Reto SSH

6 marzo 2022

línea corta

# Objetivo

Un CTF es una competición que consiste en capturar una bandera escondida en una máquina expuesta en Internet. Vuestro objetivo será alcanzar un fichero (bandera 1) donde tendréis que escribir vuestro nombre y un servidor web (bandera 2) donde tendréis que subir el archivo *index.html* que queráis.

Con este reto se pretende que los alumnos adquieran algunos conocimientos de seguridad y, más concretamente, de SSH, así como conocimientos de otras ramas mientras superan una serie de pruebas que se han establecido en distintas máquinas virtuales del centro.

# Descripción

El reto consiste en realizar una serie de conexiones SSH en máquinas sucesivas hasta llegar a una máquina final donde poder grabar vuestro nombre y subir el fichero *index.html.* Para conseguirlo tendréis que superar una serie de pruebas que os permitirán acceder a cada máquina.

En [github](https://github.com/navasdamas/CTF-SSH) disponéis de un manual muy completo sobre SSH y las herramientas que se utilizan en este reto. Al final de este documento se os ha incluido un resumen con todo lo que necesitáis para superar las pruebas.

Para comenzar el reto necesitaréis crear una araña en Java que consiga descifrar el usuario y contraseña de la primera máquina, en ella encontraréis las instrucciones para acceder a la siguiente.

El primer alumno que grabe su nombre y suba el fichero index.html será el ganador del reto. Cuando consideréis que habéis terminado el reto, poneros en contacto con algunos de los profesores para que verifique que se ha superado correctamente.

# Instrucciones

En la primera prueba tendréis que acceder mediante SSH a una máquina del centro que se encuentra en la dirección **falcondptoinformatica.synology.me** y en el **puerto 22**. El usuario de la máquina es el nombre de uno de los profesores que os imparte clase: Eladio, Ana, Loli, Guillermo, Antonio, Carlos. La contraseña de este usuario se encuentra entre las 200 contraseñas más utilizadas en España en el año 2020: <https://nordpass.com/es/most-common-passwords-list/>

En [github](https://github.com/navasdamas/CTF-SSH) tenéis un fichero llamado *pass\_csv.csv* donde se encuentra este listado de contraseñas en formato csv.

Para superar esta prueba deberéis de crear una araña en Java que realice un ataque de diccionario probando con cada usuario y cada contraseña hasta que consigáis la combinación correcta. Este programa os imprimirá por pantalla cuáles son las credenciales correctas y deberéis de realizar la conexión SSH desde el terminal de vuestro equipo. Una vez dentro de la máquina, podéis abrir o descargarlos el fichero *instrucciones\_maquina\_2.txt* para conocer las instrucciones de la siguiente prueba.

# Material necesario

* [IDE Java](https://www.jetbrains.com/es-es/idea/download/#section=windows)
* [Java JDK](https://www.oracle.com/java/technologies/downloads/)
* Librería SSH de Java: [JSch](https://mvnrepository.com/artifact/com.jcraft/jsch/0.1.55)
* Cliente SSH
* [NMap](https://nmap.org/download)

# Pistas

## Activar el cliente OpenSSH en Windows 10/11

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](https://docs.microsoft.com/es-es/windows-server/administration/openssh/openssh_install_firstuse).

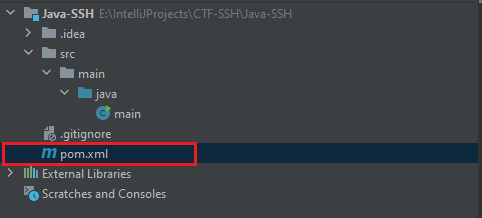
## Conexiones SSH con Java

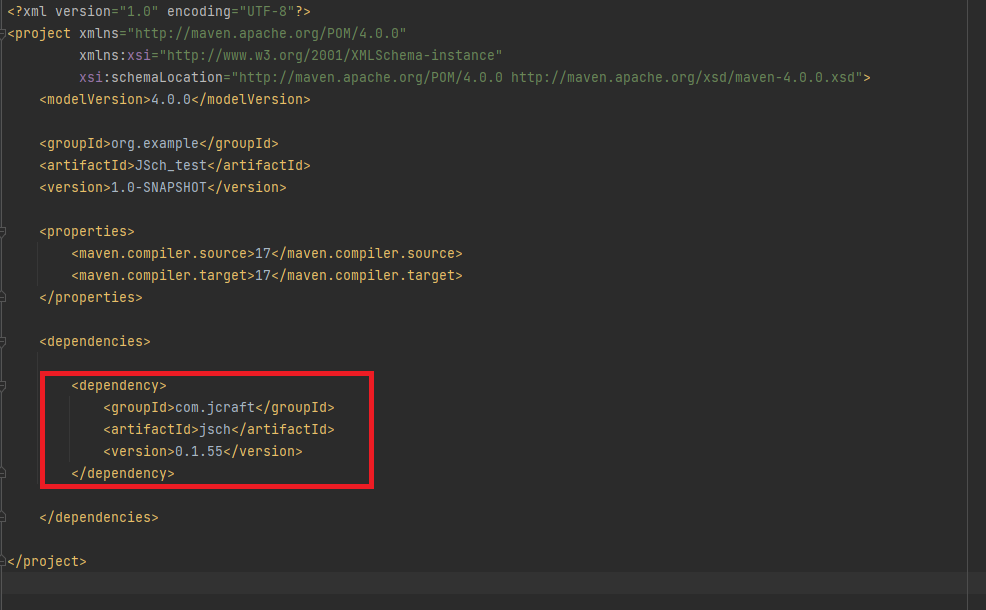
En [github](https://github.com/navasdamas/CTF-SSH) disponéis de un esqueleto de programa en Java llamado *Java-SSH* donde se muestra como se realiza una conexión SSH utilizando la librería JSch. A continuación, explicamos brevemente como se instala y utiliza esta librería.

Si queremos utilizar SSH mediante Java podemos hacer uso de la librería [JSch](http://www.jcraft.com/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>





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 *StrtictHostKeyChecking* 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();

## Conexiones SSH desde consola con usuario y contraseña

Para llevar a cabo la conexión SSH mediante consola utilizando usuario y contraseña 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:

Texto

Descripción generada automáticamente

## 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](https://nmap.org/download#windows)

En el caso de Linux podemos instalarlo ejecutando en consola:

sudo apt-get install nmap

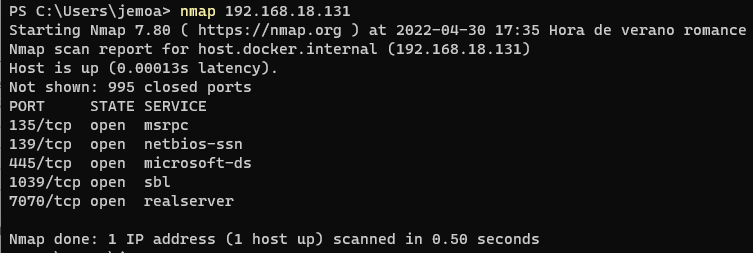
Para utilizarlo debemos de abrir un terminal y escribir:

namp **<**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>

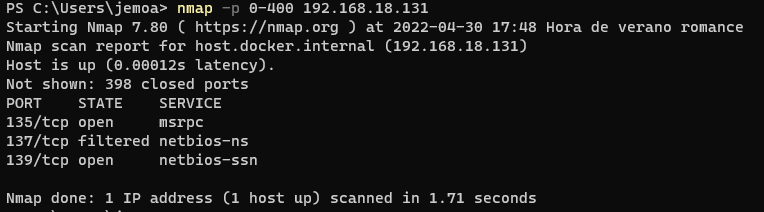


Si queremos forzar que Nmap busque en todos los puertos de un rango dado, podemos hacerlo con el siguiente comando:

namp -p **<**puerto-inicio**>-<**puerto\_fin**> <**dirección-ip**>**

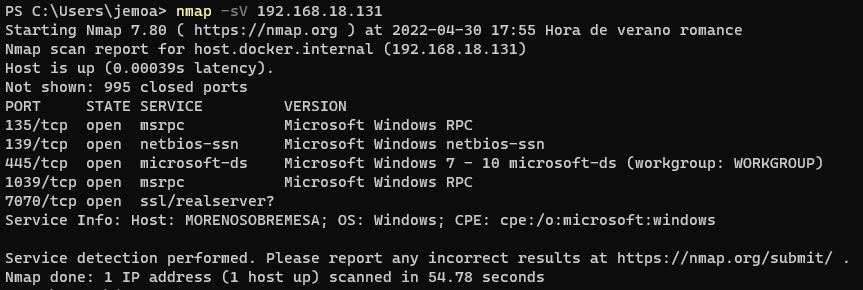
También podemos limitar el escaneo a un único puerto con:

namp -p **<**puerto**> <**dirección-ip**>**



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.

namp -sV **<**dirección-ip**>**



## Transferencia de ficheros mediante SCP

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.

Texto

Descripción generada automáticamente

## Transferencia de ficheros mediante en Java (SFTP)

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-b7b7dbcdbc85>

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 cómo 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();

## Peticiones POST