Voy a dejar de lado el examen de C que iba a suceder por la simple razón de que los ejercicios que di para la preparación me parecen suficientes. Lo pensé un par de veces pero no encontré un examen que de mejor pauta sobre lo que se aprendió.

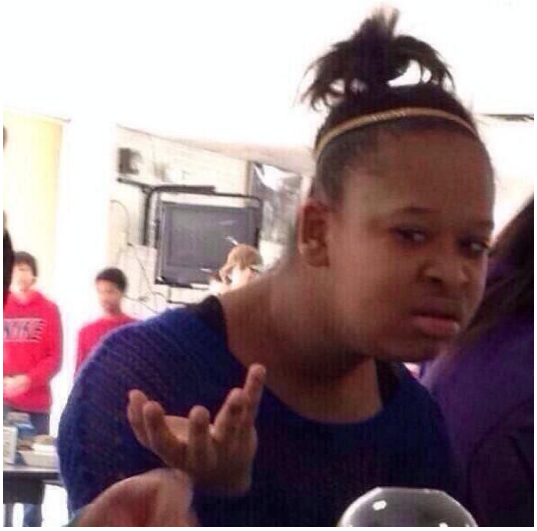
También me gustaría anunciar que estoy por hacerle cambios a la parte visual, con ayuda de De La Llave, quien fue el creador del logo de HDC. Son pequeños avances pero esperemos que sumen :).

Bueno, la realidad es que además de esto me estuve preguntando cuál es el tema siguiente. Por lo que no sé si se me desordenará un poco en este punto. Hoy vamos a encontrarnos con el famoso **SSH**.

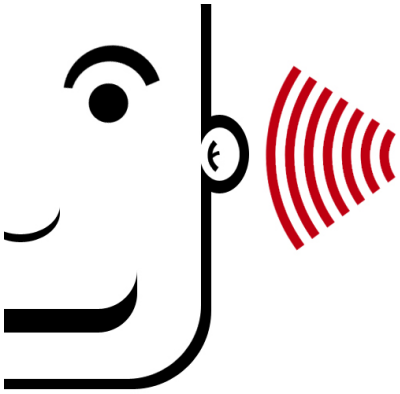
“¿Famoso? ¿Puedo pedirle un autógrafo?”

Vamos ¿Nunca has escuchado de **SSH** Manolo? Bueno, se trata de un **protocolo** para la **comunicación** entre dos **terminales** de forma **segura**. **SSH** hace referencia a **Secure SHell** y fue inventado con la idea orientada a la seguridad.

“O sea que lo usaremos porque somos hackers.”



No, SSH es usado por muchos **administradores de sistemas** como reemplazo de Telnet donde las comunicaciones eran inseguras para poder manejar servidores a **distancia**. Lo que pasaba con Telnet era que la comunicación entera iba en texto plano por la red. Es decir, si te parabas en la mitad de la comunicación entenderías todo lo que comunicaban ambas terminales-imaginen que en el medio habrán contraseñas, datos, rutas de archivos-.

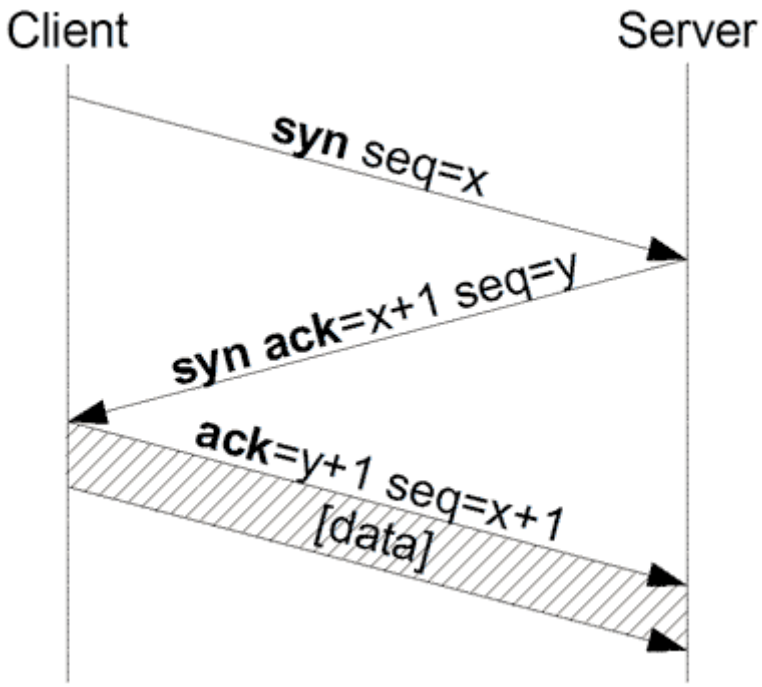


SSH, en cambio, tiene una **comunicación cifrada** y utiliza un método de **identificación** del server en el que resuelve el problema anterior en el que se hacía pasar rápidamente por otro ordenador -debe ser la primera vez que uso la palabra "ordenador"xD- simplemente tomando esa dirección IP como propia. Quisiera ver ese ataque prontamente en el curso, así que no voy a ahondar demasiado en el tema. Esta clase va más orientada al protocolo y la aplicación.

Veamos la comunicación entre un servidor SSH y un cliente SSH que se conecta al anterior paso por paso:

Al conectarse un cliente de SSH con el servidor, se realizan los siguientes pasos:

1. El cliente abre una **conexión TCP al puerto 22** del host servidor. Ya sabemos como funciona el **handshake**, no hace falta que profundice. Obviamente toda la conexión será **TCP**. Necesitamos que la comunicación se dé entera.



2. El cliente y el servidor **acuerdan la versión del protocolo a utilizar**, de acuerdo a su configuración y capacidades. Esto tiene que ver con que existen varias versiones de este protocolo que fueron cambiando para mejorar la seguridad. Si mal no recuerdo, hoy en día la actual es **SSH2**.

3. **El servidor posee un par de claves pública/privada de RSA** (llamadas ``claves de host''). **El servidor envía al cliente su clave pública.**

Acá quisiera hacer un **apartado** a la **criptografía de RSA** si es que tienen gustos por el análisis criptográfico pueden leerlo, así que voy a hacer una pausa a la explicación de SSH.

“¿Es métrico?”

Es “**simétrico**” Manolo, y no. **RSA es un sistema asimétrico**, compuesto por una clave privada y otra pública. Fue creado en 1977…

“Uh, no sirve más. Mejor que se jubile.”

Manolo, prestá atención. Se supone que el algoritmo **aún es muy seguro**. RSA se llama así por las iniciales de los 3 padres de este cifrado.

Ahora para explicar RSA enteramente **voy a agradecer a Unravel** que ha posteado una explicación básica y completa en el foro de “elhacker” y es de donde voy a basarme. **Lo primero** que tenemos que entender para comprender el RSA es la **aritmética modular**. No se asusten, es sólo un nombre y el tema no es complejo.

Supongamos que nos agarramos un **reloj**, lo miramos y ponemos la **aguja de las horas en el número 6.**

Para quien no conoce, miren la aguja verde :D

Si nosotros sumásemos **8 horas más**, debería marcar las **14**. Pero sabemos que el reloj **sólo puede marcar hasta 12 horas**, por lo que dará la vuelta. Luego de pasar las doce, vemos que esas 2 horas faltantes son las que quedan marcadas en el reloj.

Entonces **ahora el valor que tiene el reloj es de 2**. Esto sucede porque **tuvimos que comenzar nuevamente a contar luego de haber sobrepasado el número límite.** Se dice entonces que **el reloj es módulo 12**. Exactamente eso es la aritmética modular. Claro que nosotros, **incluimos el 0**. Entonces el reloj iría de 0 a 11 y las cuentas serían las mismas.

Veamos otro ejemplo:

Si tuviésemos la cuenta (**8 + 8**) pero le aplicásemos **módulo 7** (recuerden que hay 0, entonces iría del 0 al 6) , quiere decir que no se va a pasar una vez, sino **2 veces**.

“Siento que me estás tratando como un lelo.”

Lo siento, Manolo. Es para que no queden ningún tipo de dudas al respecto. Sigamos. **Si la cuenta da 16, una vez que se le aplica el módulo 7, el resultado sería de 2. La fórmula general sería:**

a = k*x + resto

Donde **a** es igual al **número que se le aplica el módulo**, **k es el módulo** que aplicamos, **x es un número entero que multiplica a k para llegar al máximo número posible**, **y el resto es lo que tenemos que sumar** (que siempre será menor al módulo) **para llegar a a**.

En nuestro caso la fórmula sería: **16 = 7*2 + 2.** Y el resultado del módulo siempre correspondería al resto, así que en realidad:

resto = a – k * x

Otro ejemplo: **supongamos el mismo módulo de 7, pero se aplica a la cuenta 7 + 8**.

resto = (7 + 8) – k * 7

resto = 15 – k * 7; k = 2

resto = 15 – 2 * 7 = 15 – 14 = 1

Es decir, que ahora **el resto es igual a 1**. Cuando sucede esto, se dice que **los números que se multiplican entre sí para llegar a 1 son inversos entre sí en ese módulo específico**. Entonces 7 y 8 son inversos en módulo 7.

Hay algo muy interesante aquí que son los números **primos relativos.**

"¿Vamos a ver al árbol genealógico completo de los números?"

No, esto nada tiene que ver con... Realmente estoy dudando de tus capacidades. **Los números primos son aquellos que sólo se dividen por si mismos o por uno,** como el 7, el 5, el 11, 13, 17, etc. Hay muchos. Lo importante ahora es que **los primos relativos son aquellos que entre sí no existen divisores en común** (o sea que pueda dividirlos a los 2) **menores a ambos números y exceptuando el 1**.

Ahora, se supone que hay una propiedad que dice que **un número *a* que es primo**

relativo con el módulo (en nuestro caso seguiremos usando el módulo 7), **entonces *a* tiene inversa dentro de ese módulo**. Justo dentro de módulo 7, **todos tienen inversa porque 7 es primo y eso declara que será primo relativo con cualquiera de los otros números.** ¿Van entendiendo? A todo esto, el cero no cuenta. Nunca cuenta, es como el marginado de los números :(.

“Creo que sí, aunque hasta ahora no cifré nada.”

Jajajaja, tranquilo. Sigamos: otro conocimiento importante es el **número phi (φ). φ declara la cantidad de números que tienen inversa en un módulo específico.** Con el ejemplo que estábamos viendo anteriormente, **φ(7) = 6** porque todos los números tenían inversa gracias a la propiedad de primos relativos (¿recuerdan? Fue hace un párrafo) y como el cero no cuenta, bueno pues lo dejamos de lado.

La fórmula génerica sería:

$$\varphi(n) = n - 1$$

recuerden que el “-1” es porque el cero está marginado y porque el número n es primo, ergo todos tienen inversa.

Ahora, **si *n*** (les recuerdo que para nosotros es 7) **está formado por dos números primos multiplicados n = p*q; siendo p y q primos** (no primos relativos), entonces:

$$\varphi(n) = (p-1) * (q-1)$$

Entonces nuestras claves para entender RSA son:

1. **resto = mod(a) = a – k * x**
2. **La propiedad entre a y n para que a tenga inversa**
3. si **n = p * q**, entonces:
4. **φ(n) = (p -1) * (q – 1)**

Ahora empieza lo divertido. La **generación de claves**. Para generar el par, primero se debe **elegir 2 números primos de gran valor**. Corresponderán a los valores **p y q**. Se multiplican para sacar n:

$$n = p * q$$

Luego calculamos **φ:**

$$\varphi(n) = (p - 1) * (q - 1)$$

Ahora elegimos un **número random que sea primo relativo con φ(n)**. Esto lo elegimos así **para que tenga un número inverso**. **Ese valor nuevo es *e* y la clave pública será el par (e,n)**. **Al número inverso** de e lo llamamos ***d*** y será nuestra **clave privada. Phi** sólo nos sirvió para generar las claves, por lo que podemos olvidarnos de él tranquilamente. En realidad, **debemos eliminarla ya que es el generador de nuestras claves y si alguien se apodera de φ, entonces podría generar nuestra clave privada a través de nuestra clave pública**. Interesante, pero nosotros no queremos ese embrollo.

“Pero espera. ¿Qué pasaría en caso de que yo fuera el atacante?”

Bueno, en ese caso las cosas **se te van a complicar**. Primero porque sólo conocés n. Para sacar φ(n) debemos tener p y q, pero para sacar p y q debemos empezar a dividir a n por números primos enormes. Esto consume **muchos recursos**, muchos en serio (se supone que con la computación cuántica podríamos olvidarnos de usar RSA xD).

Ya que tenemos las claves podemos **cifrar**:

Tengo un mensaje M y tengo la clave pública de Alicia. Para esto primero convierto a **M en un número m** (sí, un número) **menor que n** (el valor del módulo) y se hace **m elevado a la e**, en módulo n para hacer el cifrado. Es decir:

$$m^e = c \pmod{n}$$

siendo c el valor cifrado, m el mensaje llevado a un valor numérico y e que es el número random que habíamos elegido anteriormente.

Ahora nos queda **descifrar**:

Acá hay una **propiedad** que nos ayuda que es un poco loca. Dice que **teniendo un número *a*** (distinto de cero), **si lo multiplicamos por si mismo φ (n) veces, el resultado será 1** (en módulo n). Esta fórmula puede resumirse de esta manera:

$$1 \pmod n = a^{\varphi(n)}$$

En el ejemplo anterior, donde tenemos n = 7, φ(7) = 6. Podemos decir que a = 2 y que lo tenemos que elevar a la 6 (por φ(7)). Entonces:

$$2^6 = 64 = 1 \pmod 7$$

Vamos a intentar cifrar y descifrar un mensaje para ver si nos queda el mismo valor.

$$c = m^e$$

donde c es el cifrado

y

$$c^d = m$$

Entonces:

$$m = (m^e)^d = m^{ed} \pmod n$$

Recuerden que **e y d salen de φ y son inversas.** ¡Sí, inversas! Eso quiere decir, en módulo n, que **el resultado de su multiplicación es 1**. Por lo tanto:

$$m^{ed} = m$$

Ejemplo numérico:

1. Elegimos los números primos que para el ejemplo serán pequeños. **Q = 7 y p = 11.**
2. Ahora debemos calcular φ (n)

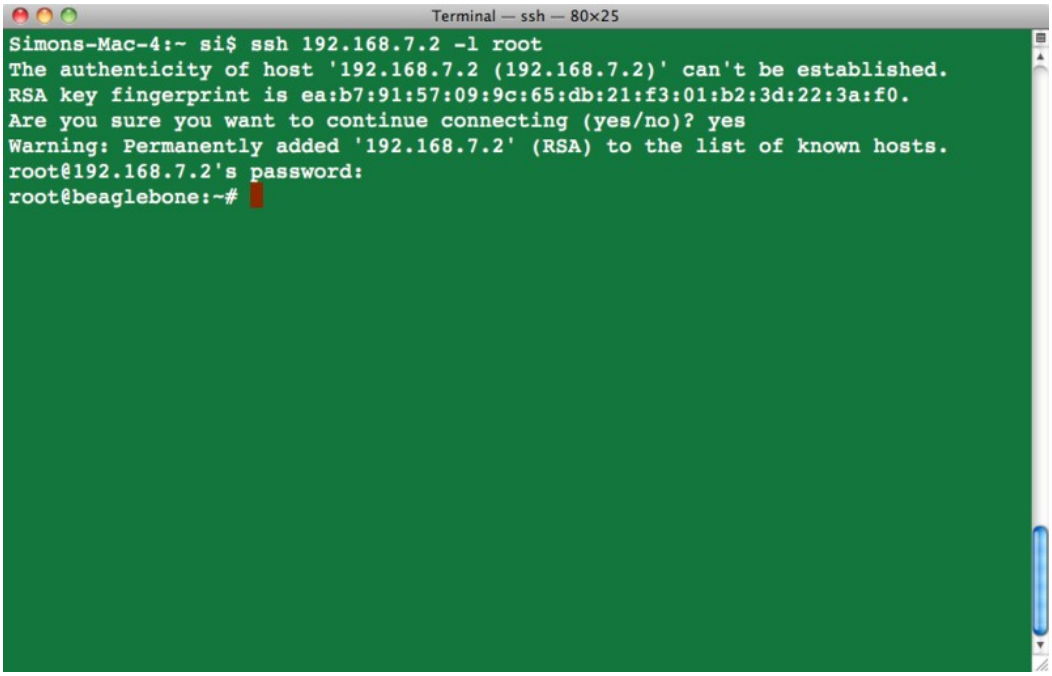
   φ (n) = (p -1) * (q – 1)
   φ (n) = 6 * 10 = 60

3. Elegimos un **primo relativo** con φ (n). **Elijo e = 13**.
4. Calculamos **la inversa de 13 que es 37**. Si quieren hacerlo ustedes busquen el algoritmo de euclides. Por lo tanto **d = 37** y la **clave pública** se compone por **(60, 13) y la clave privada es (60, 37)**.
5. Si queremos **cifrar**, tomemos un número: **m = 3; entonces 3^{13} = 1594323 = 3 (mod 60).** Este número es el que irá del cliente al servidor.
6. **Para descifrar elevamos 3^{37} = 3 (mod 60)**.

“Creo que la criptografía no es lo mío.”

Bueno, la criptografía suele estar muy pegada al ámbito matemático. No te sientas mal. Sigamos con SSH. ¿Dónde nos habíamos quedado?

4. Ah, sí. **Luego de generar las claves RSA, le pasa la pública al cliente**. El cliente compara esta clave pública con la que tiene almacenada para identificar si es real la **autenticidad**. Si no la conoce anteriormente, puede pedir verificación del usuario.
5. **El cliente ahora genera una clave para la sesión aleatoria** que se usará con un algoritmo de cifrado **simétrico** que será seleccionado.
6. **El cliente envía un mensaje al server con la clave de la sesión y el algoritmo a usar.** Obviamente primero se cifra con la clave pública RSA compartida anteriormente por el server.
7. **Desde este momento,** ambas partes se comunican con el **cifrado simétrico**.
8. **El usuario se autentica** por el método tradicional de user-pass o también puede haber otro tipo, como el uso de claves asimétricas de parte del server (con esto pueden crear una whitelist dentro del servidor).
9. Luego de todo este embrollo, **la sesión inicia**. Generalmente es interactiva (es decir que el servidor devuelve información sobre lo que el usuario está administrando.



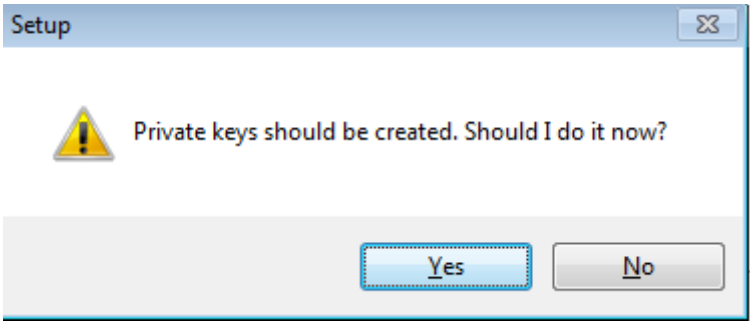
Esta sesión puede tener privilegios bajos o altos dependiendo de la configuración de ese usuario o del sistema.

“Ok. Creo que voy a volverme músico y ya.”

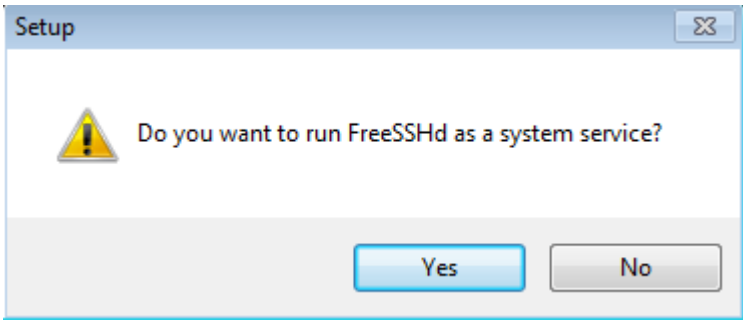
¡Que no decaiga! Además lo decís como si la música fuese algo muy fácil… No hagas que me pierda. Volvamos. Para aprovechar este sistema de seguridad también le dieron la aplicación de transferencia de archivos llamado **SFTP**.

Aunque se llama de una manera muy parecida al FTP, es un **protocolo** construido **desde cero** donde podemos **transmitir archivos**. Creo que no voy a darle hincapié a esto. **Vamos a instalar un servidor SSH en Windows**. Para esto debemos descargar el **freeSSHd** desde www.freesshd.com . Igual yo elegí este server pero pueden usar el que gusten.

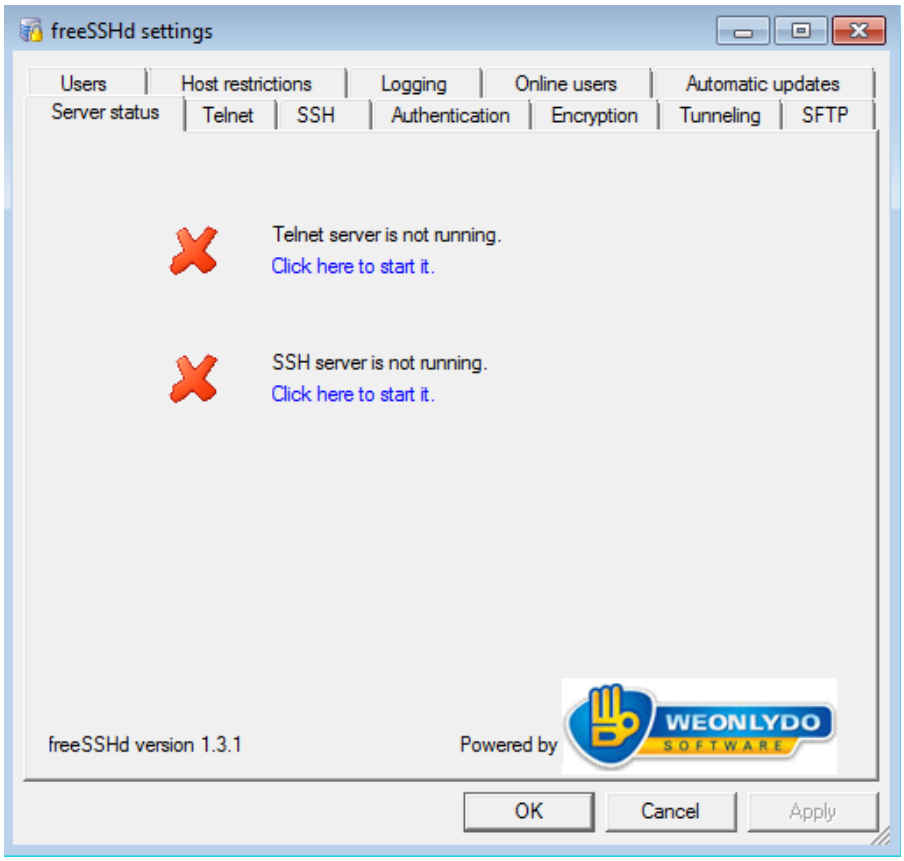
Para instalarlo, como todo programa “next” hasta encontrarnos con este cartel:



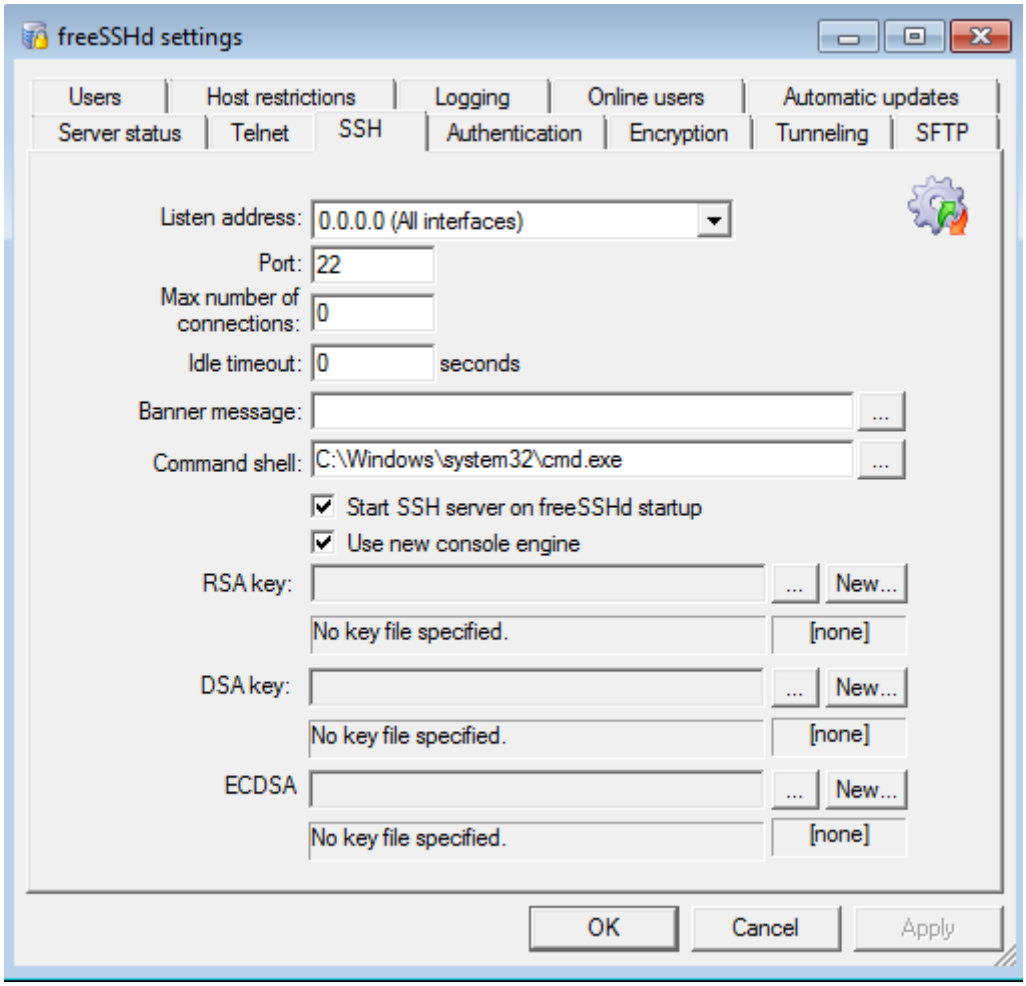
Y ponemos que **no**, al igual que en el próximo alert.



Bueno, una vez instalado podemos abrir el programa.



En esta pestaña podemos verificar si el servicio está corriendo, **iniciarlo o pararlo.** Claro que aún nos falta configurar aquello. Pasemos a la **pestaña SSH**.



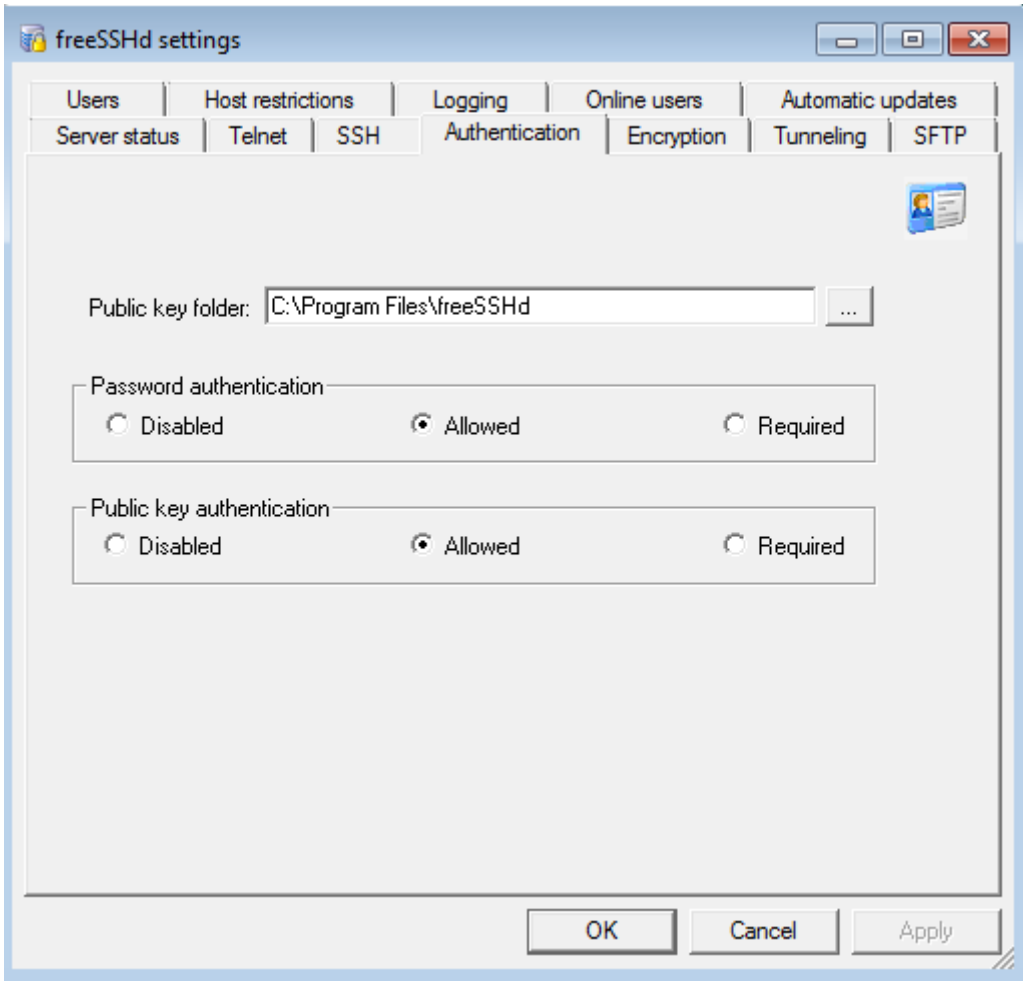
De arriba a abajo, tenemos:

- **Listen address:** aquí podemos elegir en que red deja que el servicio esté a la espera de conexiones. Yo elegiré todas las interfaces.
- **Port**: el puerto por el cual está a la escucha. Por defecto es el 22 y generalmente lo encontraremos ahí. Pueden cambiarlo si lo así lo quieren.
- **Max number of connections:** parece que estoy explicando lo obvio, pero son la cantidad de clientes que pueden conectarse al mismo tiempo. Yo lo pongo en 1.
- **Idle timeout:** Una cantidad de tiempo que tiene que pasar en inactividad para cerrar la conexión. El cero significa que no tiene límite.
- **Banner message**: es el mensaje que aparece cuando uno logra la conexión exitosa. Pondré un pequeño mensaje.
- **Command shell:** aquí que tipo de shell abrirá cuando se conecte al server.

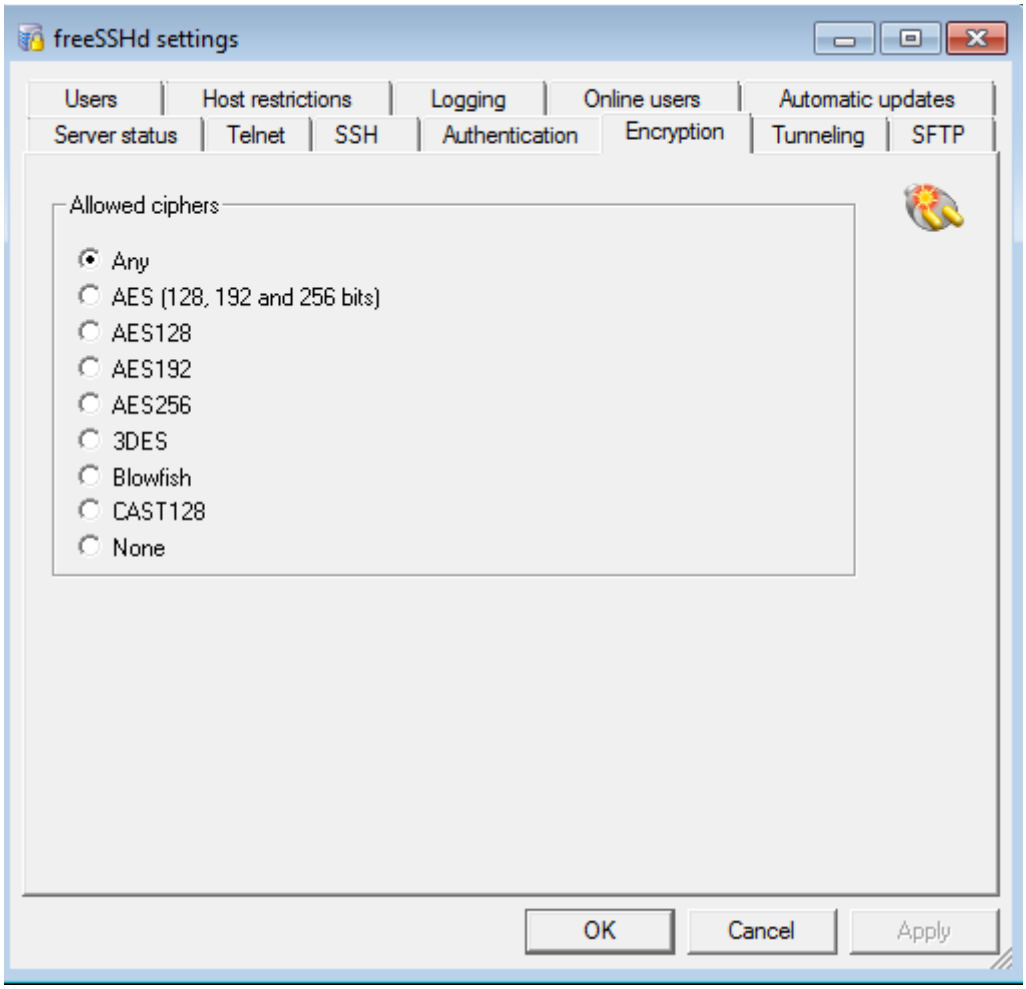
Podría ser una shell modificada, pequeña, con menos pivilegios, etc. Dejemos la que está.

- Luego nos aparecen **3 llaves** que son **RSA key** (recuerden que la usamos para iniciar la conexión y puede identificar al servidor), **DSA** que es únicamente para la firma de archivos, y **ECDSA** que es una mejora del anterior logaritmo. Creémos una nueva key RSA y la guardamos en cualquier lugar de nuestro sistema. Recuerden que esta key es nuestra clave privada y debe estar bien segura.

Pasemos a la pestaña de **Authentication**.

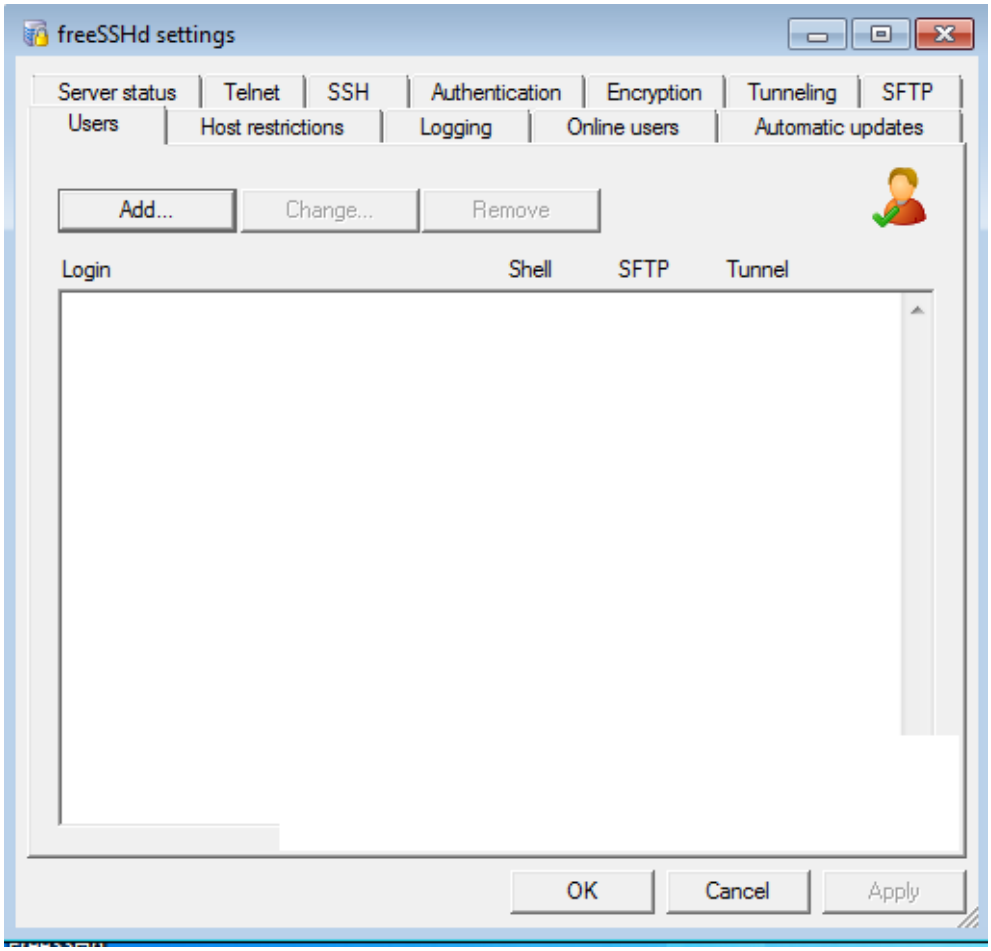


Aquí vemos ambas maneras de autenticarse que describí anteriormente en los pasos del SSH. Si tenemos una clave pública de algún cliente que sabemos que es conocido, podemos ponerlo en el path que nos aparece arriba. Sigamos que esto lo podemos dejar como está.

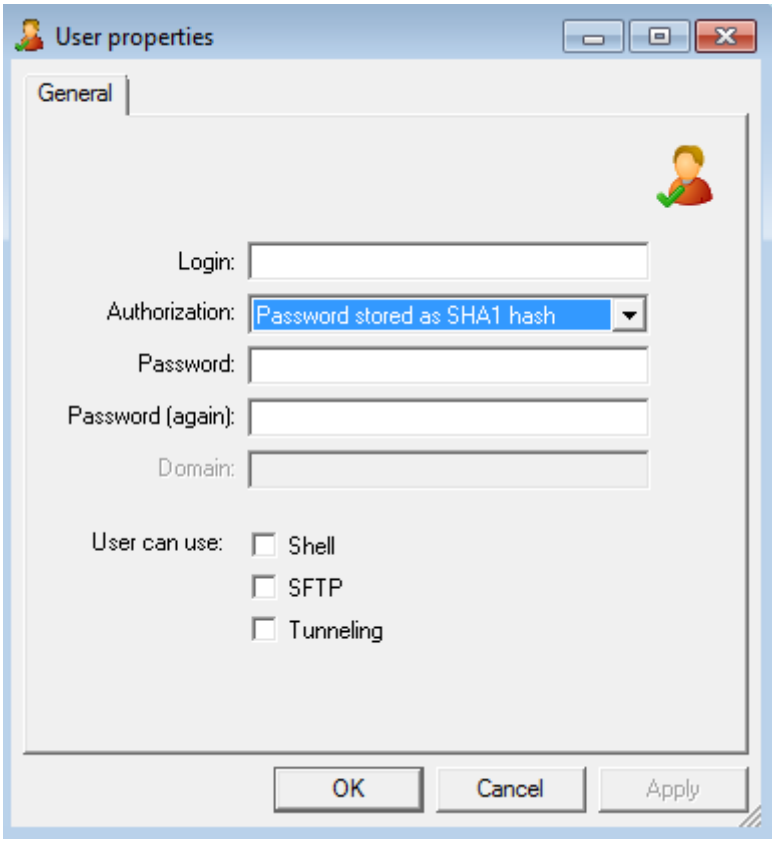


En **Encryption** podemos elegir nuestro tipo de cifrado simétrico para la conexión. **Tunneling** lo veremos más adelante en esta misma clase.

Users:

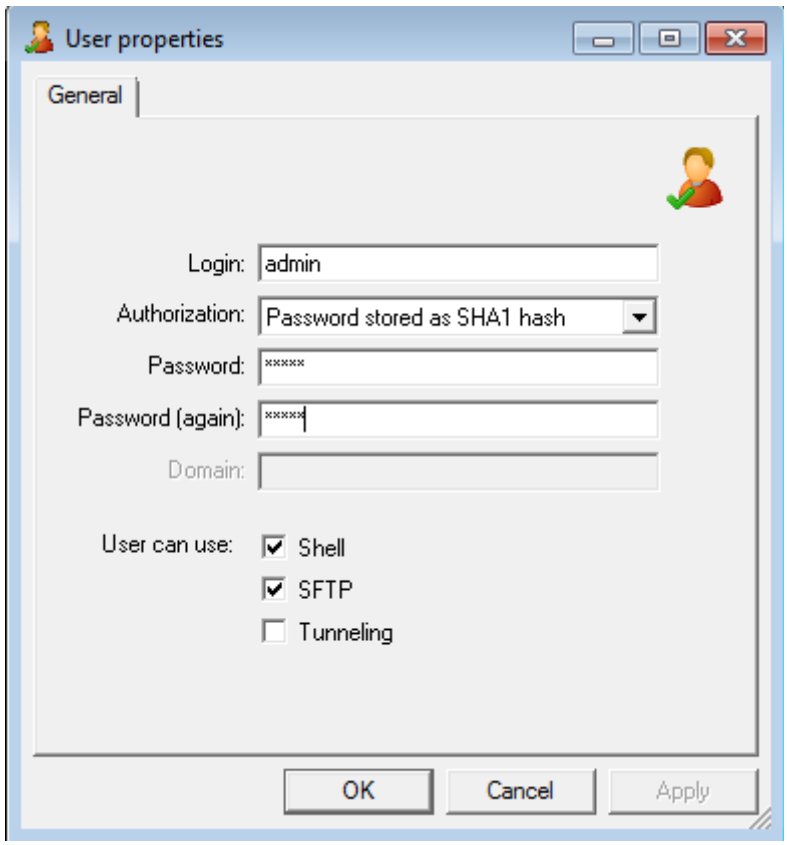


Aquí debemos **añadir un user** para que quien intente realizar una conexión y tenga éxito pueda loguearse.

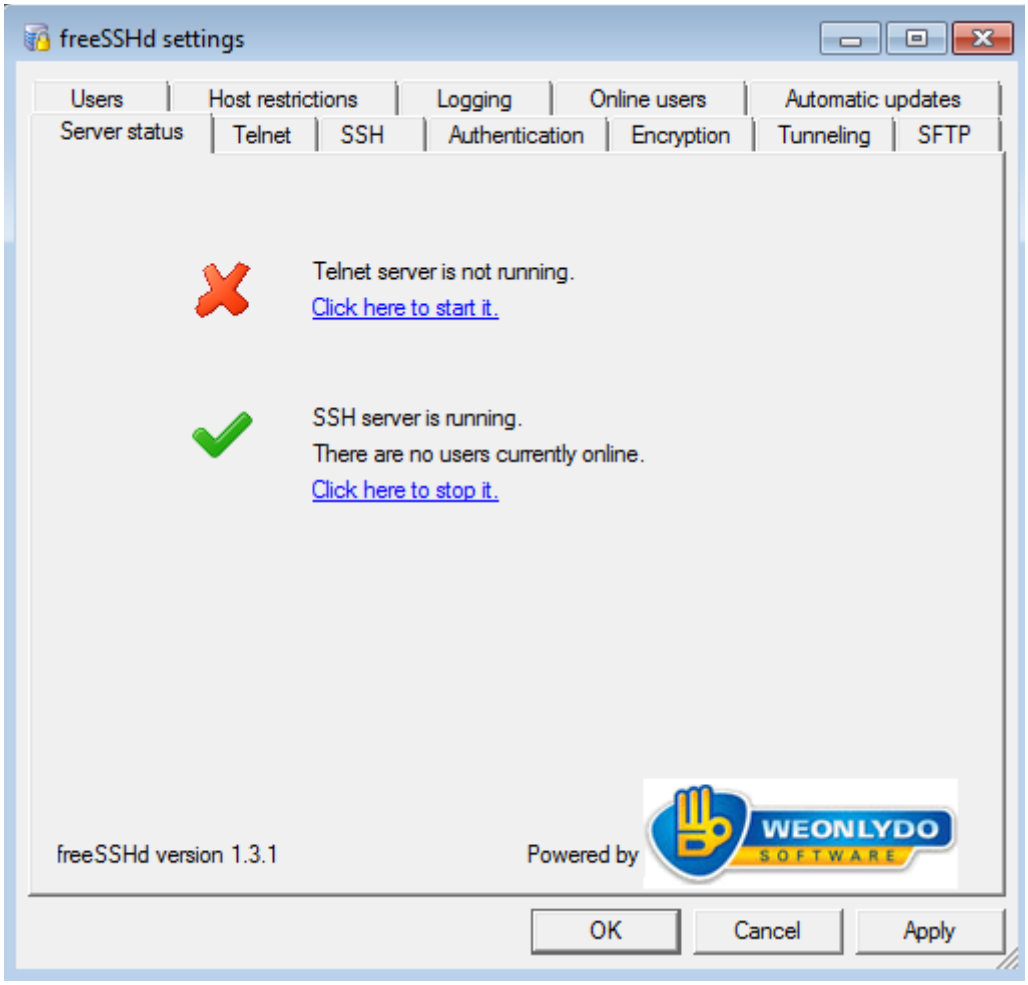


Lo primero que elegí fue el **tipo de autorización**, que en mi caso es la de un usuario y un password creados desde cero. Las otras opciones corresponden al uso de un user y pass de los usuarios de sesión del server y al uso de la clave pública contra el cliente. Voy a poner admin tanto de usuario como de contraseña.

Por último vemos los 3 privilegios (shell, sftp, tunneling) que nos concede. **Habilitaré las primeras 2.**

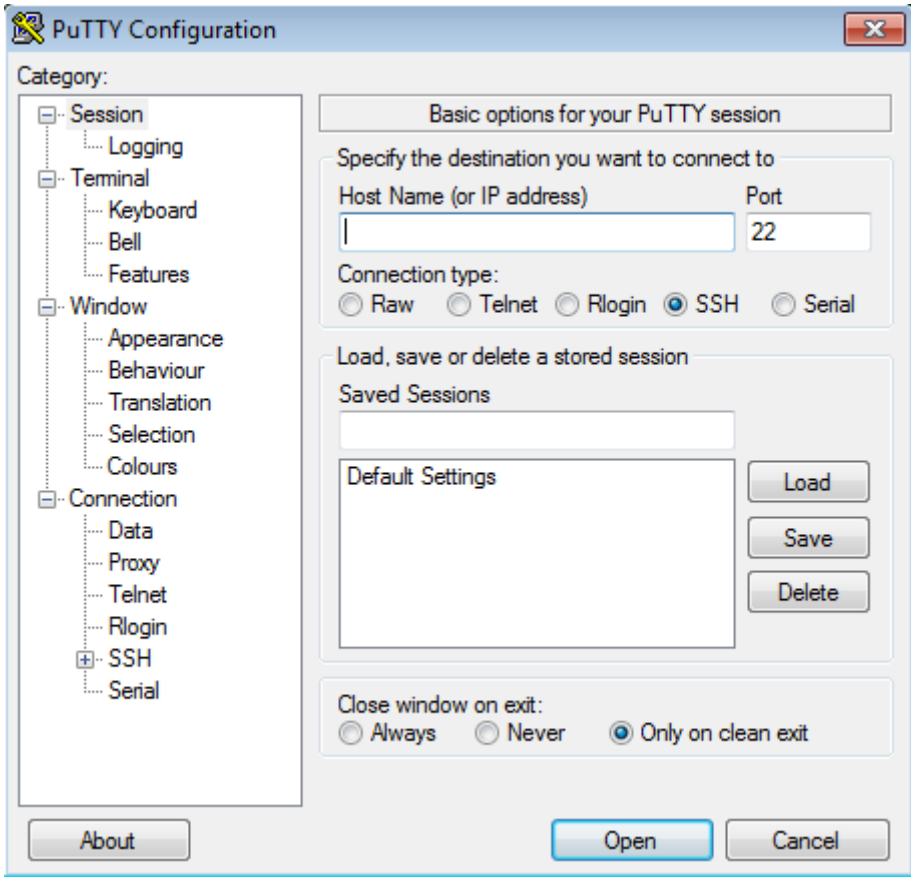


Las demás pestañas son bastante intuitivas para su uso y no hace falta que las nombre. **Le damos a Apply**, reiniciamos el programa del server (es decir cerramos y volvemos a abrir) y vamos de nuevo a la pestaña de Server status y **a correr el servicio**:



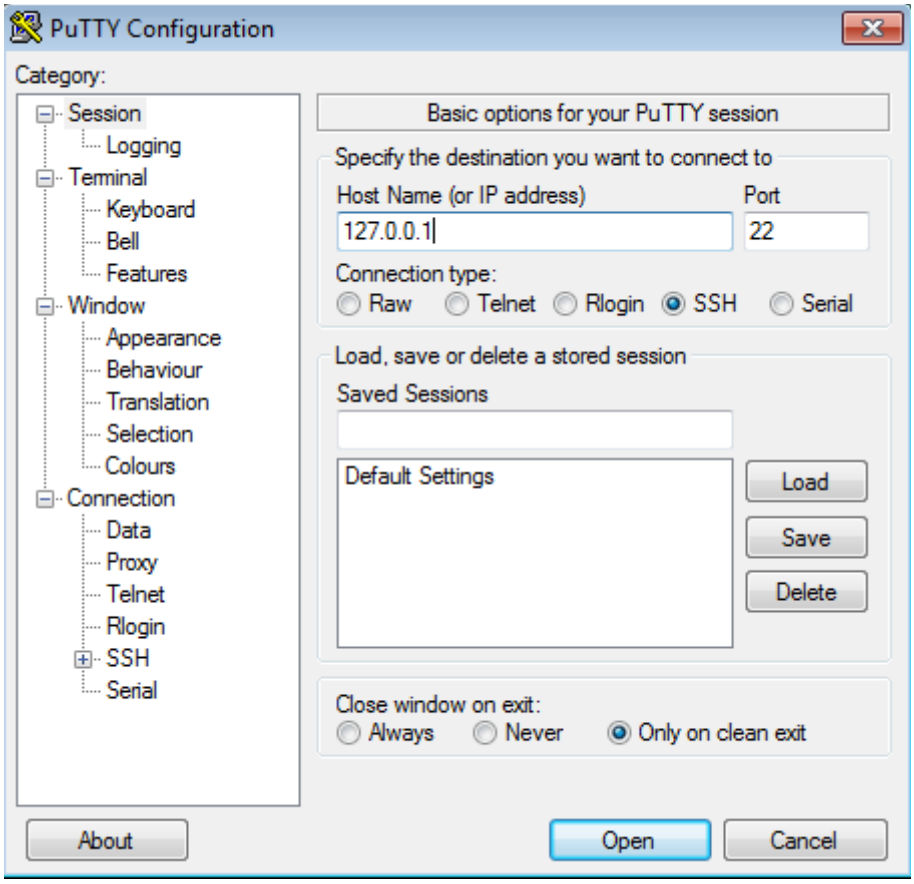
Buenísimo. Está en funcionamiento. **Ahora necesitamos un cliente** que pueda conectarse a este servicio. En realidad, yo podría hacerlo desde cualquier pc de mi red, pero por comodidad para las capturas de pantalla voy a hacerlo en la misma pc que hace de server. Hay un cliente que se usa mucho y que es muy conocido llamado **Putty**. Lo descargamos desde su página oficial en www.putty.org y no lleva instalación.

Una vez abierto vemos sus opciones básicas.

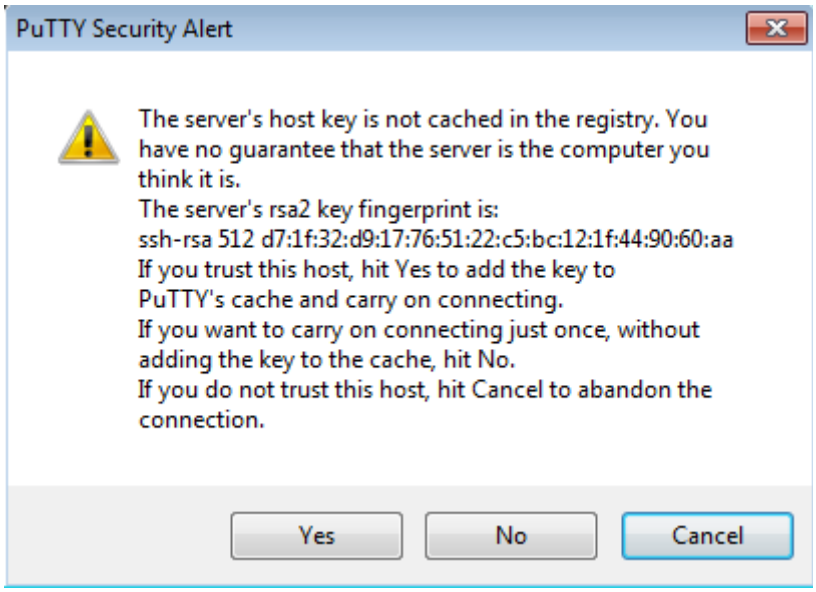


Los campos son:

- **Host name**: aquí nuestra IP. En mi caso es el localhost por lo que puedo resumirlo a 127.0.0.1
- **Port**: por defecto era el 22, que también es el que dejé.
- **Connection type**: aquí nos aparecen varias, vamos a elegir SSH.

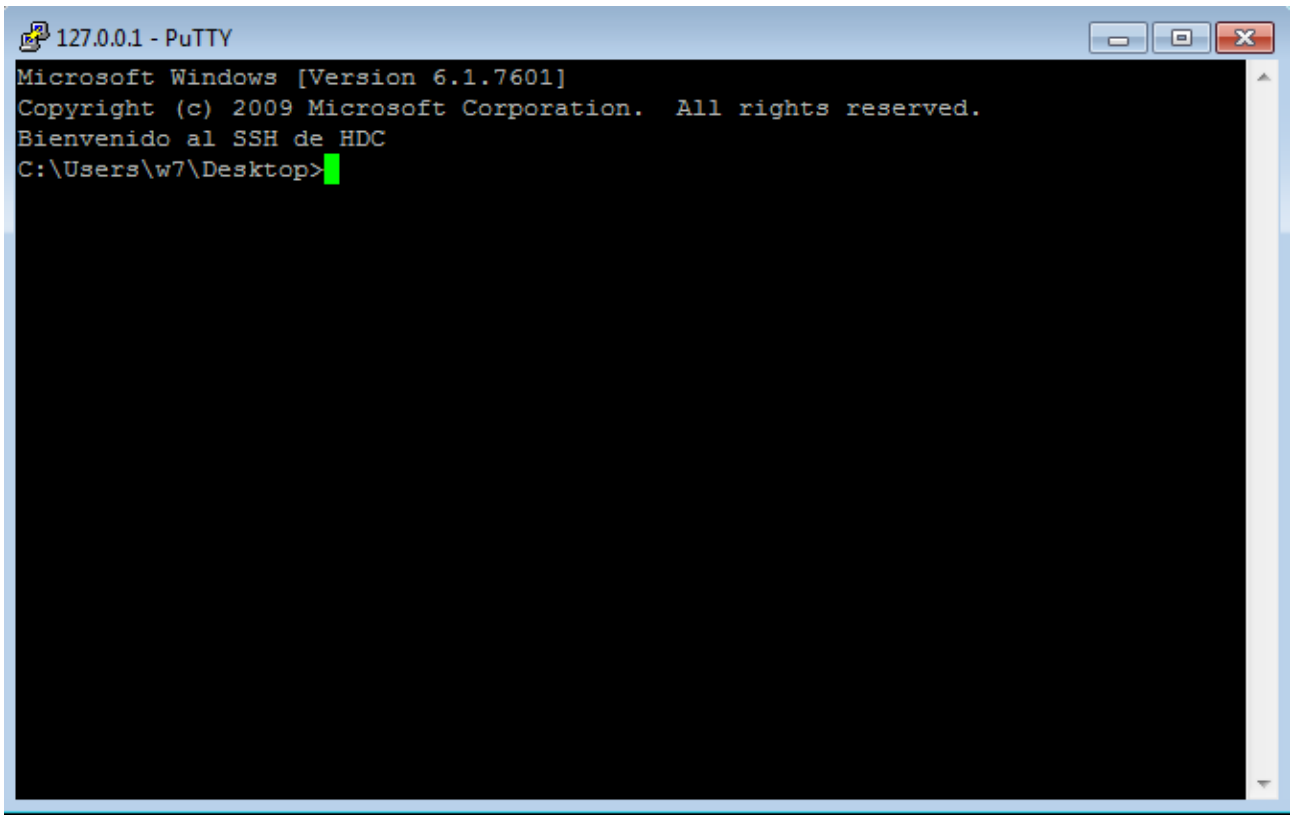


Una vez terminado le damos a **Open** y nos aparecerá este cartel.



En simples palabras dice que la firma del servidor, el id, no está en nuestra base de datos. **¿Debemos confiar en él?** Bueno, en este caso no tenemos ninguna duda y le damos al sí.

Pongo mi usuario y contraseña y...



Tenemos el **banner** que había escrito y **la shell** de control. Muy lindo, muy lindo. Este servicio es muy usado para manejar servidores. Sobre todo en donde se corre alguna distribución de Linux.

“Esperá. ¿Y el tunneling que era?”

Ah, cierto. Lo olvidé. **El tunneling es una manera de usar el medio de comunicación seguro y cifrado de SSH para una conexión de otro servicio.** Nosotros ahora no vamos a aprovecharlo pero es una gran manera de lograr bypassear proxys con black lists (lo veremos en alguna otra clase también).

“Ya vimos un montón. Pero me queda una duda. ¿Podría hacer un ataque de fuerza bruta al servicio?”

Es una excelente pregunta, Manolo. Y **sí, se puede**. Lo que es cierto es que no conozco muchos programas de fuerza bruta para Windows. Utilicé el **ncrack**, descargado desde: https://nmap.org/ncrack/dist/ncrack-0.4ALPHA-setup.exe que siendo un alpha es bastante completo.

Luego de la instalación simple, la ruta se agrega al path así que podemos ejecutar el programa desde cualquier ruta en la terminal. De igual manera el ataque no tuvo ningún éxito y no sé por qué (lo siento, esta vez no habrá explicación :D porque yo también la necesito). Pero ustedes prueben el ataque en su sistema. El comando que usé fue:

```
ncrack -p 22 -v --user admin --pass admin localhost
```

Los parámetros son fáciles de entender. Pueden listar todos los disponibles usando ncrack sin parámetros

Aunque también lo intenté desde otros sistemas y otros programas con el mismo resultado. Una lástima no poder probarles un éxito, pero ya tienen con qué entretenerse. Si llegan a lograrlo me avisan así edito el texto:).

Creo que por hoy estamos bastante completos (uf, que fue larga). Hasta la próxima:)

Pueden seguirme en Twitter: @RoaddHDC
Cualquier cosa pueden mandarme mail a: r0add@hotmail.com
Para donaciones, pueden hacerlo en bitcoin en la dirección siguiente: 1HqpPJbbWJ9H2hAZTmpXnVuoLKkP7RFSvw

Roadd.
