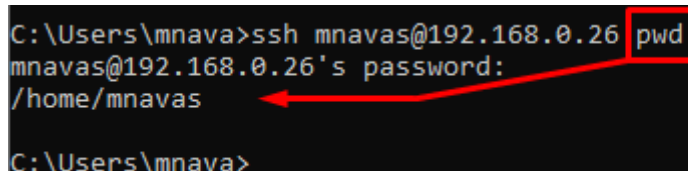
 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

Se pueden aplicar 0 actualizaciones de forma inmediata.
mnavas@vmubuntu:~$
```

Al iniciar sesión en el equipo remoto podremos ver que el *prompt* de la consola cambia, mostrándonos ahora que estamos conectados con el usuario y la máquina con la que hemos establecido la conexión SSH, ya que todas las instrucciones que introduzcamos para que el Shell las interprete se ejecutarán en el sistema remoto en lugar de en nuestra máquina local. Como se puede observar en la imagen, cuando lanzamos el comando para abrir la conexión y el cliente SSH consigue localizar y establecer comunicación con el servidor, **nos muestra por consola la huella de la clave pública del servidor** al que nos estamos conectando (en nuestro ejemplo la máquina virtual llamada “vmubuntu”). Como hemos explicado anteriormente, el servidor SSH contiene una clave privada y una clave pública, a partir de esta clave pública y mediante una función hash (en este caso nos indica que se utiliza el algoritmo SHA256) genera una cadena de texto (huella) que identifica inequívocamente a dicha clave pública. Ahora bien, el cliente obtiene este hash y, **al no poder verificar que esa huella corresponde con la clave pública del servidor por no tener almacenada esta información, delega en el usuario la decisión de confirmar que confía en la clave que le está enviando el servidor**. Si el usuario (nosotros) indicamos que sí confiamos en ella, entonces el cliente SSH procede y abre la Shell remota, no sin antes añadir la clave pública que ha recibido del servidor a la lista de claves conocidas (esta información se almacena en el archivo `known_host`, dentro de la carpeta de configuración de SSH como veremos más adelante). Esta es la configuración por

defecto de un servidor SSH, más adelante veremos cómo cambiar este comportamiento ya que obviamente lo más seguro es almacenar la clave pública del servidor al que queremos conectarnos de forma segura en el directorio adecuado (fichero `known_hosts`) para que el cliente SSH no tenga que preguntarnos sobre la autenticidad de dicha clave, pues esta comprobación lo realizaría en ese caso automáticamente.

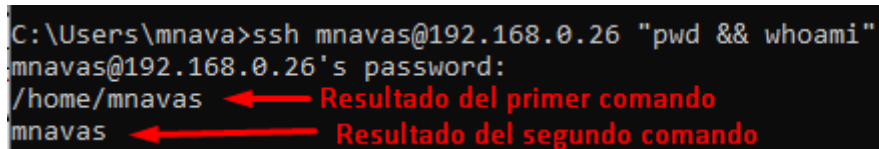
Además de abrir un Shell remoto el comando `ssh` también nos da la posibilidad de ejecutar una instrucción específica sin necesidad de abrir una consola interactiva con el usuario al que nos conectamos, de esta forma podremos ejecutar ordenes de una forma rápida y sencilla, por ejemplo:



```
C:\Users\mnava>ssh mnavas@192.168.0.26 pwd
mnavas@192.168.0.26's password:
/home/mnavas
C:\Users\mnava>
```

A terminal window showing a Windows command prompt. The user enters the command `ssh mnavas@192.168.0.26 pwd`. The `pwd` command is highlighted with a red box. The terminal shows the remote host's password prompt, the user's password, and the output `/home/mnavas`. The prompt returns to the local Windows shell.

Como se puede observar en esta ocasión el *prompt* no ha cambiado en ningún momento, pero sin embargo el comando `pwd` nos devuelve el resultado de la ejecución en el equipo remoto. Mediante esta técnica podremos incluso ejecutar varios comandos anidados, tal y como se haría en una Shell de forma tradicional uniendo los comandos con el operador `&&`, pero entrecomillados con comillas simples. Por ejemplo:



```
C:\Users\mnava>ssh mnavas@192.168.0.26 "pwd && whoami"
mnavas@192.168.0.26's password:
/home/mnavas
mnavas
```

A terminal window showing a Windows command prompt. The user enters the command `ssh mnavas@192.168.0.26 "pwd && whoami"`. The terminal shows the remote host's password prompt, the user's password, and the output of two commands: `/home/mnavas` and `mnavas`. Red arrows point from the text "Resultado del primer comando" to the first output line and "Resultado del segundo comando" to the second output line.

El ejemplo anterior ejecutaría en primer lugar el comando `pwd` y después `whoami` en el equipo remoto, y después volvería a la Shell de la máquina local. Si alguno de los comandos ejecutados devolviese algún mensaje por consola, como es el caso del ejemplo expuesto, se nos mostraría en la Shell del mismo modo que si estuviéramos conectados a la Shell remota.

Es importante destacar que, por defecto, la configuración de SSH no permite iniciar sesión de forma remota con el usuario `root`. Simplemente por cuestiones de seguridad ya que sería peligroso que un atacante que se haga con la contraseña del usuario `root` pudiera acceder de forma sencilla a un usuario con privilegios de administrador. Por tanto, si se intenta acceder con el método que acabamos de ver con el usuario `root` el cliente SSH nos devolverá el error "Permission denied, please try again" hasta un máximo de 3 ocasiones, y después cancelará el intento de conexión por SSH. Para modificar esta configuración se deberá editar el parámetro `PermitRootLogin` que por defecto contiene el valor `prohibit-password` y cambiarlo al valor `yes` en el fichero de configuración `/etc/ssh/sshd_config` evitentemente con un usuario con permisos de superadmin. De esta forma sí se nos permitiría iniciar sesión con el usuario `root` de forma remota.

```
C:\Users\mnava>ssh root@192.168.0.26
root@192.168.0.26's password:
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-25-generic x86_64)


 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

Se pueden aplicar 0 actualizaciones de forma inmediata.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@vmubuntu:~#
```

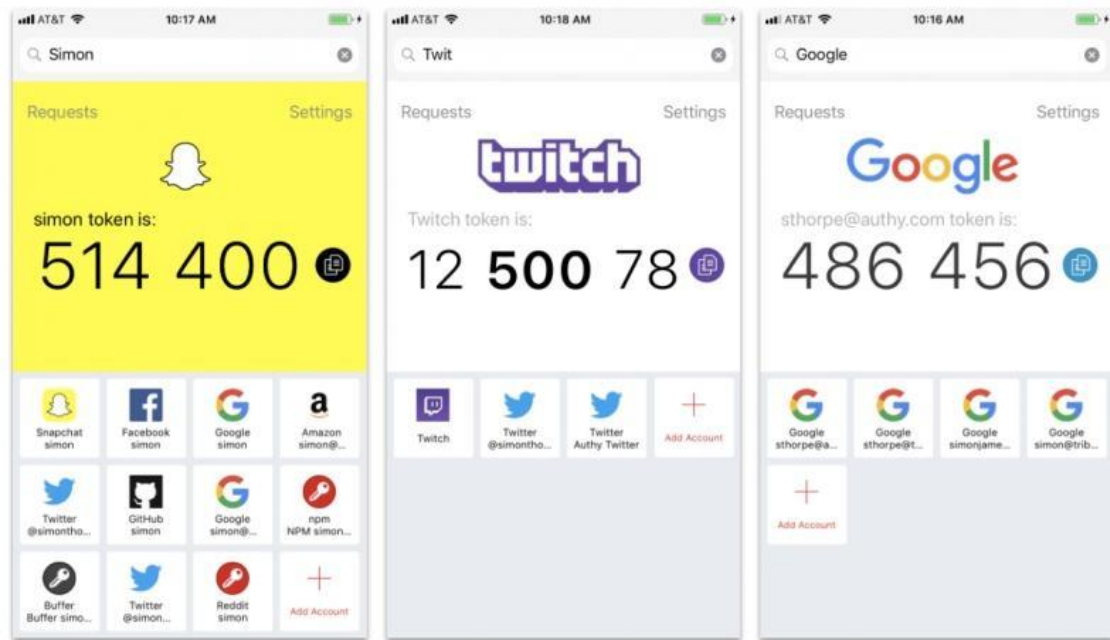


5.2 AUTENTICACIÓN MEDIANTE USUARIO, CONTRASEÑA Y 2FA

La autenticación de doble factor (2FA) es un paso adicional en el proceso de inicio de sesión, como escanear la huella o introducir un código, que ayuda a verificar tu identidad y evitar que terceros puedan acceder a tu información privada. La autenticación de doble factor ofrece un nivel extra de seguridad al que los ciberdelincuentes no pueden acceder fácilmente, ya que necesita algo más que tus credenciales de usuario y contraseña.

Cada vez son más los servicios que integran este tipo de autenticación, sin embargo, casi nunca viene activada por defecto ya que hace el inicio de sesión más tedioso para los usuarios. SSH es uno de estos casos. En el Anexo V veremos cómo se activa y configura.

El inicio de sesión a un equipo mediante SSH que tenga el 2FA activado es similar al inicio de sesión mediante usuario y contraseña, con la excepción de que, tras introducir la contraseña, se te va a pedir un código de doble autenticación. Previamente, durante la configuración del 2FA en el equipo remoto, has tenido que vincular tu equipo/cuenta a una aplicación de 2FA como pueden ser [Authy](#) o [Google Authenticator](#). De esta forma, la aplicación generará códigos de 2FA para todas las cuentas que tengas vinculadas. A continuación, os muestro un ejemplo de aplicación para 2FA:



Ahora que ya sabemos qué es el 2FA vamos a ver un ejemplo de inicio de sesión mediante SSH a un equipo que tiene configurado 2FA:

Al igual que hemos hecho en el punto anterior, introducimos:

```
ssh <usuario>@<direccion-ip> -p <puerto>
```

En nuestro caso no hemos introducido puerto porque el equipo remoto donde quiero conectarme tiene SSH en el puerto por defecto, el puerto 22.

Como vemos en la imagen, nos pide la contraseña y, en caso de introducirla correctamente, nos pide un código de verificación. Éste es el código de 2FA. Si introducimos correctamente el código, habremos iniciado sesión en el equipo.

```
PS C:\Users\jemoa> ssh pi@192.168.18.95 Indicamos el host al que queremos conectarnos
Password:
Verification code: Indicamos la contraseña y el código 2FA
Linux raspberrypi 5.10.17-v8+ #1414 SMP PREEMPT Fri Apr 30 13:23:25 BST 2021 aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Jan 24 15:26:58 2022 from 192.168.18.131

Wi-Fi is currently blocked by rfkill.
Use raspi-config to set the country before use.

pi@raspberrypi:~$ Estamos dentro!
```

5.3 AUTENTICACIÓN MEDIANTE CLAVE PÚBLICA

Continuamos con los métodos de utilización de SSH y lo hacemos con el método de autenticación más importante, la conexión con pares de claves publica/privada. Como hemos visto en la instalación del

servidor SSH se generan varios pares de claves pública/privada, uno para cada algoritmo disponible en la versión de SSH que hayamos instalado. Una de estas claves será la que el servidor le mande al cliente durante la fase de negociación, como ya hemos visto en el apartado anterior.

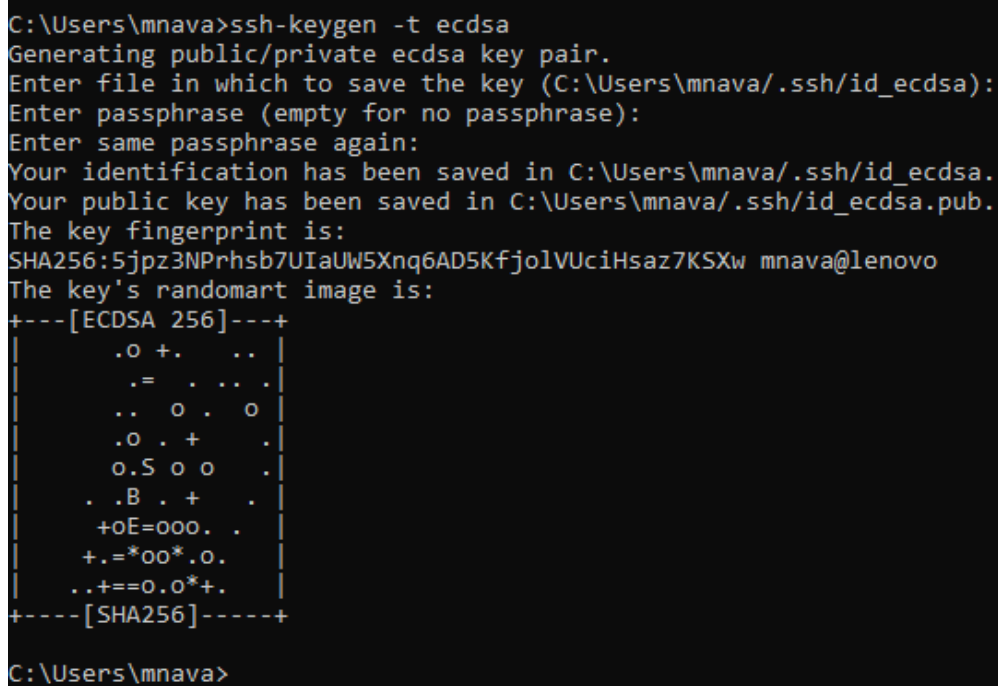
Ahora lo que vamos a hacer es la parte análoga en el lado del cliente. Inicialmente el cliente no genera automáticamente una clave privada propia. Por tanto, para llevar a cabo el proceso de autenticación mediante clave pública tendremos que

1. Generar una clave privada del cliente.
2. Obtener a partir de dicha clave la clave pública asociada.
3. Copiar esta clave pública de nuestro cliente en el servidor SSH al que queremos conectarnos.

De esta forma podremos establecer la conexión SSH sin necesidad de introducir la contraseña como hemos visto en el método anterior.

Para generar el par de claves pública/privada en nuestro cliente utilizaremos el comando `ssh-keygen` indicando el tipo de algoritmo que queremos utilizar:

```
ssh-keygen -t ecdsa
```



```
C:\Users\mnava>ssh-keygen -t ecdsa
Generating public/private ecdsa key pair.
Enter file in which to save the key (C:\Users\mnava/.ssh/id_ecdsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\mnava/.ssh/id_ecdsa.
Your public key has been saved in C:\Users\mnava/.ssh/id_ecdsa.pub.
The key fingerprint is:
SHA256:5j pz3NPrhsb7UIaUw5Xnq6AD5Kfj olVUciHsaz7KSXw mnava@lenovo
The key's randomart image is:
+---[ECDSA 256]---+
|      .O +.  .. |
|      . =   . . . |
|      .. O . O |
|      .O . +   . |
|      O.S O O   . |
|      . .B . +   . |
|      +OE=ooo. . |
|      +.=*oo*.O. |
|      ..+=O.O*+. |
+---[SHA256]-----+
C:\Users\mnava>
```

Como se puede observar en la imagen anterior, al ejecutar el comando `ssh-keygen` en primer lugar nos pregunta con que nombre queremos almacenar los archivos de claves que se generarán. Por defecto se utilizará el nombre del algoritmo que hemos seleccionado para generar los archivos de clave pública y privada: `id_<nombre-algoritmo>` y `id_<nombre_algoritmo>.pub`. Por tanto si no introducimos nada se nos guardarán con estos nombres. Después de esto nos pide que introduzcamos una clave de paso para las claves, en este ejemplo vamos a omitir esto. Simplemente no introducimos nada y pulsamos enter las dos veces que nos solicita la clave de paso y finalmente nos indicará que ha creado los archivos de claves, en concreto nos muestra por la pantalla el fichero `id_ecdsa.pub` que es la clave pública. Además también nos muestra el hash asociado a dicha clave pública y un randomart que representa a dicho hash (esto no nos será de utilidad).

Si nos dirigimos al directorio donde hemos generado las claves podremos comprobar que tenemos dos archivos, e incluso podremos examinar su contenido como se puede observar en la siguiente imagen.

```
C:\Users\mnava\.ssh>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 5484-35FF

Directorio de C:\Users\mnava\.ssh

25/04/2022  22:52  <DIR>          .
25/04/2022  21:39  <DIR>          ..
25/04/2022  22:52             505 id_ecdsa
25/04/2022  22:52             175 id_ecdsa.pub
25/04/2022  22:30             177 known_hosts
                3 archivos             857 bytes
                2 dirs 180.775.170.048 bytes libres

C:\Users\mnava\.ssh>type id_ecdsa.pub
ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBHQtiUrNYuNDHl90iJa6PJ8I87/5Nti4rofufEqWbNX
C0yktLmOHgk4+zJknGnIR2eep+f6FH5nA+seghFHo= mnava@lenovo

C:\Users\mnava\.ssh>
```

El siguiente paso, como hemos comentado es el de copiar la clave pública que nos ha generado el comando ssh-keygen en el servidor SSH. Esto podremos hacerlo de muchas formas, podríamos simplemente copiar físicamente con algún medio extraíble (memoria usb, disco duro externo, etc) el archivo id_ecdsa.pub en el archivo correspondiente de la maquina a la que queremos conectar de forma remota, pero este escenario puede ser impracticable en algunos casos. Por suerte para llevar a cabo esta tarea de una forma remota, sencilla y rápida ssh nos proporciona un comando que nos permite copiar nuestra clave pública autenticándonos de forma remota:

```
ssh-copy-id -i id_ecdsa mnavas@192.168.0.26
```

Importante: para usuarios de Windows la utilidad ssh-copy-id no está disponible. Por lo tanto para llevar a cabo la copia de la clave pública al fichero correspondiente del servidor de la manera más cómoda posible debemos utilizar el siguiente comando. ¡Ojo! Si no existe el fichero authorized_keys deberemos crearlo manualmente en el servidor antes de ejecutar este comando, de lo contrario nos devolverá un error (el archivo authorized_keys no existirá si no se ha copiado ninguna clave de un cliente anteriormente en el servidor, de lo contrario sí debería existir):

```
type id_ecdsa.pub | ssh mnavas@192.168.0.26 "cat >> .ssh/authorized_keys"
```

```
C:\Users\mnava\.ssh>type id_ecdsa.pub | ssh mnavas@192.168.0.26 "cat >> .ssh/authorized_keys"
mnavas@192.168.0.26's password:

C:\Users\mnava\.ssh>
```

Como se puede comprobar la ejecución del comando nos pide en primer lugar que introduzcamos la contraseña para autenticarnos (como hemos hecho siempre hasta ahora) y tras hacerlo con éxito nos indica que se ha añadido una clave, invitándonos a iniciar sesión de forma remota con el comando ssh usuario@ip (esto sólo si hacemos uso de ssh-copy-id, sorry Windows☹️):

```
C:\Users\mnava\.ssh>ssh mnavas@192.168.0.26
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-25-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

Se pueden aplicar 0 actualizaciones de forma inmediata.

Last login: Mon Apr 25 22:30:57 2022 from 192.168.0.14
mnavas@vmubuntu:~$
```

No nos pide contraseña !! :)

Y como se puede comprobar, efectivamente ahora el proceso de conexión remota no nos ha pedido la contraseña del usuario, sino que directamente nos ha abierto una Shell para poder interactuar con el sistema. La clave que nos ha copiado anteriormente el comando `ssh-copy-id` se almacena automáticamente en el fichero `authorized_keys` dentro del directorio oculto `.ssh` en el directorio home del usuario. Si examinamos el contenido de dicho archivo podremos comprobar que contiene la misma clave pública que teníamos en nuestra máquina de partida.

```
mnavas@vmubuntu:~$ cat .ssh/authorized_keys
ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBHQtiUrNyUNDHl90
iJa6PJ8I87/5Nti4rofufuEqWbNXC0yktLm0Hgk4+zJknGnLIR2eep+f6FH5nA+seghFHo= mnava@lenovo
mnavas@vmubuntu:~$
```

Si tuviéramos más de un par de claves pública/privada podríamos indicarle al comando `ssh` cual de ellas queremos utilizar mediante la opción `-i`. El resultado será el mismo **siempre y cuando la clave privada seleccionada tenga su análoga clave pública alojada en el servidor SSH** al que nos estamos conectando.

```
ssh -i ~/.ssh/miclaveprivada mnavas@192.168.0.26
```

Las principales ventajas de este tipo de autenticación frente al más tradicional de contraseñas son varias. En primer lugar el usuario que accede al equipo remoto **no requiere memorizar ninguna contraseña** para poder entrar (podríamos imaginar un administrador de diversos sistemas que tuviera que recordar una contraseña por cada uno de las máquinas remotas a las que tuviera que acceder, puesto que utilizar una misma clave para todas sería una práctica evidentemente poco recomendable desde el punto de vista de la seguridad de los sistemas que administra), con lo cual se le facilita en gran medida esta tarea. En segundo lugar, **permite la gestión automática de conexiones cifradas** con la máquina remota, permitiendo la **automatización de tareas** que requieran conexiones a dicha máquina. En tercer lugar, se trata de un mecanismo mucho **más seguro que el método por contraseña**, simplemente porque las contraseñas siempre son susceptibles de atacar mediante técnicas como la fuerza bruta o ataques por diccionario si no se establecen contraseñas con un nivel recomendable de complejidad. Por último, si hubiera una persona encargada de administrar la máquina remota a la que queremos conectarnos sería **mucho más sencillo para dicha persona otorgar accesos** a ésta, pues sólo tendría que pedir la clave pública a aquellos usuarios que quisieran conectarse a dicha máquina, en lugar de tener que generarle una contraseña y pasársela al usuario (nos ahorramos que el administrador del sistema conozca las contraseñas de los usuarios que se conecten de forma remota). Por este motivo es el método de autenticación que proporcionan de forma predeterminada todos los servidores cloud hoy en día (Amazon Web Services, Azure, Google Cloud, etc).

A partir de este momento podríamos incluso plantearnos eliminar el acceso mediante contraseña al servidor remoto, puesto que ya no es necesario, aunque se pueden mantener ambos métodos sin problemas. Evidentemente las acciones que hemos visto anteriormente mediante el método de autenticación por contraseña (abrir una Shell remota y ejecutar instrucciones directamente en la máquina

remota) las podemos realizar ahora de una forma mucho más ágil, sin tener que introducir la contraseña cada vez.

¿Qué ocurre si perdiéramos nuestra clave pública? Como hemos comentado anteriormente la clave pública se genera a partir de la clave privada, por esto es de vital importancia custodiar muy bien la clave privada (entre otros motivos). Si en algún momento perdiéramos el acceso a todos los archivos de clave pública que tuviéramos (recordar que la clave pública se puede distribuir libremente y por lo tanto copiar cuantas veces sea necesario) **podríamos regenerar dicho archivo a partir del fichero de clave privada** y el comando `ssh-keygen`:

```
ssh-keygen -y -f <archivo_clave_privada> >> <nuevo_archivo.pub>
```

```
C:\Users\mnava\.ssh>del id_ecdsa.pub
C:\Users\mnava\.ssh>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 54B4-35FF

Directorio de C:\Users\mnava\.ssh
26/04/2022  23:18  <DIR>      .
25/04/2022  21:39  <DIR>      ..
26/04/2022  23:17                505 id_ecdsa
26/04/2022  20:07                400 known_hosts
                2 archivos          905 bytes
                2 dirs  185.168.470.016 bytes libres

C:\Users\mnava\.ssh>ssh-keygen -y -f id_ecdsa >> id_ecdsa.pub
C:\Users\mnava\.ssh>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 54B4-35FF

Directorio de C:\Users\mnava\.ssh
26/04/2022  23:18  <DIR>      .
25/04/2022  21:39  <DIR>      ..
26/04/2022  23:17                505 id_ecdsa
26/04/2022  23:18                175 id_ecdsa.pub
26/04/2022  20:07                400 known_hosts
                3 archivos          1.080 bytes
                2 dirs  185.168.404.480 bytes libres

C:\Users\mnava\.ssh>type id_ecdsa.pub
ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIbmlzdHAyNTYAAABBLJ509KgJVnsNDy5R8zSzQPvHVSPqwXIB9Ugra2BWeN41zA/
DakNBAOSP02Z1wJ5/2w0+2/sK91eGJdAoNZ18I0= mnava@lenovo

C:\Users\mnava\.ssh>
```

Borramos nuestro archivo de clave pública

Generamos una nueva clave pública a partir del archivo de clave privada

Si comparamos con el primer archivo `id_ecdsa.pub` podremos comprobar que el contenido es el mismo, es decir la clave pública es la misma que teníamos en el archivo perdido

Si comparásemos el archivo de clave pública perdido con el que acabamos de generar observaríamos que su contenido es idéntico³. Esto es normal ya que de otra forma tendríamos que volver a subir nuestra clave pública nueva a los servidores a los que ya habíamos copiado nuestra antigua clave pública, con lo cual no necesitaríamos volver a hacer este paso.

5.4 AUTENTICACIÓN MEDIANTE CLAVE PÚBLICA Y FRASE DE PASO

En el apartado anterior hemos visto como realizar el proceso de autenticación mediante SSH con par de claves pública/privada. En concreto hemos aprendido cómo se generan estas claves, cómo copiar la clave pública al servidor ssh al que nos conectaremos y hemos comprobado que a partir de entonces la autenticación no requiere la introducción manual de ningún tipo de credencial. La desventaja de este

³ El formato de los archivos de clave pública es `<algoritmo> <clave> <comentario>`, el comentario en nuestro caso es el nombre de usuario y máquina que lo ha generado, pero dependiendo de la versión del `ssh-keygen` que utilizemos podría variar. Esto no es problema puesto que el comentario siempre se ignora y no afectaría en el proceso de autenticación en ningún caso. Nota: en este caso si comparáis con la clave que hemos estado usando en las capturas anteriores comprobaréis que no es la misma clave, pero esto es porque durante las pruebas borré en un descuido el par de claves que estábamos usando, por lo cual tuve que generar otro nuevo par de claves pública/privada y obviamente en ese caso la clave pública sí es distinta.

mecanismo de autenticación es que **el peso de la seguridad de la comunicación reside únicamente en la clave privada del cliente**. Esto ocasiona que, si en algún momento colocamos dicha clave privada en otro equipo y la perdemos o cualquier otra persona tiene acceso a ella podría autenticarse en la máquina remota con nuestra identidad sin ningún tipo de problema, pues dicha clave privada le da acceso a cualquier equipo donde su análoga clave pública esté disponible, como hemos comprobado anteriormente. Por este motivo el uso de este mecanismo por sí solo únicamente se recomienda en entornos relativamente controlados o donde tengamos cierto nivel de seguridad de que no se va a ver comprometida nuestra clave privada.

Para solucionar esta problemática existe el mecanismo de frase de paso (**passphrase**). La frase de paso (se utiliza esta denominación para no confundir con la contraseña del equipo remoto) no es más que **una contraseña que protege la clave privada**. Si añadimos una frase de paso a nuestra clave privada significa que sólo con la posesión de dicho fichero de clave privada no es suficiente para iniciar sesión en el sistema remoto, sino que además cada vez que se desee utilizar la clave privada tendremos que teclear la frase de paso que la protege.

Para ilustrar este método procedemos a crear un nuevo par de claves asignándole el nombre "confrasedepaso" mediante el comando `ssh-keygen` que ya conocemos, salvo que en este caso sí introducimos una frase de paso (passphrase) durante el proceso:

```
C:\Users\mnava\.ssh>ssh-keygen -t ecdsa
Generating public/private ecdsa key pair.
Enter file in which to save the key (C:\Users\mnava\.ssh/id ecdsa): confrasedepaso
Enter passphrase (empty for no passphrase): 
Enter same passphrase again: 
Your identification has been saved in confrasedepaso.
Your public key has been saved in confrasedepaso.pub.
The key fingerprint is:
SHA256:oQFKzVp5ZbECGPeI4mX01JcLfACx/OALQ4vGVlEboo mnava@lenovo
The key's randomart image is:
+---[ECDSA 256]---+
|.X@*+o .+.
|*O*O*oo. .
|+=&*+oo o
|+*O+.. + .
|E.o . S
|. .
|
+-----[SHA256]-----+
C:\Users\mnava\.ssh>
```

No se visualiza pero hemos indicado un passphrase

Acto seguido copiamos la nueva clave pública generada en nuestro servidor ssh con el comando `ssh-copy-id` (salvo en Windows que debemos usar el comando que explicamos anteriormente):

```
C:\Users\mnava\.ssh>type confrasedepaso.pub | ssh mnava@192.168.0.26 "cat >> .ssh/authorized_keys"
mnava@192.168.0.26's password:
C:\Users\mnava\.ssh>
```

Por último, procedemos a abrir una conexión utilizando la nueva clave, para ello como ya teníamos otro par de claves tendremos que indicarle al comando `ssh` que queremos realizar la autenticación con la nueva clave que acabamos de generar, para ello utilizaremos la opción `-i` seguido del nombre de la clave. Cuando ejecutamos el comando, **nos muestra un diálogo donde nos pide la frase de paso de la clave privada que vamos a usar**. En algunos sistemas nos puede permitir marcar una opción para que automáticamente desbloquee la clave (no nos pida su frase de paso) cuando iniciemos sesión con este

usuario. Sin embargo lo normal es que nos pida esta frase de paso siempre que solicitamos la autenticación con el equipo remoto.

```
C:\Users\mnava\.ssh>ssh -i confrasedepaso mnavas@192.168.0.26
Enter passphrase for key 'confrasedepaso':
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-25-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

Se pueden aplicar 0 actualizaciones de forma inmediata.

Last login: Mon Apr 25 23:15:21 2022 from 192.168.0.14
mnavas@vmubuntu:~$
```

Una vez que introducimos correctamente dicha frase de paso conseguimos abrir una Shell remota como se puede observar en la imagen anterior.

El uso de este mecanismo nos sitúa en la paradoja de que volvemos a requerir la interacción por parte del usuario para autenticar en el servidor, algo que habíamos superado al pasar del método de autenticación por contraseña al uso de pares de claves pública/privada. Esto obviamente penaliza la agilidad a la hora de establecer las conexiones. Para solucionar esto vamos a ver en la siguiente sección una herramienta auxiliar menos conocida pero que nos ayudará a utilizar las frases de paso para proteger nuestras claves privadas de una forma más transparente, reuniendo de esta forma lo mejor de ambos mecanismos: la seguridad de una contraseña y la agilidad de los archivos de claves.

5.5 SSH-AGENT

En este apartado vamos a conocer el programa SSH-Agent. Este software auxiliar de ssh es menos conocido pero muy útil ya que nos permite almacenar temporalmente durante un tiempo nuestras claves privadas. En el caso de que se añada una clave privada sin frase de paso SSH-Agent se encargará de incorporarle una automáticamente, mientras que si ya existe una frase de paso definida para la clave privada la podremos incorporar al momento de incluir dicha clave privada al SSH-Agent. Esta herramienta, por tanto, nos proporciona la capacidad de utilizar conexiones SSH mediante el método de autenticación con clave pública, con o sin frase de paso adicional por cada clave que gestionemos, de una manera muy cómoda ya que la frase de paso solo debemos establecerla una vez a partir del momento que deseemos establecer la conexión con la máquina remota. Por ejemplo, nos podría servir para procesos no interactivos durante el tiempo de vida de la sesión. Se trata por tanto de una herramienta fundamental para todos aquellos usuarios que trabajan constantemente con SSH para administrar y usar máquinas remotas.

Para verificar que está disponible el proceso SSH_AGENT en nuestra máquina cliente Linux podemos utilizar el siguiente comando (En Windows SSH-Agent se encuentra deshabilitado por defecto, para saber cómo habilitarlo revisar el Anexo II):

```
env | grep SSH
```

```
ps aux | grep ssh-agent
```



```
navasdanasm@osboxes:~$ env |grep SSH
SSH_AGENT_LAUNCHER=gnome-keyring
SSH_AUTH_SOCK=/run/user/1001/keyring/ssh
navasdanasm@osboxes:~$ ps aux |grep ssh-agent
navasda+  4574  0.0  0.0  6060  3956 ?        S    14:20   0:00 /usr/bin/ssh-agent -D -a /run/user/1001/keyring/.ssh
navasda+  4899  0.0  0.0  9284  2408 pts/0    S+   14:48   0:00 grep --color=auto ssh-agent
navasdanasm@osboxes:~$
```

Si se obtiene un resultado similar al que se muestra en la imagen anterior significa que SSH-Agent esta funcionando en segundo plano. Vamos a ver ahora cómo agregar una clave a la gestión de SSH-Agent.

Nota: A partir de ahora se han eliminado todas las claves que se habían generado anteriormente tanto los archivos del cliente como del fichero `authorized_keys` del servidor SSH, simplemente para partir de un entorno sin claves y que la explicación quede más clara.

En primer lugar generamos un nuevo par de claves con `ssh-keygen`. Esta vez utilizamos otro algoritmo, por ejemplo RSA:

```
ssh-keygen -t rsa
```

A continuación utilizamos el comando `ssh-add` para añadir la clave que acabamos de generar a la gestión del SSH-Agent:

```
ssh-add ~/.ssh/id_rsa
```

```
C:\Users\mnava\.ssh>ssh-agent
C:\Users\mnava\.ssh>ssh-add id_rsa
Enter passphrase for id_rsa:
Identity added: id_rsa (mnava@lenovo)
C:\Users\mnava\.ssh>
```

Como vemos nos pide la frase de paso asociada a la clave para poder añadirla. A partir de ahora, como tenemos nuestra clave añadida a la gestión del SSH-Agent, solo se nos preguntará por la frase de paso la primera vez que se intente realizar la autenticación. En las sucesivas conexiones mientras dure la sesión del usuario en la máquina local, no se nos pedirá la frase de paso para realizar la autenticación en el equipo remoto.

Otras acciones interesantes que podemos realizar con el comando `ssh-add` es comprobar la lista de identidades (claves) que tenemos añadida al SSH-Agent mediante la opción `-l` (`ssh-add -l`) o eliminar todas las identidades que estén guardadas mediante la opción `-D` (`ssh-add -D`). Nota: en sistemas Linux, si tenemos almacenados archivos de clave en el directorio por defecto (`~/.ssh/`) el comando `ssh-add -D` no eliminará las identidades correspondientes a esas claves, por tanto, antes de lanzar el comando debemos asegurarnos de eliminar físicamente los archivos de claves del directorio por defecto.

```
C:\Users\mnava\.ssh>ssh-add -l
3072 SHA256:NUH8DjFc8geRkhq5KCHerM/vi0N6C8EuzwDNdYVehII mnava@lenovo (RSA)
C:\Users\mnava\.ssh>ssh-add -D
All identities removed.
C:\Users\mnava\.ssh>ssh-add -l
The agent has no identities.
C:\Users\mnava\.ssh>
```


5.6 GESTIÓN DE FICHEROS: AUTHORIZED_KEYS Y KNOWN_HOSTS

Pasamos a continuación a explicar de forma un poco más detallada dos ficheros de uso muy importantes en ssh: **authorized_keys** y **known_hosts**. Estos ficheros se localizan en el directorio oculto `.ssh` que se encuentra dentro del directorio personal del usuario (`/home/<usuario>` o `C:\Users\<usuario>` en Windows).

El **fichero `authorized_keys`** almacena las claves públicas que tienen permiso para autenticarse como el usuario al que pertenece. Por tanto visualizando el contenido de este fichero podremos saber cuales son las claves públicas de las personas que tienen acceso de forma remota a la máquina siempre que dispongan de su correspondiente clave privada.

```
mnavas@vmubuntu:~$ cat .ssh/authorized_keys
ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBN6AS14jRddLPvFvd10
/oa90AEVf0gXzdZUQd6aLezWb8qmKhPKKzjJJi8qVn6NQHlaMAR75VjzFqXES6GzVQjY= mnava@lenovo
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQgQC8rxB6vDgALEdQLLogDL2AMnOW3jve6sF16G5qo9ACfNEJQ2jDkte
g8n+EFc5tEcLayT9P5id6+uwqHUu51ev/C19bmKpUskngn3AVs7Tb6TDRwGNxeF50hkbbeXBtgaSRv/Oyv9xok1GIAH
C41x/5+XXEV33LJHHevQnBXSwm1/K49wrf+77T3TEONr11fS+BJU/X8E6DpspdEORzESf7tIG42z1vv8/wJBQXpCQnJH
XjAI3M3IhT0sf+2DQAjXd1mIN+npdUxhBy8UjEwN9ANlwGpk/eLfeLowGaI9uWeRb5/gj804mk1jZaQkTDBDFnqBX10
VG2ipcaZ4MzTxU5iBkki6C0zTDkPcIKmBG3RyZdgnIx9AcE7KCHgGoW0ubLWylJ74dQVAlc3mnDwzINm2ZRw4ZF4TLk
/L2dFNrPM46oBIVQkKDYe0g7/iW/487Fm4WPghzTY1Q6A8wcTS1Z4gVN1T5XcJMcDipaH8v+nbzYPNRC1o+V8h3T6xQ
8fVnk= mnava@lenovo
mnavas@vmubuntu:~$
```

Por ejemplo, en la captura anterior podemos ver que en nuestro servidor de prueba tenemos la clave pública de la clave RSA que hemos generado antes junto con la clave anterior que hemos utilizado en las distintas pruebas. El formato de la llave pública siempre es el mismo, comenzando por el nombre del algoritmo, seguido del “chorizo” ilegible que representa la clave en sí, el cual variará en cuanto a tamaño dependiendo del algoritmo que se utilice para su generación, y al final opcionalmente puede contener un comentario, en este caso el usuario y nombre de la máquina con la que se generó dicha clave. Aquí se pueden observar también las diferencias en cuanto a la longitud de la clave de los diferentes algoritmos de criptografía asimétrica que hemos utilizado, pues se ve perfectamente que las claves que utiliza el algoritmo RSA son bastante mas largas que las que utiliza el algoritmo ECDSA.

La gestión del fichero `authorized_keys` consiste en lo siguiente: como estamos dando autorización a las claves que existen en este fichero, si en algún momento se deja de utilizar una clave que ya ha sido registrada en el fichero **debemos borrarla del mismo**, simplemente editando el archivo. Por ejemplo, podríamos borrar la clave RSA que hemos generado para el ejemplo con SSH-Agent. Por tanto, como esa clave privada ya no existe debemos proceder también a borrar su clave pública de este fichero dejándolo en este caso solo con la otra clave pública (la correspondiente al algoritmo ECDSA). Si quisiéramos volver a tener acceso con una clave que ya hemos borrado del fichero `authorized_keys`, obviamente tendríamos que, manualmente o mediante algún mecanismo como `ssh-copy-id`, volver a copiar la clave pública asociada en dicho fichero.

Por otra parte, el **fichero `known_hosts`** incluye las claves públicas de los equipos a los que nos hemos conectado como clientes. Como vimos en el primer intento de conexión que realizamos al servidor, éste nos preguntaba si confiábamos en la clave pública del servidor. Al contestarle que sí lo que hizo ssh fue copiar la clave pública del servidor en este fichero `known_host`, de tal forma que las sucesivas veces que nos hemos conectado ya no ha sido necesario que el cliente ssh nos pregunte esto, pues ya encuentra la clave pública asociada a la clave privada que encuentra en el servidor durante la fase de negociación, como ya explicamos en su momento. Ahora bien, sabiendo esto el proceso lógico a seguir es, de alguna forma segura obtener la clave pública de los servidores a los que nos quisiéramos conectar y copiarla en el fichero `known_host`, **para que así desde el primer intento de conexión el cliente ssh pueda confiar automáticamente en que la máquina a la que se está intentando conectar es la correcta** y procedería a la fase de autenticar el usuario. El formato en el que se guardan las claves en este fichero es el siguiente:

```
<hash_ip_remota> <clave_publica_remota>
```

La dirección IP de la máquina remota se almacena indirectamente mediante su hash para que, si este archivo cae en manos de alguna persona ilegítima no pudiera saber con certeza a qué servidor corresponde la clave⁴. Sin embargo, este comportamiento puede cambiarse en la configuración del cliente SSH para que almacene directamente la dirección IP o el dominio del servidor si se desea.

El comando `ssh-keygen` también nos ofrece la posibilidad de interactuar con el fichero `known_hosts`. Podemos por ejemplo conocer si una determinada IP remota se encuentra registrada en nuestro fichero `known_hosts` mediante el comando:

```
ssh-keygen -F <ip>:<puerto>
```

```
C:\Users\mnava\.ssh>ssh-keygen -F 192.168.0.26
# Host 192.168.0.26 found: line 2
192.168.0.26 ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBMSr4e
o3UyUpS5LGp2zTnI5YL9PcwiMrV4iWz1SDcqrjxusF7kmqyEn90da45EHcFrLXLmRzHyDQ5FJaNfYm4=
```

También podremos eliminar la clave pública asociada a una máquina remota que ya estuviera registrada en nuestro fichero `known_host`, mediante la opción `-R`. Al hacer esto automáticamente y por si acaso nos guarda una copia del fichero `known_host` con extensión `.old`.

```
C:\Users\mnava\.ssh>ssh-keygen -R 192.168.0.26
# Host 192.168.0.26 found: line 2
C:\Users\mnava\.ssh\known_hosts updated.
Original contents retained as C:\Users\mnava\.ssh\known_hosts.old
C:\Users\mnava\.ssh>
```

5.7 FORWARDING

El concepto de Forwarding de SSH es un aspecto más avanzado que engloba una serie de **técnicas que nos permiten reenviar algo al equipo que nos conectamos por SSH para hacer uso de una funcionalidad adicional**. Vamos a ver dos casos concretos de Forwarding que se suelen utilizar habitualmente: el `ForwardAgent` y el `X11Forwarding`.

El **ForwardAgent** es una utilidad de ssh que nos permite **redireccionar el ssh-agent de la máquina local a otras máquinas remotas para poder acceder a otros hosts que sean inaccesibles** desde la máquina origen pero **sin tener que almacenar la clave privada de los hosts destinos en la máquina intermedia** a la que nos conectamos. Esto parece enrevesado, pero es bastante habitual. Pensemos en un escenario en el que existe una red de varios equipos y solo uno de ellos (denominado habitualmente como bastion) tiene habilitado el puerto 22 de ssh al exterior y puede comunicarse con el resto de los hosts internos de la red. Si yo quisiera acceder con un equipo fuera de dicha red a alguno de los hosts que hay dentro de la red que no sea el equipo bastión tendría que, en primer lugar, abrir una conexión por ssh con el equipo bastión y, una vez allí abrir otra Shell remota con el host final al que quiero acceder. Si no tuviéramos activado el `ForwardAgent` (por defecto está desactivado, por lo que hay que habilitar la propiedad `ForwardAgent` en el archivo de configuración `/etc/ssh/ssh_config`) debería de disponer o bien de la contraseña del usuario o de la clave privada disponible físicamente en el equipo bastión para poder acceder al host final

⁴ Como ya comentamos al principio del documento, los códigos hash generados mediante una función resumen son irreversibles, es decir no podríamos nunca obtener la dirección IP a través de su hash.

tal y como hemos visto hasta ahora, lo cual desde el punto de vista de la seguridad de estas claves privadas no sería recomendable.

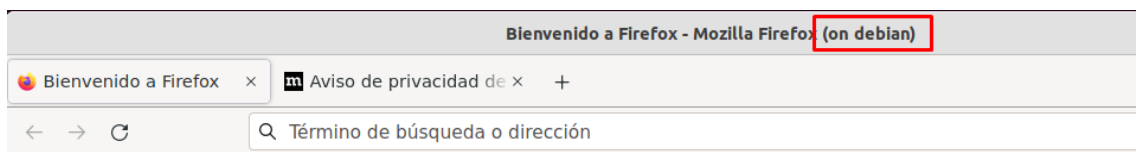
```
# test of available options, their meanings and defaults, please
# ssh_config(5) man page.

Include /etc/ssh/ssh_config.d/*.conf

Host *
  ForwardAgent yes
  # ForwardX11 no
  # ForwardX11Trusted yes
  # PasswordAuthentication yes
  # HostbasedAuthentication no
  # GSSAPIAuthentication no
```

Para habilitarlo simplemente borramos el carácter # al principio de la fila

Por otro lado, el **X11Forwarding** nos permite a través de ssh la **ejecución de una aplicación gráfica remota pero que se visualice en la máquina local**. Para ello nuevamente tenemos que configurarlo en el cliente ssh, concretamente habilitando la propiedad **ForwardX11** en el mismo fichero de configuración de antes. A partir de entonces podemos abrir una Shell remota e invocar a la aplicación gráfica que deseemos. La aplicación se ejecutará en la máquina remota pero se visualizará en la máquina local con la que estamos conectados. Por ejemplo si ejecutamos Firefox podremos reconocer en el título de la barra superior que la aplicación se está ejecutando en la máquina remota (en nuestro caso llamada “debian”).



6 CONFIGURACIÓN DE SSH

En este capítulo pasamos a ver un poco más en detalle algunos parámetros de configuración que podemos modificar para cambiar ciertos aspectos de la conexión con SSH. No podemos examinar uno a uno todos los parámetros que existen, ya que existen muchísimos parámetros distintos y además muchos de ellos son para un uso más avanzado, lo cual creemos que queda fuera del objetivo de este documento, pero sí vamos a detenernos en revisar aquellos que son más significativos. Dividiremos este apartado en dos partes: la configuración en el cliente y la configuración del servidor SSH.

6.1 CONFIGURACIÓN DEL CLIENTE SSH

La configuración del cliente ya la hemos trabajado sin saberlo en el apartado de *Forwarding*. Y es que esta configuración se guarda en el fichero `/etc/ssh/ssh_config`.

```
# Configuration data is parsed as follows:
# 1. command line options
# 2. user-specific file
# 3. system-wide file
# Any configuration value is only changed the first time it is set.
# Thus, host-specific definitions should be at the beginning of the
# configuration file, and defaults at the end.

# Site-wide defaults for some commonly used options. For a comprehensive
# list of available options, their meanings and defaults, please see the
# ssh_config(5) man page.

Include /etc/ssh/ssh_config.d/*.conf

Host *
# ForwardAgent no
# ForwardX11 yes
# ForwardX11Trusted yes
# PasswordAuthentication yes
# HostbasedAuthentication no
# GSSAPIAuthentication no
# GSSAPIDelegateCredentials no
# GSSAPIKeyExchange no
# GSSAPITrustDNS no
# BatchMode no
# CheckHostIP yes
# AddressFamily any
# ConnectTimeout 0
# StrictHostKeyChecking ask
# IdentityFile ~/.ssh/id_rsa
# IdentityFile ~/.ssh/id_dsa
# IdentityFile ~/.ssh/id_ecdsa
# IdentityFile ~/.ssh/id_ed25519
# Port 22
# Ciphers aes128-ctr,aes192-ctr,aes256-ctr,aes128-cbc,3des-cbc
# MACs hmac-md5,hmac-sha1,umac-64@openssh.com
# EscapeChar ~
# Tunnel no
# TunnelDevice any:any
# PermitLocalCommand no
# VisualHostKey no
# ProxyCommand ssh -q -W %h:%p gateway.example.com
# RekeyLimit 1G 1h
# UserKnownHostsFile ~/.ssh/known_hosts.d/%k
SendEnv LANG LC_*
HashKnownHosts yes
GSSAPIAuthentication yes
```

Este archivo consta de una lista de parámetros (uno por cada fila) a los cuales se le asigna un valor que dependerá del tipo de parámetro, normalmente siendo un valor booleano (sí/no) para activar o desactivar cierta característica o funcionalidad. Aquí encontramos entre otros los parámetros que hemos visto en la sección anterior: **ForwardAgent** y **ForwardX11**. Siempre que veamos una línea que comienza por el carácter almohadilla (quizás más conocido en la actualidad como hashtag, #) significa que esa línea está comentada, y por tanto el valor que se indique para el parámetro de esa línea no tiene efecto alguno en la configuración de SSH, digamos que es como si no existiera, por lo cual para esos casos SSH lo que hará será coger el valor por defecto para los parámetros que aparezcan comentados. Si abrimos un fichero

ssh_config por defecto, es decir sin haber sido modificado, encontraremos que los parámetros que se encuentran activos por defecto son los siguientes:

El parámetro `SendEnv` sirve para indicarle al cliente SSH **qué variables de entorno queremos enviar desde la máquina cliente a la máquina remota**. Por defecto como vemos se encuentran mencionadas las variables `LANG` y `LC_*`. Por ejemplo, la primera de ellas sirve para establecer el **idioma** y el **juego de caracteres** que se quiere utilizar. Por tanto, si nos conectamos a una máquina remota donde está disponible esta configuración que nosotros indicamos en la variable de entorno `LANG` (ya que puede ser que esa máquina solo tenga instalado por ejemplo el inglés, en ese caso no nos podría mostrar el que le indicamos porque no está disponible) aparecerá el mismo valor que tenemos declarado en la variable de entorno de la máquina origen.

¿Por qué se incluye esto en la configuración del cliente SSH? Porque evidentemente resulta muy cómodo que si yo me conecto a una máquina remota siga utilizando el mismo idioma que tengo definido en mi equipo, ya que por defecto en la máquina remota podría estar definido, por ejemplo, el inglés, el ruso o el chino... De forma análoga me puede interesar llevarme otras variables de entorno de mi máquina cliente a las máquinas remotas donde me quiera conectar.

Otro de los parámetros que encontramos activados por defecto en el cliente ssh es `HashKnownHosts`. Si recordáis explicamos que en el archivo `known_hosts` del cliente ssh se almacenan las claves públicas de los servidores a los que nos hemos conectado. El primer parámetro de esta clave era el código hash del servidor (que podría referenciarse por su dirección IP o por su nombre de dominio), pues bien, esta directiva de la configuración **nos permite elegir que se almacene el código hash de esa dirección o directamente la dirección en texto claro**, lo cual resulta menos seguro pero más legible para el administrador.

El último parámetro que encontramos activado en la configuración por defecto del cliente es el `GSSAPIAuthentication`. Este está relacionado con la **autenticación mediante el mecanismo GSSAPI**, lo cual se utiliza en entornos muy específicos y peculiares. De hecho, si buscáis información al respecto se suele recomendar deshabilitar este tipo de autenticación porque puede influir un poco en el rendimiento durante el proceso de inicio de sesión remota. Sin embargo, si se utiliza el método de autenticación por clave pública esto resulta irrelevante, ya que este método tiene más prioridad que el método GSSAPI, con lo cual como se realiza la autenticación antes de entrar en juego el proceso de autenticación por GSSAPI no influye para nada en el rendimiento del proceso de autenticación. Pese a esto, si deseamos deshabilitar este parámetro tampoco influiría en nada en nuestro caso.

Otros parámetros interesantes que no están activos por defecto en la configuración son:

- **EscapeChar** sirve para definir el carácter de escape que nos permitirá interrumpir la conexión con la máquina remota si hay algún problema de conexión por cualquier motivo (Por defecto es `Alt gr + 4 + .`).
- **GlobalKnownHostFile** para definir un fichero `known_host` global para todos los usuarios.
- **NumberOfPasswordPrompts** nos permite elegir el número de veces que pide la contraseña antes de rechazar la conexión, por defecto es 3. Se define un valor en el cliente y otro en el servidor, por lo que siempre se aplicará el menor de ellos.
- **StrictHostKeyChecking** gestiona el comportamiento del cliente cuando se intenta conectar a un equipo remoto del cual no conoce su clave pública. Por defecto su valor es `ask`, pues como ya sabemos si el cliente no reconoce la clave pública del servidor pregunta al usuario directamente qué debe hacer (establecer la conexión o rechazarla), pero podemos definir que automáticamente acepte la conexión (valor `no`) o la rechace (valor `yes`).

Estas configuraciones, además de establecerse de forma global en el fichero `/etc/ssh/ssh_config` también se pueden establecer a nivel de usuarios dentro del directorio `.ssh` que encontramos en la carpeta personal de cada usuario, de forma que **cada usuario puede personalizar sus preferencias de conexión** con los distintos hosts con los que trabaje. Por ejemplo en este fichero podríamos configurar qué claves de las que dispone el usuario debe usarse con su respectivo host remoto, así como el usuario al que nos queremos conectar, de forma que esta información no tendríamos que indicarla explícitamente en el comando SSH cada vez que queramos conectarnos a nuestras máquinas remotas. Por ejemplo para hacer esto podríamos escribir nuestro fichero de configuración del usuario de la siguiente forma:

```
Host 192.168.0.26
  HostName 192.168.0.26
  User mnavas
  PreferredAuthentications publickey
  IdentityFile ~/.ssh/id_ecdsa

Host 192.168.0.28
  HostName 192.168.0.28
  User mnavas
  PreferredAuthentications publickey
  IdentityFile ~/.ssh/id_rsa
```

Gracias a este fichero de configuración podremos conectar a las máquinas indicadas de una forma mucho más fácil y rápida, como se puede ver en la siguiente captura:

```
mnavas@Manuels-MacBook-Pro ~ % ssh 192.168.0.26
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-27-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

Se pueden aplicar 4 actualizaciones de forma inmediata.
Para ver estas actualizaciones adicionales, ejecute: apt list --upgradable

Last login: Fri Apr 29 00:36:22 2022 from 192.168.0.31
mnavas@vmubuntu:~$ exit
logout
Connection to 192.168.0.26 closed.
mnavas@Manuels-MacBook-Pro ~ % ssh 192.168.0.28
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-25-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

0 updates can be applied immediately.

Last login: Fri Apr 29 00:36:27 2022 from 192.168.0.31
mnavas@vbubuntu:~$
```

Por último, comentar que en determinadas circunstancias puede ser necesario pasarle al comando SSH algún parámetro de configuración. De forma que si necesitamos alguna característica en un momento puntual no es necesario tener que editar el archivo de configuración del usuario o el de configuración global para aplicar esta característica. Para hacer esto se utiliza la opción `-o`.

```
ssh -o "ForwardAgent yes" mnavas@192.168.0.26
```

Importante destacar que el orden de preferencia de SSH para aplicar configuraciones sigue el orden contrario a la explicación que hemos realizado. Es decir siempre que indiquemos cualquier configuración en el comando SSH siempre tendrá preferencia sobre las configuraciones del usuario y las configuraciones globales. Y a su vez, toda configuración que se indique en el fichero config de un usuario concreto tendrán preferencia sobre las posibles configuraciones que se estipulen en el fichero de configuración global `/etc/ssh/ssh_config`.

6.2 CONFIGURACIÓN DEL SERVIDOR SSH

Al igual que el cliente tiene una serie de parámetros de configuración el servidor SSH también presenta una serie de parámetros de configuración, que en esta ocasión siempre deberemos editar con permiso de superusuario y, al contrario que ocurría en el cliente donde podían establecerse distintas configuraciones por niveles (global, usuario, comando), en esta ocasión toda configuración del servidor se realizará en un único archivo de configuración del servicio SSH. Este fichero se encuentra alojado en `/etc/ssh/sshd_config`⁵.

Al igual que vimos con el fichero `ssh_config` del cliente, este archivo está compuesto por una serie de parámetros con valores asociados, aunque muchos de ellos no se estarán aplicando por estar comentados mediante el carácter “#”. Sin embargo es necesario recalcar que existen una serie de directivas que tienen un valor por defecto y que no aparecen en el fichero de configuración del servicio. Es decir, que los parámetros de configuración que se aplican al servidor SSH están compuestos por aquellos parámetros que están explícitamente indicados en el fichero de configuración más otros parámetros que encuentran implícitos por defecto, lo cual puede ser un poco confuso ya que estos parámetros no los vemos si revisamos el fichero `sshd_config`. Para facilitar por tanto la comprensión de esto disponemos del comando `sshd` que nos permite mediante la opción `-T` obtener por la salida del terminal todos aquellos parámetros que se están aplicando al servicio. Por ejemplo en la siguiente captura podemos ver los primeros parámetros de configuración que nos devuelve el comando `sshd` en el servidor SSH que estamos utilizando para las pruebas.

```
mnavas@vmubuntu:~$ sudo sshd -T
port 22
addressfamily any
listenaddress [::]:22
listenaddress 0.0.0.0:22
usepam yes
loggingracetime 120
x11displayoffset 10
maxauthtries 6
maxsessions 10
clientaliveinterval 0
clientalivecountmax 3
streamlocalbindmask 0177
permitrootlogin yes
ignorerhosts yes
```

La lista de parámetros a configurar en esta ocasión es incluso más larga que en el caso del cliente SSH. Nos vamos a detener un poco simplemente en comentar aquellos que resultan más relevantes:

- **Port:** indica el puerto donde el servicio escuchará las conexiones. Realmente utilizar un puerto distinto al 22 no aumenta significativamente la seguridad del sistema, ya que existen herramientas como `nmap` que permiten realizar escáneres y reconocer los puertos abiertos de una máquina. Pero puede servir para librarse de los habituales bots que existen en la red

⁵ En Windows, `sshd` lee los datos de configuración de `%programdata%\ssh\sshd_config` de forma predeterminada, aunque puedes especificar un archivo de configuración diferente si inicias `sshd.exe` con el parámetro `-f`. Si el archivo no está presente, `sshd` genera uno con la configuración predeterminada cuando se inicia el servicio.

intentando constantemente realizar ataques de fuerza bruta y derivados, con lo cual se libera un poco de ruido sobre el servicio.

- **PermitRootLogin:** permite elegir cómo podría acceder el usuario root al sistema por SSH.
- **PubKeyAuthentication:** indican si se permite el tipo de autenticación por clave pública y si se puede o no utilizar algún algoritmo de cifrado en concreto (Por ejemplo, RSAAuthentication para el RSA).
- **PermitEmptyPasswords:** podemos indicar que no se permita el acceso con contraseñas vacías.
- **X11Forwarding:** indica si el servidor acepta o no esta característica.
- **AddressFamily:** para indicar si se permite el acceso por ipv4, ipv6 o ambos.
- **ListenAddress:** para indicar por cuales de las interfaces de red de la que disponga el sistema se permite la conexión por SSH.

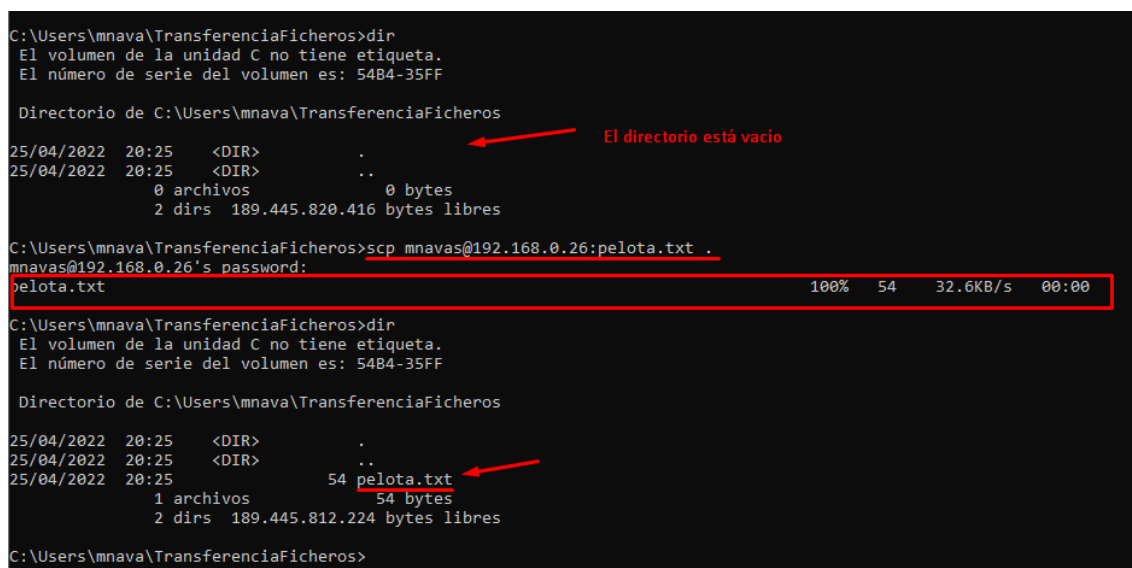
7 TRANSFERENCIA DE FICHEROS

En este apartado vamos a ver cómo podemos utilizar SSH como herramienta de transferencia de archivos. SSH no se utiliza sólo para conectarnos a un equipo remoto y gestionarlo mediante comandos, abrir aplicaciones de forma remota y demás, como hemos ido viendo a lo largo del documento, sino que además uno de los usos que resulta más útil para los usuarios de SSH es la transferencia de ficheros entre la máquina local y la remota. Existen dos comandos que podemos utilizar para llevar a cabo esta tarea, vamos a revisar ambos por separado:

El comando `scp` nos permite transferir un fichero entre un origen y un destino con una sintaxis sencilla. Tanto origen como destino se indican con la sintaxis `usuario@maquina:ruta`. Veamos el siguiente ejemplo, supongamos que tenemos un archivo `pelota.txt` en la carpeta personal del usuario `mnavas` de nuestra máquina remota, y queremos transferir dicho archivo al directorio actual de nuestra máquina local. Para ello ejecutaremos el siguiente comando:

```
scp mnavas@192.168.0.26:pelota.txt .
```

Con el comando anterior estamos indicando que queremos conectarnos **al usuario mnavas en la máquina remota con dirección IP 192.168.0.26**, y después de los dos puntos indicaremos la **ruta** (absoluta o relativa) **al fichero que queremos transferir**. En el destino indicamos simplemente un punto, indicando que deseamos traer el fichero que hemos indicado al directorio actual de nuestra máquina, donde estamos ejecutando el comando. En la siguiente captura se puede observar que en el directorio no existía ningún archivo, pero tras ejecutar el comando `scp` encontraremos el fichero que hemos transferido desde la máquina remota.



```
C:\Users\mnava\TransferenciaFicheros>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 54B4-35FF

Directorio de C:\Users\mnava\TransferenciaFicheros

25/04/2022  20:25  <DIR>          .
25/04/2022  20:25  <DIR>          ..
               0 archivos            0 bytes
               2 dirs  189.445.820.416 bytes libres

C:\Users\mnava\TransferenciaFicheros>scp mnavas@192.168.0.26:pelota.txt .
mnavas@192.168.0.26's password:
pelota.txt                                           100%  54   32.6KB/s   00:00

C:\Users\mnava\TransferenciaFicheros>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 54B4-35FF

Directorio de C:\Users\mnava\TransferenciaFicheros

25/04/2022  20:25  <DIR>          .
25/04/2022  20:25  <DIR>          ..
25/04/2022  20:25                54 pelota.txt
               1 archivos            54 bytes
               2 dirs  189.445.812.224 bytes libres

C:\Users\mnava\TransferenciaFicheros>
```

The screenshot shows a Windows command prompt window. The first part shows the directory contents of `C:\Users\mnava\TransferenciaFicheros` before the transfer, which is empty except for the current and parent directories. A red arrow points to the empty state with the text "El directorio está vacío". Then, the command `scp mnavas@192.168.0.26:pelota.txt .` is executed. The prompt asks for the password of `mnavas@192.168.0.26`. After the password is entered, the file `pelota.txt` is transferred, showing a progress bar at 100% with a size of 54 bytes and a speed of 32.6KB/s. Finally, the directory is shown again, and the file `pelota.txt` (54 bytes) is now present in the local directory. A red arrow points to the newly added file with the text "El directorio está vacío" (though the text in the image is "El directorio está vacío" pointing to the empty state before, and the second arrow points to the file after transfer).

Si quisiéramos hacerlo al revés, es decir enviar el fichero desde nuestro directorio a la máquina remota, tan solo debemos indicar como origen el fichero en nuestro directorio local e indicar como segundo parámetro la ubicación donde queremos transferirlo siguiendo la misma sintaxis `usuario@maquina:ruta`. Por ejemplo:

```
scp pelota_devuelta.txt mnavas@192.168.0.26:pelota2.txt
```

En este caso, si no indicamos nada en la ruta (tras los dos puntos) el archivo se guardaría con el mismo nombre que tuviera en el origen y dentro del directorio personal del usuario `mnavas`. Si quisiéramos guardarlo en otra ubicación bastaría con indicar simplemente la ruta como hemos indicado antes, por ejemplo `mnavas@192.168.0.26:./almacen/pelota2.txt` (Ojo: tener en cuenta que los

directorios que indiquemos ya deben existir, si en el ejemplo anterior no existiera el directorio almacen en el directorio home del usuario la ejecución del comando `scp` nos devolvería un error porque no puede encontrar la ruta especificada).

El comando `scp` también nos permite transferir archivos con recursividad, es decir indicándole que nos transfiera todos los archivos que contenga un directorio determinado, mediante la opción `-r`. Por ejemplo con el siguiente comando podríamos descargar a nuestra máquina local todos los archivos contenidos en la carpeta “Apuntes” que se encuentra en `/home/mnavas/` a nuestro directorio local:

```
scp -r mnavas@192.168.0.26:/home/mnavas/Apuntes/ .
```

```
C:\Users\mnava\TransferenciaFicheros>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 5484-35FF

Directorio de C:\Users\mnava\TransferenciaFicheros

25/04/2022  20:37    <DIR>          .
25/04/2022  20:25    <DIR>          ..
25/04/2022  20:37             113 pelota.txt
25/04/2022  20:37             113 pelota_devuelta.txt
                2 archivos                226 bytes
                2 dirs 189.475.262.464 bytes libres

C:\Users\mnava\TransferenciaFicheros>scp -r mnavas@192.168.0.26:/home/mnavas/Apuntes/ .
mnavas@192.168.0.26's password:
tema1.txt                                100% 0 0.0KB/s 00:00
tema2.txt                                100% 0 0.0KB/s 00:00
tema3.txt                                100% 0 0.0KB/s 00:00

C:\Users\mnava\TransferenciaFicheros>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 5484-35FF

Directorio de C:\Users\mnava\TransferenciaFicheros

25/04/2022  20:51    <DIR>          .
25/04/2022  20:25    <DIR>          ..
25/04/2022  20:51    <DIR>          Apuntes
25/04/2022  20:37             113 pelota.txt
25/04/2022  20:37             113 pelota_devuelta.txt
                2 archivos                226 bytes
                3 dirs 189.474.910.208 bytes libres

C:\Users\mnava\TransferenciaFicheros>dir Apuntes
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 5484-35FF

Directorio de C:\Users\mnava\TransferenciaFicheros\Apuntes

25/04/2022  20:51    <DIR>          .
25/04/2022  20:51    <DIR>          ..
25/04/2022  20:51             0 tema1.txt
25/04/2022  20:51             0 tema2.txt
25/04/2022  20:51             0 tema3.txt
                3 archivos                0 bytes
                2 dirs 189.473.857.536 bytes libres
```

El directorio solo contiene los archivos que hemos utilizado en el ejemplo anterior

Encontramos el directorio que hemos transferido por SSH desde nuestra máquina remota

Encontramos los mismos archivos que había en la máquina remota

De forma análoga procederíamos para subir de forma recursiva archivos desde nuestra máquina local a la máquina remota por SSH. Como se puede imaginar, podríamos incluso realizar transferencias de ficheros entre distintas máquinas remotas, siempre y cuando dispongamos del método de acceso correspondiente en ambas máquinas que participen en la transacción. Por ejemplo el siguiente comando nos copiaría el fichero “mifichero” desde la maquina 1 conectados como el usuario “usuario1” hasta el directorio personal del “usuario2” en la máquina 2:

```
scp usuario1@maquina1:mifichero usuario2@maquina2:
```

Como se puede comprobar las posibilidades del comando `scp` son muchas y muy útiles. Por ese motivo se trata de una herramienta indispensable para cualquier administrador de sistemas remotos.

El otro comando que nos permite realizar transferencia de archivos es **sftp**. Este comando no debe confundirse con el protocolo ftp ya que no tiene nada que ver, solamente utiliza el nombre porque realiza la transferencia de los datos al estilo de ftp pero todo se realiza sobre el protocolo ssh que estamos revisando. El comando `sftp` nos permite conectar por ssh con una máquina remota para que nos abra un prompt, como se podría hacer con cualquier cliente ftp clásico, y utilizar sus mismos comandos (`ls`, `get`,

put, etc) para realizar transferencias de archivos entre la máquina local y la remota. Es una forma de trabajar distinta a la que hemos visto con scp. Se puede utilizar mediante línea de comandos, pero también es destacable que este método puede implementarse en cualquier aplicación cliente con GUI que nos permita habilitar sftp (por ejemplo FileZilla), lo que nos permitirá utilizar el equivalente a ftp pero por un canal muy seguro como es SSH.

```
C:\Users\mnava\TransferenciaFicheros>sftp mnava@192.168.0.26
mnava@192.168.0.26's password:
Connected to 192.168.0.26.
sftp> ls
Apuntes Descargas Documentos Escritorio Imágenes
Música Plantillas Público Vídeos almacen
pelota.txt pelota2.txt pelota_devuelta.txt snap
sftp>
```

8 TÚNELES SSH

Los túneles SSH, también conocidos como *TCP Forwarding* o *SSH Port Forwarding* es un método avanzado del protocolo SSH que se utiliza para transportar datos arbitrarios a través de una conexión SSH cifrada, de forma que así se permiten que las conexiones realizadas a un puerto local sean reenviadas a una máquina remota a través de un canal seguro y viceversa. Se utiliza fundamentalmente en dos casos:

- Para utilizar un servicio saltándonos un cortafuegos, ya sea en el entorno del cliente o del servidor remoto. Básicamente se trata de definir el túnel a través de una máquina con SSH que conecta directamente con el equipo remoto al que queremos acceder, pero nos es imposible porque tendríamos que atravesar el cortafuegos.
- Para redirecciones en entornos donde existe un canal abierto pero inseguro, para aumentar la protección de la información que se quiere intercambiar con la máquina destino.

Para esto haríamos uso de lo que se conoce como un modo **LocalForwarding** de SSH, que lo que nos permite es, que en la máquina local se abre un puerto desde el cual se establece un túnel remoto a hasta un servidor SSH que a su vez tiene conectividad con el servicio final que yo quiero ver o utilizar.

Suponemos el siguiente escenario donde deseamos conectarnos a la máquina MV1, donde tenemos un servidor Apache con una página web publicada. Supongamos también que no tenemos acceso directo desde nuestra máquina local a dicha MV1, pero sin embargo sí tenemos acceso a otra máquina MV2. Pues bien, con este escenario podríamos abrir un túnel desde nuestra máquina local a MV2 y ésta a su vez se conectará directamente con el servidor web de la MV1, de forma que si accedemos al puerto local indicado tendremos un acceso directo al servicio que queremos obtener a través del túnel SSH. Para hacer esto utilizaremos el siguiente comando:

```
ssh -L <puerto_local>:<direccion_mv1>:<puerto_mv1> <usuarioMV2>@<direccion_mv2> -N
```

La opción **-L** nos permite indicar el puerto de la máquina local **<puerto_local>** que deseamos abrir y asociar al servicio que ofrece la MV1 indicada en el comando por **<dirección_mv1>**, concretamente en su puerto **<puerto_mv1>**. Como segundo argumento del comando le pasaremos el **usuario@maquina** a la cual queremos conectar mediante SSH y que nos transmitirá el servicio indicado de la MV1. La opción **-N** indica que no se ejecute un comando remoto, se utiliza simplemente para indicar que estamos invocando una redirección. De esta forma si tuviéramos por ejemplo un firewall entre la máquina local y MV1 no nos impediría establecer la conexión, pues estamos dirigiendo el servicio a través de la conexión SSH.

9 REFERENCIAS

<https://www.ssh.com/academy/ssh>

<https://websetnet.net/es/complete-guide-to-configuring-ssh-in-ubuntu/>

<https://www.chrisjhart.com/Windows-10-ssh-copy-id/>

<https://derechodelared.com/segundo-factor-de-autenticacion-ssh/>

● ANEXO I – ACTIVAR EL CLIENTE OPENSSH NATIVO EN WINDOWS 10

Para instalar los componentes de OpenSSH:

1. Abre **Configuración**, selecciona **Aplicaciones > Aplicaciones y características** y, a continuación, seleccione **Características opcionales**.
2. Examina la lista para ver si **OpenSSH** ya está instalado. Si no es así, en la parte superior de la página, selecciona **Agregar una característica** y, a continuación:
 - Busca **OpenSSH Client** (Cliente OpenSSH) y, a continuación, haga clic en **Instalar**
 - Busca **OpenSSH Server** (Servidor OpenSSH) y, a continuación, haga clic en **Instalar**

Una vez completada la instalación, vuelve a **Aplicaciones > Aplicaciones y características** y **Características opcionales** y debería ver **OpenSSH** en la lista.

Nota: La instalación del servidor OpenSSH creará y habilitará una regla de firewall denominada `OpenSSH-Server-In-TCP`. Esto permite el tráfico SSH entrante en el puerto 22. Si esta regla no está habilitada y este puerto no está abierto, las conexiones se rechazarán por el firewall.

Para más información al respecto consultar [la documentación de microsoft sobre OpenSSH en Windows](#).

● ANEXO II – HABILITAR SSH-AGENT EN WINDOWS 10

En el sistema operativo Windows la herramienta SSH-Agent no se encuentra habilitada por defecto. Para hacer esto solamente tenéis que seguir los siguientes pasos:

1. Abre la opción **Administrar características opcionales** en el menú de inicio y asegúrate de que tienes el cliente SSH abierto en la lista. Si no es así, deberías poder añadirlo.
2. Abre los **servicios** desde el menú de inicio
3. Desplázate hasta el **Agente de Autenticación OpenSSH > clic derecho > propiedades**.

| | | | | |
|------------------------------|---|-------------|-------------------|-----------------|
| NVIDIA LocalSystem Container | Container service for NVIDIA root features | En ejecu... | Automático | Sistema local |
| OneDrive Updater Service | Keeps your OneDrive up to date. | | Manual (dese... | Sistema local |
| OpenSSH Authentication Agent | Agent to hold private keys used for public key authentication | | Deshabilitado | Sistema local |
| Optimización de distribución | Realiza tareas de optimización de distribución de contenido | En ejecu... | Automático (in... | Servicio de red |
| Optimizar unidades | Ayuda al equipo a ejecutarse de manera más eficaz ... | | Manual | Sistema local |

4. Cambia el tipo de inicio de Desactivado a cualquiera de las otras 3 opciones. Por ejemplo en **Automático (Inicio retardado)**
5. Abre el cmd y teclea `where ssh` para confirmar que la ruta que os devuelve está en System32. Por ejemplo os debería mostrar algo parecido a la siguiente ruta
C:\Windows\System32\OpenSSH\ssh.exe. Si no lo está es posible que tengáis que cerrar y volver a abrir el cmd para que os aplique el cambio.
6. Por último ejecutad el comando `ssh-agent` para que comience a ejecutarse esta herramienta. Ahora podréis utilizar `ssh-add` para añadir la identidad que deseéis gestionar con ssh-agent:

```
C:\Users\mnava\.ssh>ssh-agent
C:\Users\mnava\.ssh>ssh-add id_rsa
Enter passphrase for id_rsa:
Identity added: id_rsa (mnava@lenovo)
C:\Users\mnava\.ssh>
```

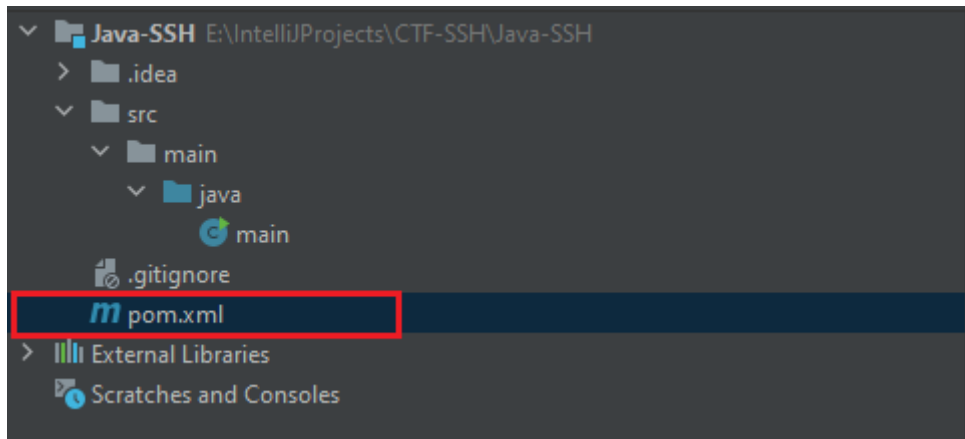
Podéis encontrar más información en este post de [StackOverflow](#).

● ANEXO III – SSH UTILIZANDO JAVA

Si queremos utilizar SSH mediante Java podemos hacer uso de la librería [JSch](#). Esta librería permite realizar conexiones a servidores mediante SSH, utilizar port forwarding, X11 forwarding, transferencia de ficheros, etc.

Para incluir esta librería en nuestro proyecto podemos hacerlo a través de Maven incluyendo la siguiente dependencia en el archivo pom.xml:

```
<dependency>
  <groupId>com.jcraft</groupId>
  <artifactId>jsch</artifactId>
  <version>0.1.55</version>
</dependency>
```



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>JSch_test</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>com.jcraft</groupId>
      <artifactId>jsch</artifactId>
      <version>0.1.55</version>
    </dependency>
  </dependencies>
</project>
```

La forma de utilizar esta librería es muy simple. En la clase donde la queramos utilizar deberemos de importar:

```
import com.jcraft.jsch.*;
```

A continuación, definimos los parámetros necesarios para realizar una conexión mediante SSH: host al que queremos conectarnos, usuario y contraseña con el que queremos acceder y puerto donde se encuentra el servidor SSH.

```
String host = "192.168.0.4";
String username = "user";
String password = "passwd";
int port = 22;
```

Después generamos una instancia de JSch y creamos una sesión de SSH utilizando los parámetros que hemos definido anteriormente.

```
JSch jsch = new JSch();

Session jschSession;
jschSession = jsch.getSession(username, host, port);
jschSession.setPassword(password);
```

IMPORTANTE: A diferencia de por consola, si el host al que nos queremos conectar no lo tenemos dentro de nuestro fichero de `known_hosts`, la conexión se rechazará automáticamente. Es por ello que es necesario deshabilitar la variable `StrictHostKeyChecking` de la configuración de la sesión. Esta variable es la que se encarga de comprobar que el host al que queremos acceder lo tenemos dentro de nuestro fichero `known_hosts`.

```
jschSession.setConfig("StrictHostKeyChecking", "no");
```

Por último, probamos a realizar la conexión

```
try {
    jschSession.connect();
    System.out.println("CONEXION ESTABLECIDA");
} catch (JSchException e) {
    System.out.println("FALLO EN LA CONEXION");
}
```

Cuando queramos cerrar la conexión se utiliza la función `disconnect()`.

```
jschSession.disconnect();
```

En el siguiente enlace tenéis disponible una plantilla de un proyecto en java (Java-SSH) para realizar una conexión mediante SSH: <https://github.com/navasdamas/CTF-SSH>

● ANEXO IV – TRANSFERENCIA DE FICHEROS MEDIANTE SFTP EN JAVA

En Java, la transferencia de ficheros mediante SSH se puede realizar utilizando SCP o SFTP. Utilizar SCP a través de java no es tan fácil como desde consola ya que se trabaja a nivel de bytes y el código se vuelve más complejo. Podéis ver como se haría en el siguiente enlace: <https://medium.com/@ldclakmal/scp-with-java-b7b7dbcd85>

Es por esto por lo que nosotros vamos a utilizar SFTP, un protocolo que, al igual que SCP, permite realizar operaciones sobre archivos remotos a través de una conexión SSH pero de una forma mucho más fácil desde Java. Este protocolo viene integrado en la librería de JSch por lo que no tendremos que incluir nada nuevo en nuestro proyecto de java. En este anexo vamos a explicar como descargar y subir archivos a través de esta librería

Para comenzar, una vez que hemos realizado la conexión SSH como hemos visto en el Anexo III, debemos crear un canal SFTP y conectarnos a él:

```
jschSession.connect();
System.out.println("CONEXION ESTABLECIDA");

ChannelSftp channelSftp = (ChannelSftp)
```



```
jschSession.openChannel("sftp");  
channelSftp.connect();
```

Ahora, en el caso de que queremos descargar un fichero, definimos la ruta del fichero que queremos descargar y la ruta de nuestro ordenador donde lo queremos almacenar:

```
String remoteFile = "prueba.txt";  
String localDir = "./";
```

Tras esto solo tenemos que ejecutar la función `get()` pasándole los parámetros que hemos definido:

```
channelSftp.get(remoteFile, localDir + "jschFile.txt");
```

Si por el contrario queremos subir un fichero, debemos de definir la ruta local donde se encuentra el fichero y el directorio remoto donde queremos subirlo. Después, en lugar de utilizar la función `get()` utilizaríamos la función `put()`:

```
String localFile = "prueba2.txt";  
String remoteDir = "./";  
channelSftp.put(localFile, remoteDir + "jschFile2.txt");
```

Por último, debemos acordarnos de cerrar la conexión con el canal SFTP y, después, cerrar la conexión SSH:

```
channelSftp.exit();  
jschSession.disconnect();
```

● ANEXO V – INSTALACIÓN DE 2FA PARA SSH

Vamos a instalar el factor de doble autentificación en el servicio de SSH de nuestro equipo remoto. Para ello, primero debemos de acceder a nuestro equipo e instalar el paquete: **libpam-google-authenticator**

Desde Ubuntu Server se hace con el siguiente comando en la terminal:

```
sudo apt install libpam-google-authenticator
```

Ahora debemos de configurar el módulo de 2FA para SSH. Para hacer esto tenemos que editar el fichero **/etc/pam.d/sshd** que es donde se encuentra la configuración de los módulos PAM (Pluggable Authentication Module) que afectan al servicio SSH. Un módulo PAM es una infraestructura para dar soporte a diferentes módulos de autentificación, en nuestro caso, para 2FA.

Podemos editar el fichero utilizando el editor **nano** que se encuentra integrado en el terminal de Ubuntu.

```
sudo nano <ruta_fichero>
```

```
moreno@ubuntu:~$ sudo nano /etc/pam.d/sshd
```

```
GNU nano 6.2 /etc/pam.d/sshd
# PAM configuration for the Secure Shell service

# Standard Un*x authentication.
@include common-auth

# Disallow non-root logins when /etc/nologin exists.
account    required    pam_nologin.so

# Uncomment and edit /etc/security/access.conf if you need to set complex
# access limits that are hard to express in sshd_config.
# account   required    pam_access.so

# Standard Un*x authorization.
@include common-account

# SELinux needs to be the first session rule. This ensures that any
# lingering context has been cleared. Without this it is possible that a
# module could execute code in the wrong domain.
session [success=ok ignore=ignore module_unknown=ignore default=bad]      pam_selinux.so close

# Set the loginuid process attribute.
session   required    pam_loginuid.so

# Create a new session keyring.
session   optional    pam_keyinit.so force revoke

[ Read 55 lines ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo     M-A Set Mark
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line  M-E Redo     M-G Copy
```

Con el editor nano podemos abrir un fichero en consola y editarlo. Para añadir el módulo de 2FA nos vamos al final del fichero e incluimos la siguiente línea:

```
auth required pam_google_authenticator.so
```

```
# This includes a dynamically generated part from /run/motd.dynamic
# and a static (admin-editable) part from /etc/motd.
session optional pam_motd.so motd=/run/motd.dynamic
session optional pam_motd.so noudate

# Print the status of the user's mailbox upon successful login.
session optional pam_mail.so standard noenv # [1]

# Set up user limits from /etc/security/limits.conf.
session required pam_limits.so

# Read environment variables from /etc/environment and
# /etc/security/pam_env.conf.
session required pam_env.so # [1]
# In Debian 4.0 (etch), locale-related environment variables were moved to
# /etc/default/locale, so read that as well.
session required pam_env.so user_readenv=1 envfile=/etc/default/locale

# SELinux needs to intervene at login time to ensure that the process starts
# in the proper default security context. Only sessions which are intended
# to run in the user's context should be run after this.
session [success=ok ignore=ignore module_unknown=ignore default=bad]      pam_selinux.so open

# Standard Un*x password updating.
@include common-password
auth required pam_google_authenticator.so

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo     M-A Set Mark
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line  M-E Redo     M-G Copy
```

Para guardar pulsamos **Ctrl + x**. Le decimos que queremos guardar lo que hemos modificado pulsando **y**, después le damos **Enter** para que lo guarde con el mismo nombre.

Ahora vamos a modificar el fichero de configuración de SSH para que pida el código de 2FA. El fichero de configuración es **/etc/ssh/sshd_config**. Lo abrimos con el editor nano, buscamos el parámetro que indico a continuación y lo ponemos a **yes**. Si el parámetro se encuentra comentado con **#**, lo descomentamos:

```
ChallengeResponseAuthentication
```

IMPORTANTE: Dependiendo de la distribución de Linux que estemos utilizando puede que el parámetro anterior no exista, entonces debemos buscar:

KbdInteractiveAuthentication

```
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!
#PasswordAuthentication yes
#PermitEmptyPasswords no

# Change to yes to enable challenge-response passwords (beware issues with
# some PAM modules and threads)
KbdInteractiveAuthentication yes

# Kerberos options
#KerberosAuthentication no
#KerberosOrLocalPasswd yes
#KerberosTicketCleanup yes
#KerberosGetAFSToken no

# GSSAPI options
#GSSAPIAuthentication no
#GSSAPICleanupCredentials yes
#GSSAPIStrictAcceptorCheck yes
#GSSAPIKeyExchange no
```

Guardamos los cambios y reiniciamos el servicio de SSH utilizando:

```
sudo systemctl restart ssh
```

Podemos ver que todo funciona correctamente ejecutando:

```
sudo systemctl status ssh
```

```
moreno@ubuntu:~$ sudo systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2022-04-30 13:46:08 UTC; 25s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
  Process: 13459 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
 Main PID: 13461 (sshd)
    Tasks: 1 (limit: 2249)
   Memory: 1.7M
      CPU: 18ms
   CGroup: /system.slice/ssh.service
           └─13461 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

abr 30 13:46:08 ubuntu systemd[1]: Stopping OpenBSD Secure Shell server...
abr 30 13:46:08 ubuntu systemd[1]: ssh.service: Deactivated successfully.
abr 30 13:46:08 ubuntu systemd[1]: Stopped OpenBSD Secure Shell server.
abr 30 13:46:08 ubuntu systemd[1]: Starting OpenBSD Secure Shell server...
abr 30 13:46:08 ubuntu sshd[13461]: Server listening on 0.0.0.0 port 22.
abr 30 13:46:08 ubuntu sshd[13461]: Server listening on :: port 22.
abr 30 13:46:08 ubuntu systemd[1]: Started OpenBSD Secure Shell server.
```

IMPORTANTE: Ahora debemos de iniciar sesión con el usuario con el cual queremos configurar el 2FA. Si hay varios usuarios los siguientes pasos hay que repetirlos para cada uno de ellos.

Una vez hemos iniciado sesión con el usuario deseado abrimos el terminal y ejecutamos:

```
google-authenticator
```

Nos va a preguntar si queremos que el token esté basado en el tiempo, es decir, que el token cambie cada cierto tiempo basándose en la fecha y hora actual. Vamos a indicar que sí queremos escribiendo **y**

```
Do you want authentication tokens to be time-based (y/n) y
```

A continuación, se nos muestra un código QR que deberemos de escanear con nuestra aplicación de 2FA preferida e introducir el código que nos da la app.



Tras esto se nos van a hacer las siguientes preguntas:

- Si queremos actualizar el fichero de google authenticator de nuestro usuario. A esta decimos que sí escribiendo **y**

```
Do you want me to update your "/home/moreno/.google_authenticator" file? (y/n) y
```

- Si queremos que un token solo pueda ser utilizado una vez. Esto restringe el inicio de sesión a un login cada 30 segundos aproximadamente que es el tiempo que pasa hasta que se genera un nuevo token. Indicamos que sí.

```
Do you want to disallow multiple uses of the same authentication token? This restricts you to one login about every 30s, but it increases your chances to notice or even prevent man-in-the-middle attacks (y/n) y
```

- Si queremos introducir varios tokens. El token anterior al que se ha generado y el token siguiente. Esto se hace para poder compensar la posible desincronización que haya entre el cliente y el servidor. Nosotros vamos a indicar que no, solo queremos iniciar sesión introduciendo un único token.

```
By default, a new token is generated every 30 seconds by the mobile app.
In order to compensate for possible time-skew between the client and the server,
we allow an extra token before and after the current time. This allows for a
time skew of up to 30 seconds between authentication server and client. If you
experience problems with poor time synchronization, you can increase the window
from its default size of 3 permitted codes (one previous code, the current
code, the next code) to 17 permitted codes (the 8 previous codes, the current
code, and the 8 next codes). This will permit for a time skew of up to 4 minutes
between client and server.
Do you want to do so? (y/n) n
```

- Por último, nos pregunta si queremos establecer un límite de intentos para así protegernos contra ataques de fuerza bruta. Activando esta opción limitamos los intentos de login a 3 cada 30 segundos. Vamos a indicar que sí.

```
If the computer that you are logging into isn't hardened against brute-force
login attempts, you can enable rate-limiting for the authentication module.
By default, this limits attackers to no more than 3 login attempts every 30s.
Do you want to enable rate-limiting? (y/n) y
```

Y listo, ya tendríamos configurado SSH con 2FA para el usuario deseado. Ahora cuando queramos conectarnos nos va a pedir el código de doble autenticación.

```
PS C:\Users\jemoa> ssh moreno@192.168.18.167
Password:
Verification code: Ahora nos pide el código para poder iniciar sesión
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-27-generic x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/advantage

System information as of sáb 30 abr 2022 14:11:05 UTC

System load:  0.0               Processes:            105
Usage of /:   45.4% of 9.75GB   Users logged in:     1
Memory usage: 12%              IPv4 address for enp0s3: 192.168.18.167
Swap usage:   0%

* Super-optimized for small spaces - read how we shrank the memory
  footprint of MicroK8s to make it the smallest full K8s around.

https://ubuntu.com/blog/microk8s-memory-optimisation

0 updates can be applied immediately.

Last login: Sat Apr 30 13:16:37 2022 from 192.168.18.131
moreno@ubuntu:~$
```

● ANEXO VI – INSTALACIÓN Y USO DE NMAP

Nmap es un programa de código abierto que sirve para realizar rastreos de puertos en equipos. Es muy útil cuando queremos saber que puertos tiene abiertos un dispositivo o en qué puerto está ejecutándose un determinado servicio.

Para utilizar Nmap en Windows basta con bajarnos el instalador desde su página web e instalarlo en nuestro equipo: [Nmap](#)

En el caso de Linux podemos instalarlo ejecutando en consola:

```
sudo apt-get install nmap
```

Para utilizarlo debemos de abrir un terminal y escribir:

```
nmap <dirección-ip>
```

Esto analizará los **puertos más comunes** de la dirección ip introducida y nos dirá cuales tiene abiertos. Además, nos indicará el servicio que **considera** que se está ejecutando en dicho puerto.

IMPORTANTE: La lista de los puertos más comunes que analiza la podéis encontrar en el siguiente enlace: <https://unix.stackexchange.com/questions/238640/nmap-doesnt-appear-to-list-all-open-ports>.

En cuanto al servicio, por defecto Nmap le asigna un nombre de servicio a cada puerto teniendo en cuenta la siguiente lista: <https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml>

```
PS C:\Users\jemoa> nmap 192.168.18.131
Starting Nmap 7.80 ( https://nmap.org ) at 2022-04-30 17:35 Hora de verano romance
Nmap scan report for host.docker.internal (192.168.18.131)
Host is up (0.00013s latency).
Not shown: 995 closed ports
PORT      STATE SERVICE
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
1039/tcp  open  sbl
7070/tcp  open  realserver

Nmap done: 1 IP address (1 host up) scanned in 0.50 seconds
```

Si queremos forzar que Nmap busque en todos los puertos de un rango dado, podemos hacerlo con el siguiente comando:

```
nmap -p <puerto-inicio>-<puerto_fin> <dirección-ip>
```

También podemos limitar el escaneo a un único puerto con:

```
nmap -p <puerto> <dirección-ip>
```

```
PS C:\Users\jemoa> nmap -p 0-400 192.168.18.131
Starting Nmap 7.80 ( https://nmap.org ) at 2022-04-30 17:48 Hora de verano romance
Nmap scan report for host.docker.internal (192.168.18.131)
Host is up (0.00012s latency).
Not shown: 398 closed ports
PORT      STATE SERVICE
135/tcp   open  msrpc
137/tcp   filtered netbios-ns
139/tcp   open  netbios-ssn

Nmap done: 1 IP address (1 host up) scanned in 1.71 seconds
```

Por último, podemos forzar a Nmap a que nos diga exactamente qué servicio se está ejecutando en cada puerto. Este comando tarda más en ejecutarse, pero nos puede ser útil si necesitamos encontrar un servicio en concreto que sabemos que no se está ejecutando en su puerto habitual.

```
nmap -sV <dirección-ip>
```

```
PS C:\Users\jemoa> nmap -sV 192.168.18.131
Starting Nmap 7.80 ( https://nmap.org ) at 2022-04-30 17:55 Hora de verano romance
Nmap scan report for host.docker.internal (192.168.18.131)
Host is up (0.00039s latency).
Not shown: 995 closed ports
PORT      STATE SERVICE          VERSION
135/tcp   open  msrpc            Microsoft Windows RPC
139/tcp   open  netbios-ssn     Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds    Microsoft Windows 7 - 10 microsoft-ds (workgroup: WORKGROUP)
1039/tcp  open  msrpc            Microsoft Windows RPC
7070/tcp  open  ssl/realserver?
Service Info: Host: MORENOSOBREMESA; OS: Windows; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 54.78 seconds
```

● ANEXO VI – PETICIONES GET

En las páginas web actuales, los clientes (también llamados navegadores) no solo obtienen un elemento HTML del servidor, sino que también envían información como la siguiente:

- El texto de búsqueda que el usuario ha escrito en el motor de búsqueda
- El contenido de los formularios
- El filtro de selección en tiendas online
- El orden de una lista

Para enviar ciertos tipos de información al servidor, el protocolo HTTP provee diferentes métodos de petición. Los dos más importantes son **GET** y **POST**, los cuales, aunque entregan los mismos resultados, revelan algunas diferencias entre ellos. Nosotros nos vamos a centrar en el método GET que se el que se va a utilizar durante el reto.

Con el método GET, los datos que se envían al servidor **se escriben en la misma dirección URL**. En la ventana del navegador, lo encontrarás así:

```
<protocolo>://<URL>:<puerto>/<ruta>/<parámetro>
```

Por ejemplo:

```
http://mi-server.com:8080/saludo/hola_server
```

Esto enviaría a la ruta /saludo del servidor el dato “hola_server”. Si en el servidor, la ruta especificada está configurada para recibir un parámetro de tipo string (como es “hola_server”), este procesará la petición y enviará al cliente una respuesta 200 (OK).

Desde Java se pueden lanzar peticiones GET utilizando la librería **Apache HttpClient** que se puede importar a través de Maven de la siguiente forma:

```
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpclient</artifactId>
  <version>4.5.13</version>
</dependency>
```

Para realizar la petición GET primero debemos de crearnos un cliente Http:

```
//Creamos un cliente  
final CloseableHttpClient httpClient = HttpClients.createDefault();
```

Después establecemos la ruta de la petición GET con la información que queremos enviar:

```
//Establecemos la ruta donde se quiere enviar la petición get  
HttpGet request = new HttpGet("http://mi-  
server.com:8080/saludo/hola_server");
```

Por último, realizamos la petición y recogemos e imprimimos la respuesta:

```
//Enviamos la petición y recogemos la respuesta  
try (CloseableHttpResponse response = httpClient.execute(request)) {  
  
    // Imprimimos el código de respuesta  
    System.out.println(response.getStatusLine().toString());  
  
    HttpEntity entity = response.getEntity();  
  
    //Imprimimos la información que nos envía el servidor  
    if (entity != null) {  
        // return it as a String  
        String result = EntityUtils.toString(entity);  
        System.out.println(result);  
    }  
}
```

En el siguiente enlace tenéis disponible una plantilla de un proyecto en java (Java_GET) para realizar peticiones GET: <https://github.com/navasdamas/CTF-SSH>