

SERVIDOR OPENSSH EN LINUX: CONFIGURACIÓN PARA MÁXIMA SEGURIDAD

SSH o también conocido como **Secure Shell**, es un protocolo y el nombre del programa que lo implementa. SSH es ampliamente conocido por ser el protocolo seguro para la administración remota de servidores, routers, switches y un largo etcétera de equipos. El protocolo SSH permite manejar por completo el servidor o dispositivo de red mediante un intérprete de órdenes, además, también podemos redirigir el tráfico de X para ejecutar programas gráficos a través de la propia sesión SSH.

Otras características fundamentales de SSH son que nos va a permitir **copiar datos de manera segura, tanto archivos como carpetas, a través del protocolo SFTP** (SSH FTP), un protocolo hecho desde cero y que no tiene nada que ver con FTPS o FTPES (FTP sobre SSL/TLS). El protocolo SSH es fundamental en el ámbito de las redes y sistemas, además, podremos configurarlo en detalle para dotar a nuestro sistema de la máxima seguridad posible.

El protocolo SSH proporciona **confidencialidad** (los datos van cifrados punto a punto), **autenticación** (podremos autenticarnos frente al servidor SSH de múltiples maneras, con usuario/clave, criptografía de clave pública e incluso podremos configurar un segundo factor de autenticación), **integridad** (si los datos se modifican o los modifica un usuario malintencionado se podrá detectar, ya que usa HMAC para comprobar la integridad de todos y cada uno de los datos).

Existen dos versiones de SSH, la versión 1 no se recomienda hoy en día usarla, de hecho, por defecto siempre se utiliza ya la versión SSHv2. Por defecto SSH utiliza el protocolo TCP de la capa de transporte, y el número de puerto es el 22, no obstante, podremos cambiar el número de puerto para mitigar posibles escaneos de bots al servicio SSH.

En este manual que os presentamos, os vamos a enseñar a configurar el servidor SSH de OpenSSH, un programa que está disponible para sistemas operativos basados en Unix y Linux, como por ejemplo FreeBSD, OpenBSD, Debian, Red Hat Enterprise Linux y un largo etcétera de distribuciones. En esta guía no solo aprenderemos a configurar correctamente el archivo de configuración `sshd_config`, sino que usaremos programas adicionales para dotar al sistema de la máxima seguridad posible. En este manual todas las configuraciones las realizaremos con Debian, no obstante, todas las medidas de seguridad pueden ser implementadas en cualquier sistema operativo basado en Linux y Unix.

VENTAJAS DEL PROTOCOLO SSH

Una de las principales ventajas con las que cuenta este protocolo, es que puede ser implementado en diferentes sistemas operativos de forma estándar, lo cual hace que la creación de canales seguros entre diferentes equipos, tanto locales como remotos, sea más sencilla.

Este también nos da la posibilidad de administrar routers, servidores, plataformas de virtualización y otros Sistemas Operativos, entre otros. Pero estas no son sus únicas ventajas, si no que nos podemos encontrar beneficios en diferentes apartados.

- **Conexión:** Este protocolo se utiliza para conectarse a servidores, y realizar cambios sobre los mismos desde un terminal. Sus claves SSH pueden ayudar a automatizar estos accesos, que en muchas ocasiones utilizan scripts, diferentes sistemas de backup y otras herramientas de gestión para la configuración.
- **Criptografía:** Todo el tráfico generado al utilizar este protocolo está encriptado. Desde la transferencia de un archivo, navegar por internet o ejecutar algún tipo de comando, todo se encuentra protegido. Puesto que su diseño es para poder realizar gestiones y trabajar más allá del propio lugar donde se encuentra el usuario, las claves SSH hacen que el inicio de sesión sea único, por lo cual los usuarios pueden cambiar entre diferentes cuentas sin necesidad de introducir contraseñas para cada uno
- **Flexibilidad:** Este es muy utilizado para ejecutar scripts y otros software que permiten a los programas y sistemas acceder a los datos que establezcamos y a los recursos de forma remota, sin perder seguridad.

¿CÓMO FUNCIONA EL PROTOCOLO SSH?

Lo primero que se debe establecer, es una coincidencia entre los números primos por parte del cliente y del servidor. Pero estos no son necesario que tengan factores en común. A este valor se le conoce como «Valor Semilla» (Seed Value). Luego de verificar esa coincidencia, ambas partes proceden a acordar un mecanismo para cifrar los datos. Y de esta forma generar otro conjunto de valores, mediante la manipulación de los valores semilla de antes. Esto se realiza de una forma algorítmica específica, y no aleatoriamente. A esto se le conoce como generador de cifrado, y son capaces de realizar grandes operaciones sobre el valor semilla. El ejemplo más claro actualmente es AES.

Luego tanto cliente como servidor, generan otros números primos de forma independiente, el cual se utiliza como clave privada y secreta para realizar la comunicación. Esta clave privada que se acaba de generar, junto con el número compartido y los algoritmos de cifrado como AES, se utilizan para calcular la clave pública. Las cual se procede a distribuir a al otro equipo. Una vez tienen la clave privada, la clave pública del otro equipo y el número primo original, los utilizan para crear una clave final que se comparte. Esta se calcula también de forma independiente en ambos equipos, pero se crea la misma clave de cifrado en las dos partes.

Una vez cliente y servidor cuentan con la clave compartida, se puede cifrar de forma simétrica la sesión SSH. Toda ella en este caso. Y con esto, utilizan la misma clave para realizar el cifrado y el descifrado de los mensajes. Pero antes de todo ello, el usuario deberá proceder a autenticarse en el sistema para poder llevar a cabo sus actividades en él. En el caso de no tener acceso, no podrá realizar ningún tipo de comunicación.

TIPOS DE CIFRADO

Para realizar este cifrado, **OpenSSH** usa tres tipos de cifrado, lo que es una ventaja ofrecida por este protocolo para asegurar las transferencias de información entre el host

y el cliente. En este caso, el host es el servidor remoto al que intentamos acceder, y el cliente es el equipo que estamos utilizando para acceder al host.

Estas tecnologías de cifrado son:

CIFRADO SIMÉTRICO

Para este tipo de cifrado se utilizará una clave secreta, esta será tanto como para cifrar como para descifrar un mensaje, sin diferenciar entre el host y el cliente. Pero, cualquier que conozca la clave, puede descifrar los mensajes. También se le puede llamar como clave compartida o cifrado secreto compartido.

Lo que hace que este algoritmo sea seguro, es que la clave no se transmite entre el cliente y el host. En su defecto, ambos comparten datos públicos y luego los manipulan para realizar el cálculo de la clave secreta de forma independiente. En el caso de que otro equipo capture los datos compartidos, no podrá calcular dicha clave, pues el algoritmo no es conocido.

Este nos proporciona algunas ventajas, como por ejemplo la facilidad de configurar, es muy directo y se puede usar por todo el mundo y por usuarios de todas las edades y niveles de conocimiento. Pero como todo, tiene su desventaja. Y en este caso es que es necesario compartir la clave secreta con los destinatarios. Si ponemos un PEM de ejemplo, la clave secreta es cifrada con la contraseña del usuario, con el fin de garantizar que luego sea más sencillo de adivinar. Por lo cual si usamos la misma clave, y alguien la descubre con algún método malicioso, podrá ver todos nuestros datos sin ningún tipo de restricción.

CIFRADO ASIMÉTRICO

Este tipo usa dos claves separadas para cifrar y descifrar. Una es la clave pública y otra la privada, y juntas forman un **par de claves pública-privada**. En cuanto a la pública, esta se transfiere y comparte con todas las partes de la comunicación. Mientras que la privada permanece en privado, por lo cual para que sea seguro, ningún otro equipo debe conocerla.

Este cifrado utiliza tres tipos de cifrado para generar las claves públicas, estos son **RSA**, **DSA** y las **curvas elípticas**. La integridad de cada uno de estos, se establece de forma matemática, lo cual constituye el núcleo del propio algoritmo.

El cifrado asimétrico no se usa para cifrar la conexión SSH por completo, en su lugar solo se usa durante el intercambio de claves. Esta, antes de iniciar la conexión, se generan el par de claves temporales y se comparten las privadas para poder generar la clave secreta, y poder realizar la conexión.

Su gran ventaja es que no obliga a los usuarios a compartir claves, a diferencia del simétrico, por lo cual se elimina la necesidad de distribuirlos. Con este cifrado, al disponer de una clave privada, podremos obtener una pública, pero este proceso no puede ser realizado al revés. También es compatible con firmar digitales, las cuales identifican identidades, en este caso del destinatario y garantizan que no se realizan modificaciones del mensaje durante el camino del mismo.

Sus desventajas, son principalmente, que no es tan simple de configurar como el simétrico, e implica que los destinatarios necesitan experiencia, conocimiento o uso del mismo método.

HASHING

Hashing es otra forma de cifrado que se usa en **Secure Shell Connections**. Sus funciones son unidireccionales y se diferencian de las otras dos formas de encriptación, pues estas no se destinan a ser descifradas en ningún momento. En cada entrada generan un único valor de longitud determinada, la cual no muestra tendencia a ser explotada, por lo cual es prácticamente imposible revertirla.

Generar un hash es sencillo, pero es imposible generar un entrada para un hash, por lo cual si un cliente tiene una entrada correcta se pueden generar hashes y proceder a comparar su valor para que se verifique si poseen la entrada adecuada y correcta.

Esto se puede usar en firmas digitales, por ejemplo. Estas se usan en todo internet, de modo que cada vez que procedemos a acceder a una página web, se utiliza un entramado de firmas digitales, de forma que se establece la conexión de confianza entre usuario y servidor.

Las firmas digitales, incluyen mensajes de forma que el destinatario pueda verificar la identidad a través de los mismos. Y el receptor verifica el mensaje. En este caso, hashing nos facilita métodos para que todos los usuarios de una cadena de bloques, estén de acuerdo con algún estado actual de dicha cadena. Luego las firmas digitales proporcionan una metodología de seguridad la cual verifica que las transacciones se hacen únicamente por los propietarios de la cadena. Entre las dos propiedades, tratan de evitar posibles corrupciones. Uno de los ejemplos más actuales y conocidos que nos podemos encontrar es el blockchain.

INSTALACIÓN DE OPENSSH Y PUESTA EN MARCHA

OpenSSH es el programa servidor/cliente SSH más utilizado por los routers, switches, servidores y un largo etcétera de dispositivos. Este programa es completamente gratuito y de código abierto. La **instalación de este servidor SSH** (si es que no lo tienes ya instalado por defecto) es muy sencilla, simplemente debemos poner en un terminal la siguiente orden:

```
sudo apt install openssh-server
```

Una vez instalado, debemos tener en cuenta ciertos directorios y órdenes para iniciar, parar y reiniciar el servicio SSH.

Para editar la **configuración del servidor SSH** debemos hacer en consola:

```
sudo nano /etc/ssh/sshd_config
```

Otro directorio que tenemos que tener muy en cuenta es la de host conocidos, ya que aquí también es donde configuraremos las claves criptográficas RSA/DSA. El **directorio**

donde se encuentran los hosts conocidos y las claves públicas es el siguiente:

```
/home/usuario/.ssh/
```

Este directorio por defecto está oculto (.ssh) y hay un directorio por cada usuario que haya en el sistema operativo y que se conecte a un servidor remoto.

Para arrancar el servidor:

```
sudo /etc/init.d/ssh start
```

Para parar el servidor:

```
sudo /etc/init.d/ssh stop
```

Para reiniciar el servidor:

```
sudo /etc/init.d/ssh restart
```

Una vez que hemos instalado el servidor SSH, sabemos dónde están los archivos de configuración del servidor y el directorio donde se almacenan las claves públicas, vamos con la configuración del sshd_config ya que es el archivo de configuración fundamental de OpenSSH.

CONFIGURACIÓN DE SSHD_CONFIG PARA LA MÁXIMA SEGURIDAD

Es muy importante configurar adecuadamente el archivo de configuración sshd_config para mitigar al máximo cualquier posible ataque que nos hagan desde la red local o desde el exterior. Proteger el servidor SSH debe ser primordial, porque si se hacen con el control del servidor podríamos comprometer toda la red local doméstica y profesional.

CAMBIAR EL PUERTO POR DEFECTO DEL SERVIDOR SSH

Por defecto los servidores SSH utilizan el puerto 22 para las conexiones. Es recomendable cambiar este número de puerto, para evitar que bots o cibercriminales puedan intentar iniciar sesión, aunque por sí solo esto no proporciona seguridad, sí podremos pasar desapercibidos a los escaneos masivos desde Internet. Si por ejemplo queremos usar el puerto 22445 debemos poner en el fichero de configuración lo siguiente:

- **Port 22445**

BLOQUEAR EL ACCESO ROOT EN LAS CONEXIONES REMOTAS

Por defecto, cualquier usuario en el sistema operativo que tenga permisos de Shell, podrá iniciar sesión en el servidor. Además, debemos tener en cuenta que si tenemos activado el usuario root, también podrá conectarse al servidor de forma local o remota, evitando al atacante tener que «adivinar» el nombre de usuario. Por defecto, los bots siempre intentan atacar el puerto 22 y al usuario «root».

Desactivando al propio usuario root, y usando «sudo» para elevar a permisos de superusuario, evitaremos esto. Además, OpenSSH también nos permitirá deshabilitar el login del usuario root para dotar al sistema de mayor seguridad:

- **PermitRootLogin no**

De esta manera las conexiones root quedarán bloqueadas evitando que usuarios no autorizados puedan realizar ataques de fuerza bruta contra nuestro servidor SSH para adivinar los credenciales del usuario Root. También tenemos otras opciones en este apartado, como por ejemplo «**PermitRootLogin without-password**» donde se permite autenticación pero no con usuario y contraseña, sino con claves criptográficas RSA.

Existen otras configuraciones recomendadas para evitar las conexiones no deseadas a nuestro servidor SSH. Estas conexiones son:

- **LoginGraceTime**: Estableceremos el tiempo necesario para introducir la contraseña, evitando que el atacante tenga que «pensar mucho».
- **MaxAuthTries**: Número de intentos permitidos al introducir la contraseña antes de desconectarnos.
- **MaxStartups**: Número de logins simultáneos desde una IP, para evitar que se pueda utilizar la fuerza bruta con varias sesiones a la vez.
- **AllowUsers**: Es crear una lista blanca de usuario. Este parámetro nos permite configurar los usuarios que podrán conectarse. Una medida muy restrictiva pero a la vez muy segura ya que bloqueará todas las conexiones de los usuarios que no estén en el listado. Los usuarios que tengamos aquí podrán conectarse, y el resto no.
- **DenyUsers**: Parecido al anterior, pero ahora creamos una lista negra. Los usuarios que tengamos aquí no podrán conectarse, y el resto sí.
- **AllowGroups/DenyUsers**: Exactamente igual a lo anterior, pero en lugar de crear una lista blanca/negra de usuarios, es de grupos de usuarios.

Por ejemplo, un archivo de configuración de sshd_config sería el siguiente:

| | | |
|-----------------|----------------|---------|
| Port | | 22445 |
| PermitRootLogin | | no |
| LoginGraceTime | | 30 |
| MaxAuthTries | | 3 |
| MaxStartups | | 3 |
| AllowUsers | sergio | sergio2 |
| DenyUsers | adrian adrian2 | |

Una medida de seguridad adicional es configurar los algoritmos de intercambio de claves, el cifrado simétrico y también, la configuración del HMAC para la comprobación de la integridad. Actualmente es recomendable aplicar la siguiente configuración para tener una seguridad muy alta:

KexAlgorithms curve25519-sha256@libssh.org,ecdh-sha2-nistp521,ecdh-sha2-nistp384,ecdh-sha2-nistp256,diffie-hellman-group-exchange-sha256 Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctr MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-

etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-512,hmac-sha2-256,umac-128@openssh.com

Con esta configuración tendremos las mejores suites criptográficas para el servidor, sin embargo, es posible que clientes antiguos no puedan conectarse al no soportar estos algoritmos. Debemos tener este detalle muy en cuenta, y probar qué algoritmos son compatibles y cuáles no.

Si hemos creado nuevas claves de RSA o DSA por unas con mayor longitud de bits, deberemos ponerlo en el fichero de configuración (o sustituir las anteriores, y así no tendremos que tocar el fichero de configuración), de esta forma obtendremos una seguridad adicional si por ejemplo usamos claves RSA de 4096 bits o superior.

```
HostKey /etc/ssh/ssh_host_ed25519_key
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
```

Para generar unas claves RSA de 4096 bits nuevas, simplemente deberemos ejecutar el siguiente comando:

```
ssh-keygen -f /etc/ssh/ssh_host_rsa_key -t rsa -b 4096
```

Si queremos generar nuevas claves ECDSA (con máxima longitud de 512 bits) o ED25519 tendremos que introducir los siguientes comandos:

```
ssh-keygen -f /etc/ssh/ssh_host_ecdsa_key -t ecdsa -b 521
ssh-keygen -f /etc/ssh/ssh_host_ed25519_key -t ed25519
```

AUTENTICACIÓN EN SSH: TODOS LOS MODOS EXPLICADOS EN DETALLE

En este apartado os vamos a enseñar los diferentes métodos de autenticación que tenemos disponibles en el servidor. Normalmente tenemos dos métodos principalmente: usuario y contraseña (algo que sabemos), y también con claves criptográficas (algo que tenemos). No obstante, podemos dotar al sistema de una seguridad adicional, combinando estas autenticaciones con por ejemplo un One Time Password generado por una aplicación como Google Authenticator o Latch OTP. Además, también podremos dotar al sistema de Latch, para evitar que cualquier usuario inicie sesión si no tenemos el «pestillo» abierto.

USUARIO Y CLAVE

Si queremos habilitar el login en el servicio a través del usuario y contraseña del sistema, el archivo de configuración deberá tener esta sentencia:

- PasswordAuthentication yes

De lo contrario, si queremos impedir la autenticación a través de usuario/clave, y permitir únicamente las conexiones a través de claves criptográficas, deberemos indicar no:

- PasswordAuthentication no

Esta sentencia afecta a todos los usuarios del sistema. Para no quedarnos sin acceso al servidor, deberíamos asegurarnos de que la sentencia **PubkeyAuthentication** esté configurada a «yes», para permitir el inicio de sesión con claves criptográficas.

Hay otra sentencia relacionada con esto llamada **ChallengeResponseAuthentication**, si ponemos la configuración a «no», no permitirá las conexiones donde se interactúe con el teclado, por lo que si por ejemplo tenemos configurado un One Time Password, no podremos iniciar sesión en el sistema. Si únicamente vamos a usar claves criptográficas, podréis ponerlo a «no» sin problemas.

CLAVE PÚBLICA SSH

Para configurar el acceso con clave pública al servidor, deberemos poner la sentencia siguiente a «yes»:

- PubkeyAuthentication yes

Así es como activamos la configuración con clave pública SSH en el sistema, no obstante, aún hay algunos pasos que deberemos hacer para que nos podamos conectar a dicho servidor, y es pasar la clave pública al propio equipo. Para hacerlo, deberemos permitir (de momento) la autenticación con usuario/clave, una vez que terminemos todos los pasos podremos denegar la autenticación con usuario y contraseña sin ningún problema.

Desde el ordenador donde nos queramos conectar al servidor con claves criptográficas, debemos crear dichas claves y pasárselas al servidor. Para crear unas claves RSA de 4096 bits tenemos que poner en el cliente SSH la siguiente orden:

```
ssh-keygen -t rsa -b 4096
```

En el asistente de generación de estas claves, nos pondrá si queremos guardarlas en /home/usuario/.ssh/id_rsa, le decimos que sí. Posteriormente nos permitirá poner a la clave privada una contraseña de paso, de esta forma, si perdemos la llave privada no pasará nada porque no podrán conectarse, debido a que es necesario siempre introducir una contraseña de paso para poder realizar la conexión correctamente.

Una vez que hayamos creado la clave pública y privada en nuestro equipo, debemos **enviar la clave pública al servidor SSH** donde nos queramos conectar, ojo: la clave pública.

```
ssh-copy-id usuario@IP_servidor
```

Automáticamente la clave pública se copiará a dicho servidor, y ya podremos habilitar la autenticación con solo clave pública y automáticamente nos habremos dado de alta. La salida ofrecida por este comando debería ser similar a esto:

```
The authenticity of host '12.34.56.78 (12.34.56.78)' can't be established.  
RSA key fingerprint is b1:2d:33:67:ce:35:4d:5f:f3:a8:cd:c0:c4:48:86:12.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added '12.34.56.78' (RSA) to the list of known hosts.
```



```
user@12.34.56.78's password:  
Now try logging into the machine, with "ssh 'user@12.34.56.78'", and check in:  
~/.ssh/authorized_keys  
to make sure we haven't added extra keys that you weren't expecting.
```

En esta salida, el usuario deberá confirmar que quiere añadir la identidad e introducir las credenciales de login para la cuenta que se quiere utilizar en ese servicio. Por este motivo es importante que en el servidor aún mantengamos la posibilidad de autenticarnos con usuario/clave. Una vez completado este proceso, tendríamos que ser capaces de hacer inicie de sesión en este equipo sin introducir la contraseña:

```
ssh usuario@IP
```

Recordad poner la directiva «PasswordAuthentication no» para no permitir accesos vía usuario y clave.

USUARIO Y CLAVE CON ONE TIME PASSWORD

En esta parte del manual, vamos a autenticarnos con el usuario/contraseña de siempre, pero además, deberemos usar un OTP con Google Authenticator o Latch OTP para entrar en la sesión. Es decir, dotaremos al servidor SSH de verificación en dos pasos.

Lo primero que debemos hacer es instalar una serie de dependencias necesarias para poder configurar la doble autenticación en nuestro servidor SSH. Para ello abriremos un terminal y teclearemos:

```
sudo apt install libpam0g-dev make gcc wget ssh
```

Una vez que las dependencias están instaladas en nuestro sistema ya podemos descargar el software para la doble autenticación.

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

Ya tenemos instalado Google Authenticator en nuestro sistema operativo. Los siguientes pasos a seguir son la configuración de la herramienta para poderla utilizar en nuestro SSH.

Para comenzar con la configuración de Google Authenticator simplemente debemos teclear en el terminal:

```
google-authenticator
```

NOTA: Ejecutadlo sin permisos de root (sudo), es decir, ejecutadlo como usuario «normal». Si lo ejecutáis como root, ese Google Auth solo estará disponible en el propio root, y al deshabilitarlo en el propio SSH no hará nada.

A continuación, veremos un sencillo asistente desde el terminal. Lo primero que nos preguntará es si queremos que nuestros tokens de acceso estén basados en el tiempo. A continuación, veremos la clave privada, la clave de verificación y los códigos de emergencia si no tenemos nuestro móvil a mano. Debemos guardar todos estos datos de

forma segura de manera que podamos recuperar el acceso en caso de pérdida de la clave de autenticación.

Después le decimos que guarde los cambios en el archivo de nuestra carpeta /home y nos preguntará si queremos que cada token sea utilizado una única vez, aunque eso limite a un inicio de sesión cada 30 segundos. Para protegernos frente a posibles ataques MITM seleccionamos que sí. Por último, nos preguntará si queremos ampliar el periodo de validez de cada código en lugar de solo 1 minuto y 30 segundos (para evitar problemas de sincronización de tiempo). Para evitar ataques de fuerza bruta también podemos limitar las conexiones a 3 por cada 30 segundos.

El escaneo del código QR o la introducción manualmente de código se puede realizar en programas como Google Authenticator, Authy, Latch y un largo etcétera. Recomendamos el uso de Google Authenticator o Latch.

El siguiente paso que debemos hacer es abrir el fichero de configuración de «sshd» para indicarle que utilice este módulo para el inicio de sesión. Para ello podemos hacer un «clear» para limpiar el terminal y teclear en él:

```
sudo nano /etc/pam.d/sshd
```

Y añadiremos al final del fichero la siguiente línea:

```
auth required pam_google_authenticator.so
```

Guardamos los cambios y abrimos el fichero sshd_config con el siguiente comando:

```
sudo nano /etc/ssh/sshd_config
```

Y cambiamos la línea «ChallengeResponseAuthentication no» por «ChallengeResponseAuthentication yes».

Reiniciamos el servidor con «sudo /etc/init.d/ssh restart» y una vez que vuelva a arrancar, ya tendremos la autenticación en dos pasos habilitada.

Una vez hecho, con el inicio de sesión también nos pedirá una clave de un solo uso, el código generado por nuestra aplicación móvil.

CLAVE PÚBLICA SSH CON ONE TIME PASSWORD

La configuración a nivel de clave pública debe ser exactamente igual que antes, y a nivel de instalación y configuración del Google Authenticator también. La única diferencia es que en el fichero **sshd_config** deberemos tener algo así:

```
PasswordAuthentication no
ChallengeResponseAuthentication yes
PubKeyAuthentication yes
UsePAM yes
AuthenticationMethods publickey,keyboard-interactive
```

Y en el fichero /etc/pam.d/sshd debemos tener algo como esto:

```
#@include common-auth
auth required pam_google_authenticator.so
```

Muy importante poner una almohadilla (#) para comentar el @include, de esta forma, nos autenticaremos correctamente con clave pública más el código OTP generado por el móvil. De esta forma, estaremos diciéndole al servidor que acepte autenticación con clave pública únicamente.

CONFIGURACIÓN DE LATCH EN OPENSSH Y ELEGIR MÉTODO DE AUTENTICACIÓN

Latch para OpenSSH actualmente tiene la limitación de que solo podremos usarlo para un usuario, es decir, si vamos a intentar iniciar sesión en el servidor SSH con varios usuarios, no podremos proteger nuestro equipo con Latch porque está limitado a solo uno. Os recomendamos [acceder a la web oficial del plugin Latch para Unix en GitHub](#), ahí encontraréis el manual paso a paso para configurarlo correctamente con el servicio SSH si os interesa. Debemos recordar que solo permite parear un usuario.

MEDIDAS DE SEGURIDAD ADICIONALES RECOMENDABLES

En este apartado vamos a usar software adicional para proteger el servidor de SSH, ya que es fundamental para tener una seguridad adicional. Usaremos tanto el firewall basado en iptables, como port-knocking para ocultar siempre el servicio detrás del firewall, así como fail2bat y DenyHost para detectar posibles ataques de fuerza bruta y pararlo con el firewall.

CONFIGURACIÓN DE FIREWALL IPTABLES PARA PROTEGER SSH

El firewall de iptables incorporado en la mayoría de sistemas basados en Linux, nos va a permitir limitar el número de conexiones simultáneas a nuestro servidor, y también, vamos a poder usar un módulo especial llamado «recent» para guardar en una base de datos todos los intentos de conexiones, ideal para evitar ataques de fuerza bruta.

Limitar el número de conexiones simultáneas en iptables

Si un atacante intenta conectarse múltiples veces desde la misma IP, podremos limitar dicho número de conexiones para mitigar su ataque. Esto no corta de raíz un posible ataque, sino que lo mitiga con dos objetivos: no tener un gran consumo de memoria y CPU en el equipo por abrir múltiples conexiones SSH, retrasar un posible ataque por fuerza bruta abriendo múltiples conexiones.

Esta sentencia ACEPTA hasta 5 conexiones que provengan de la misma IP pública, a partir de la sexta conexión lo bloqueamos. Si tenemos varias interfaces de red, deberemos usar el «-i» para poner por qué interfaz estamos aplicando esta regla, si no ponemos nada se aplicará a todas ellas.

```
iptables -A INPUT -p tcp --dport 22445 -m connlimit --connlimit-upto 5 --connlimit-mask 32 -j ACCEPT
```

Configurar el módulo recent para mitigar ataques de fuerza bruta en el servicio SSH

El módulo recent sirve para limitar el número de conexiones por segundo a nivel de IP, esto es ideal para protegernos de ataques al puerto SSH porque un atacante probará múltiples contraseñas. En el siguiente ejemplo, daremos de alta en una base de datos todas las conexiones al puerto TCP 22445 de destino, y daremos de alta el origen de esas conexiones (la IP pública de un posible atacante).

Posteriormente, chequeamos que en la tabla no haya más de 4 coincidencias en los últimos 60 segundos. Si hay más de 4 coincidencias, procedemos a bloquear todas las conexiones a partir de ahí.

```
iptables -A INPUT -p tcp --dport 22445 -m conntrack --ctstate NEW -m recent --set -  
-name ssh --rsource
```

Este módulo se diferencia del anterior, en que comprueba por tiempo estas conexiones. Normalmente para proteger el servicio SSH es mejor utilizar este módulo recent que el anterior.

PORT-KNOCKING PARA «ESCONDER» EL SERVICIO SSH

Port knocking (tocar puertos) es un método discreto de abrir puertos que, por defecto, el firewall mantiene cerrado. Funciona requiriendo intentos de conexión a una serie de puertos predefinidos cerrados. Cuando la secuencia correcta de “toques» a puertos (intentos de conexión) es recibida, el firewall abre entonces cierto(s) puerto(s). El beneficio es que, en un escaneo de puertos normal, parecería que el servicio del puerto simplemente no está disponible.

Port Knocking es una aplicación, que por defecto no se encuentra instalada en los sistemas operativos, podéis instalarla a través de los repositorios oficiales. En nuestro caso, al usar Debian 9, la instalación sería así:

```
sudo apt-get install knockd
```

Una vez instalado, debemos habilitarlo editando:

```
sudo nano /etc/default/knockd
```

Y poniendo: **START_KNOCKD=1**

A continuación, iniciamos el servicio:

```
sudo service knockd start
```

El fichero de configuración de knockd está en **/etc/knockd.conf**

Debemos editarlo con cualquier editor de archivos, necesario permisos de root. El fichero tendrá el siguiente aspecto:

[options]
UseSyslog

Y para poder «abrir» el puerto del servicio SSH, deberemos poner en consola «knock IP_address 7000 8000 9000». Una vez que hayamos utilizado el servicio, podremos cerrarlo con la sentencia que hayamos puesto anteriormente en el fichero de configuración, no tiene por qué ser justo al revés (depende de cómo hayas configurado el fichero de arriba).

Otra opción que tenemos en Port Knocking es el desbloqueo temporal del servidor, de esta manera, abriremos el puerto durante 10 segundos (o los que quieras), y es entonces cuando deberemos iniciar sesión en el servicio. Posteriormente, el puerto se cerrará y si estamos logueados no nos echará de dicho servidor.

[options]
UseSyslog

De esta forma, al salirnos del servidor no hará falta que «cerremos» la puerta como ocurría en el caso anterior, ya que solo ha estado abierta durante 10 segundos.

FAIL2BAN PARA MITIGAR ATAQUES DE FUERZA BRUTA EN SSH

Podemos también instalar el programa fail2ban para banear IPs que hagan muchos intentos de conexión fallidos (que metan mal la clave). Este programa es muy conocido y utilizado, ya que es muy fácil de configurar y poner en marcha. Podemos instalarlo poniendo

```
sudo apt install fail2ban
```

Ahora copiamos el archivo .conf en el mismo archivo .local para que se aplique esta configuración:

```
sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
```

Ahora podemos crear con fail2ban una regla personalizada para SSH, creamos un nuevo archivo de configuración que se encargará de sobrescribir el archivo principal jail.local anterior.

```
sudo nano /etc/fail2ban/jail.d/sshjail.local
```

Y pegamos esto:

```
[sshd]
enabled                =                true
port                   =                22445
bantime                =                3600
findtime               =                600
filter                 =                sshd
maxretry = 3
```

Y lo iniciamos, paramos y reiniciamos haciendo como si fuera el SSH, en este caso start=iniciar:

```
sudo /etc/init.d/fail2ban restart
```

Para ver los LOGS de conexión tendréis que mirar en la ruta /var/log/auth.log, y para ver los logs de fail2ban tenéis que mirarlos aquí: /var/log/fail2ban.log

DENYHOSTS: COMO FAIL2BAN PERO ORIENTADO ESPECÍFICAMENTE A SSH

También tenemos otro programa dedicado exclusivamente al SSH, se llama DenyHost y tiene una gran base de datos de IPs conocidas como atacantes. Es otra solución a posibles ataques si siempre tenemos nuestro servidor SSH expuesto a Internet. El funcionamiento de DenyHosts es igual que fail2ban, vigila los logs de conexión para detectar si estamos ante un ataque de fuerza bruta, y además comprueba bases de datos de IP para detectar si son botnets.

Para instalar este programa necesitaremos hacerlo desde los repositorios oficiales, suponemos que en /etc/apt/source.list:

```
sudo apt install denyhosts
```

La primera parte de la configuración se realiza en la ruta /etc/hosts.allow, aquí tendremos que poner la lista blanca con nuestros equipos para que nunca nos bloquee, esto es fundamental para no bloquearnos a nosotros mismos a través del firewall, sobre todo si estás usando un servidor remoto es realmente importante hacer este paso.

```
sudo nano /etc/hosts.allow
```

La sintaxis que debes usar es la siguiente:

```
sshd: DIRECCION_IP
```

Por ejemplo, con esto estarías permitiendo tres direcciones IP de origen:

```
sshd: 192.168.1.2
```

Guardamos para aplicar los cambios, y ahora tendremos que editar el fichero de configuración global de DenyHosts:

```
sudo nano /etc/denyhosts.conf
```

En este fichero de configuración veremos comentadas las diferentes opciones de configuración disponibles, la sintaxis es muy parecida a Fail2ban, por lo que no tendrás problemas a la hora de configurarlo.

PAM_TALLY2: OTRA ALTERNATIVA MUY RECOMENDABLE

Esta herramienta es muy parecida también a las anteriores, realiza exactamente la misma función: gestionar los intentos fallidos en el servidor SSH. Este programa nos permitirá

contar los intentos de acceso de diferentes usuarios con sus correspondientes direcciones IP de origen, si hay demasiados intentos de inicio de sesión fallidos, automáticamente se les denegará el acceso a través del firewall. Esta herramienta tiene dos partes, el `pam_tally2.so` que es el módulo PAM que podremos configurar en nuestro sistema operativo Linux, y también tendremos el `pam_tally2` que es el programa en sí mismo.

Por último, el programa **SSHGuard** también es muy recomendable para proteger adecuadamente nuestro servidor SSH de intentos de intrusión.

CONFIGURAR LATCH PARA PROTEGER EL USUARIO ADMINISTRADOR

Latch es una tecnología que nos permite proteger el acceso a nuestras identidades digitales, en este caso, podríamos utilizar Latch para proteger la cuenta de administrador en nuestro servidor o PC cuando vamos a acceder a él vía SSH. Esta funcionalidad nos permite añadir una capa más de seguridad, porque podremos controlar el acceso a una determinada cuenta a través de nuestro smartphone, bloqueando siempre o permitiendo los inicios de sesión. Si tenemos el Latch cerrado, aunque introduzcamos correctamente la contraseña de acceso no podremos iniciar sesión porque no tendremos permiso.

Esta herramienta también la podemos usar en cuentas de bancos, redes sociales, servidores OpenVPN y otros muchos lugares, como en un WordPress. Esto es ideal para añadir una capa más de seguridad al sistema. Lo primero que tenemos que hacer es registrarnos como desarrollador en el portal de Latch y añadir una aplicación nueva al sistema y configurarla con los parámetros que se nos piden.

Una vez que hemos hecho esos pasos que son necesarios, debemos entrar en el **GitHub de Eleven Paths** donde se encuentra **el plugin para SSH** el cual descargaremos de forma gratuita. Una vez que descargamos el .ZIP, procedemos a copiarlo y descomprimirlo en el sistema donde vayamos a asegurar el acceso vía SSH con Latch.

INSTALACIÓN DE LATCH PARA SSH EN UBUNTU

Os vamos a enseñar cómo configurar Latch con OpenSSH en Ubuntu, lo primero que debemos hacer es instalar el típico gcc, make y por supuesto OpenSSH además de las librerías necesarias. Instalamos gcc y make y todas las librerías necesarias:

```
sudo apt-get install gcc make libpam0g-dev libcurl4-openssl-dev libssl-dev
```

Una vez que hemos instalado todo lo necesario, procedemos a irnos a la carpeta de Latch para SSH que hemos descomprimido y ejecutamos el archivo llamado «install» de la siguiente forma:

```
sudo ./install
```

Empezará a compilar lo necesario para nuestra distribución y posteriormente lo instalará, en la siguiente captura de pantalla se puede ver este proceso:

```
latch-plugin-ssh-master: bash - Konsole
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
bron@bron-virtual-machine:~/Escritorio/latch-plugin-ssh-master$ sudo ./install
latch.conf
latch plugin installing...
rm -f /lib/security/pam_latch.so *.o
cc -Wall -fPIC -shared -Xlinker -x -o /lib/security/pam_latch.so ../latch.c pam_latch.c -lcurl -lcrypto -lpam
pam_latch.c: In function 'getAccountId':
pam_latch.c:80:9: warning: unused variable 'token' [-Wunused-variable]
    char * token = NULL;
        ^
running install
running build
running build_py
running install_lib
running install_egg_info
Removing /usr/local/lib/python2.7/dist-packages/easygui-0.96.egg-info
Writing /usr/local/lib/python2.7/dist-packages/easygui-0.96.egg-info
bron@bron-virtual-machine:~/Escritorio/latch-plugin-ssh-master$
```

Es recomendable reiniciar el servicio SSH una vez que se ha instalado:

```
sudo /etc/init.d/ssh restart
```

CONFIGURACIÓN DE LATCH.CONF CON ID DE APLICACIÓN Y CLAVE SECRETA

A continuación, nos metemos en la carpeta python y renombramos el archivo latch-model.conf a latch.conf y lo editamos para incorporar el ID de la aplicación y la clave secreta.

```
cd                                     python/
mv                                     latch-model.conf latch.conf
nano latch.conf
```

Y editamos el fichero de texto con la información de la web de Latch (ID de aplicación y «secreto») que tenemos al añadir una nueva aplicación:


```
# Configuration file for the latch SSH plugin
# Identify your Application# Application ID value
app_id = SrLv8S60NBjb0qI9EvQN
#Secret key value
secret_key = kr89S7D0I4qSzZ9NK4QWC0CXQdgxQ6HIXFrgwq2G
```

A continuación, podéis ver una captura de pantalla de cómo quedaría editado:

```
GNU nano 2.2.6 Archivo: latch.conf
#
# Configuration file for the latch SSH plugin
#
# Identify your Application
# Application ID value
#
app_id = SrLv8S60NBjb0qI9EvQN
# Secret key value
#
secret_key = kr89S7D0I4qSzZ9NK4QWC0CXQdgxQ6HIXFrgwq2G
```

Una vez editado guardamos y cerramos.

Ahora sólo nos quedaría parear nuestro dispositivo móvil con Latch en el sistema para empezar a funcionar.

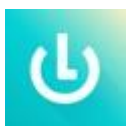
UTILIZACIÓN DE LATCH CON UN DISPOSITIVO ANDROID Y UBUNTU

Para utilizar Latch en un dispositivo móvil lo primero que tenemos que hacer es bajarnos la aplicación oficial de las distintas tiendas de aplicaciones:



DescargarQR-Code

Latch



DescargarQR-Code

Latch by ElevenPaths

Developer: **Telefonica Digital Espana S.L.U.**

Una vez que la hemos instalado en nuestro dispositivo móvil, la ejecutamos y nos aparecerá un asistente que nos dirá cómo funciona y para qué sirve la aplicación.



A continuación, tendremos que introducir la cuenta registrada en Latch, podemos introducir la cuenta de desarrollador sin problemas, los usuarios se registrarán una cuenta de usuario, pero los menús en el terminal son los mismos.



Empieza a proteger tus servicios con Latch

¿Ya tienes una cuenta?

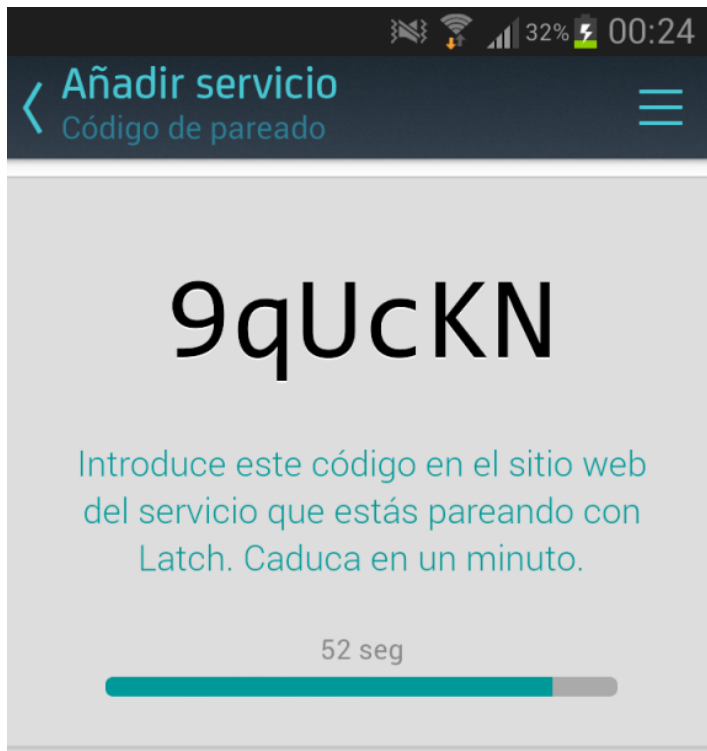
No, regístrate

Sí, inicia sesión

¿Alguna pregunta? Escribenos



Una vez que hemos iniciado sesión con nuestro usuario, tendremos un asistente para comenzar a parear servicios con nuestro terminal. Pinchamos en la parte inferior para que nos proporcione un código y parear el teléfono con el servicio OpenSSH.



En el sistema tenemos que teclear por terminal la siguiente orden:

```
sudo pairSSH CODIGO -f RUTA ARCHIVO .CONF
```

En nuestro caso estamos en /python/ por lo que hemos introducido la siguiente orden:

```
sudo pairSSH 9qUcKN -f latch.conf
```

Una vez que ejecutamos esta orden en el ordenador nos aparecerá que el servicio se ha pareado correctamente, a continuación podéis ver el mensaje exacto:

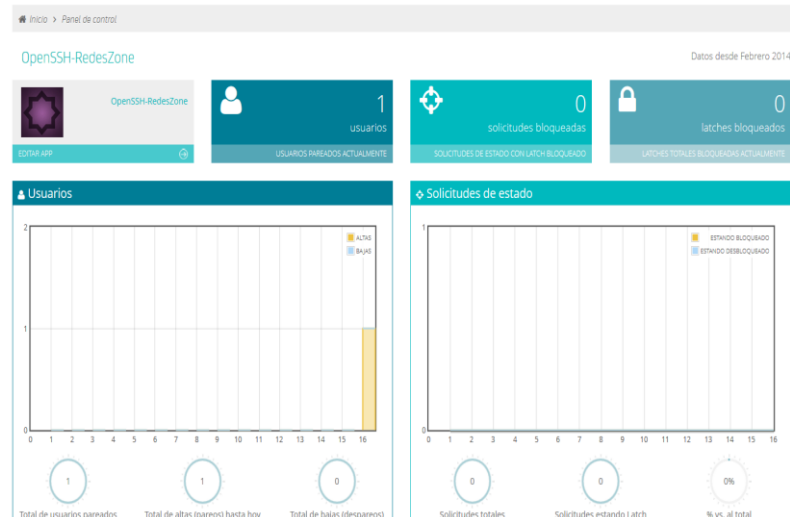
```
bron@bron-virtual-machine:~/Escritorio/latch-plugin-ssh-master/python$ sudo pairSSH 9qUcKN -f latch.conf
Account paired successfully
bron@bron-virtual-machine:~/Escritorio/latch-plugin-ssh-master/python$ █
```

Y de forma instantánea y automática, en nuestro terminal móvil nos aparecerá algo como esto:

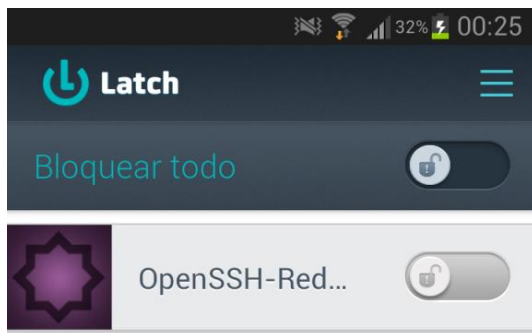


Por lo que ya habremos pareado nuestro servicio con un terminal móvil (un usuario), si entramos en el panel de control de Latch vía web nos aparecerá que tenemos un usuario registrado:

Panel de control



En la aplicación móvil podremos bloquear y desbloquear el servicio cuando queramos, simplemente tenemos que arrastrar el candado del servicio a bloquear.



Si pinchamos sobre el nombre o la foto del servicio, nos aparecerán las principales opciones que nos proporciona la aplicación Latch para móviles, como por ejemplo programar los bloqueos a unas determinadas horas del día:



PROBANDO EL FUNCIONAMIENTO DE LATCH CON OPENSSSH

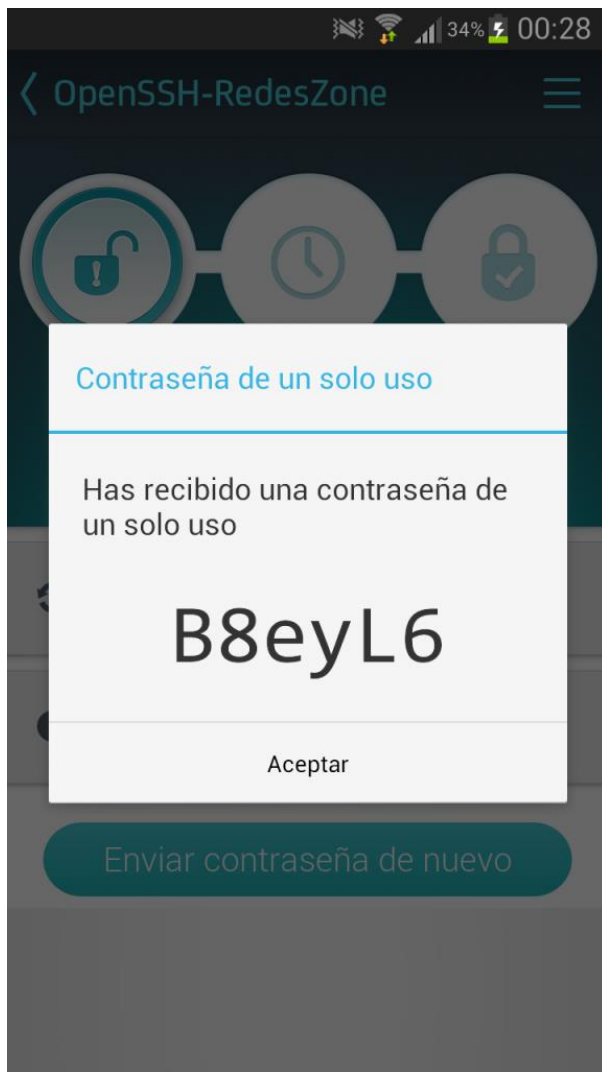
Una vez que está todo instalado y configurado, vamos a simular ser un usuario malintencionado que quiere acceder a nuestro servidor SSH. Como somos muy celosos de nuestra seguridad, siempre tendremos el Latch cerrado a menos que vayamos a utilizar el servicio.

Si entramos vía SSH al sistema protegido con Latch e introducimos la contraseña, en el smartphone nos aparecerá el siguiente aviso:



Si no fuiste tú y las alertas continúan, recomendamos cambiar tu contraseña y contactar al proveedor del servicio.

Si no desbloqueamos el servicio no podremos conectarnos, por tanto, procedemos a desbloquear el servicio y a intentar de nuevo la conexión. Como en el panel de control configuramos el OTP de forma obligatoria, siempre que intentemos iniciar sesión nos pedirá una contraseña de un solo uso, tal y como podéis ver en la siguiente imagen:



En el terminal del sistema Ubuntu podéis ver las órdenes que hemos introducido, el primer «Password:» corresponde al Latch cerrado, y el segundo «Password:» al Latch abierto, pero como tenemos un OTP, hasta que no introducimos la clave generada no nos dejará pasar:

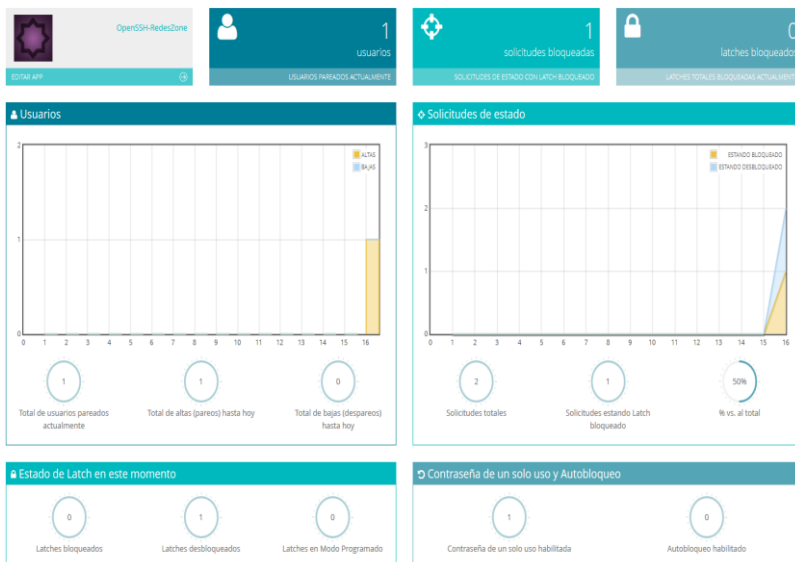
```
bron@bron-virtual-machine:~/Escritorio/latch-plugin-ssh-master/python$ ssh bron@localhost
Password:
Password:
One-time code: B8eyL6
Welcome to Ubuntu 14.04 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

32 packages can be updated.
19 updates are security updates.

Last login: Wed May  7 10:10:06 2014 from localhost
bron@bron-virtual-machine:~$
```

Si entramos en el panel de control de Latch veremos los intentos de intrusión que hemos tenido y también las claves OTP que hemos generado.



CONFIGURAR UN SEGUNDO USUARIO PARA ENTRAR VÍA SSH

Una vez que hemos instalado Latch, podremos configurar que el usuario «bron1» también pueda usar Latch para conectarse a su cuenta, sin embargo esta acción la debe realizar el usuario «bron1» de forma explícita, desde su terminal o sesión de usuario y haciendo exactamente que el primer usuario (pero con su código generado):

```
sudo pairSSH CODIGO -f latch.conf
```

Actualmente existe una limitación y es que no podremos tener dos usuarios bajo el mismo ID de aplicación en el terminal móvil con el mismo usuario, es decir:

Desde una misma cuenta Latch en el terminal móvil no podemos gestionar el usuario «bron» y «bron1» de forma separada, únicamente se pueden gestionar a la vez, ambos cerrados, ambos abiertos, no es como un segundo «servicio» en el terminal. Si queremos hacer que sean servicios separados, se deben utilizar cuentas diferentes. Según nos ha comentado Chema Alonso, en próximas versiones sí se permitirá este uso (perfecto para el usuario administrador y un usuario sin privilegios que se usa habitualmente).

CÓMO DAR DE BAJA UN USUARIO EN LATCH

Para dar de baja a un usuario pareado con Latch, basta con irnos a la carpeta python y ejecutar la siguiente orden:

```
sudo python unpair.py
```

```
bron@bron-virtual-machine:~/Escritorio/latch-plugin-ssh-master/python$ sudo python unpair.py
[sudo] password for bron:
Account unpaired successfully
```

De forma automática se despareará el usuario y será notificado a través del terminal móvil que se ha dado de baja el servicio.



Tu lista de servicios ha cambiado,
puedes consultar la lista actualizada
pulsando el botón de abajo.

Actualizar lista de servicios

CÓMO DESINSTALAR LATCH SSH DEL SISTEMA

Para desinstalar Latch del sistema basta con irnos a la carpeta python y ejecutar la siguiente orden:

```
sudo python uninstall.py
```

```
bron@bron-virtual-machine:~/Escritorio/latch-plugin-ssh-master/python$ sudo python uninstall.py
Uninstall completed
```

Y a continuación deberemos reiniciar el servicio OpenSSH:

```
sudo /etc/init.d/ssh restart
```

Tal y como habéis visto, Latch es un sistema de seguridad adicional realmente interesante para proteger nuestro servicio de SSH.

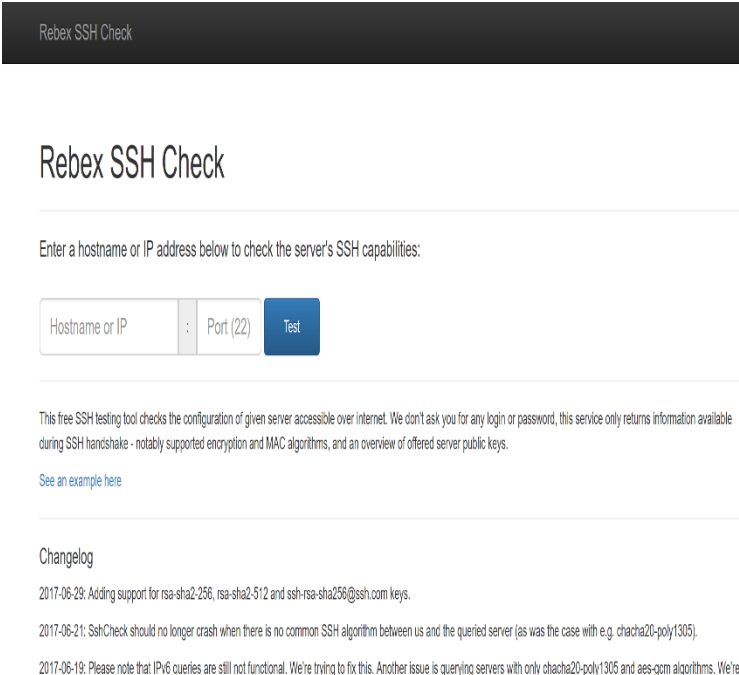
CÓMO COMPROBAR LA SEGURIDAD DE MI SERVIDOR SSH

Hoy en día disponemos de varias herramientas para comprobar la seguridad de nuestro servidor OpenSSH que acabamos de configurar, estos programas se encargarán de analizar la configuración del servidor realizando conexiones al servidor, con el objetivo de detectar algún tipo de configuración incorrecta o poco segura.

COMPRUEBA LA CONFIGURACIÓN CON REBEX SSH CHECK

La herramienta gratuita y online **Rebex SSH Check** realizará un escaneo rápido de los algoritmos de intercambio de claves, algoritmos de cifrado simétrico, algoritmo de clave, así como los algoritmos MAC que tenemos actualmente configurados en nuestro servidor SSH. Si hacemos uso de algún algoritmo que actualmente no se considera seguro, nos lo indicará, para posteriormente nosotros configurar el servidor SSH correctamente y quitar este algoritmo de los «permitidos».

Lo primero que tendremos que hacer es entrar en la web oficial, veremos que nos pide tanto nuestra dirección IP o dominio, así como el puerto donde esté escuchando el servidor SSH. Cuando hayamos introducido los datos de nuestro servidor SSH pinchamos en el botón de «TEST».



The screenshot shows the Rebex SSH Check website. At the top, there's a dark header with the text "Rebex SSH Check". Below it, the main heading "Rebex SSH Check" is displayed. A prompt says "Enter a hostname or IP address below to check the server's SSH capabilities:". Below this is a form with two input fields: "Hostname or IP" and "Port (22)", followed by a blue "Test" button. Under the form, there is a paragraph explaining that the tool checks server configuration over the internet and returns information available during the SSH handshake. A link "See an example here" is provided. At the bottom, there is a "Changelog" section with three entries dated 2017-06-29, 2017-06-21, and 2017-06-19, detailing updates and known issues.

Rebex SSH Check

Enter a hostname or IP address below to check the server's SSH capabilities:

Hostname or IP : Port (22) Test

This free SSH testing tool checks the configuration of given server accessible over internet. We don't ask you for any login or password, this service only returns information available during SSH handshake - notably supported encryption and MAC algorithms, and an overview of offered server public keys.

[See an example here](#)

Changelog

2017-06-29: Adding support for rsa-sha2-256, rsa-sha2-512 and ssh-rsa-sha256@ssh.com keys.

2017-06-21: SshCheck should no longer crash when there is no common SSH algorithm between us and the queried server (as was the case with e.g. chacha20-poly1305).

2017-06-19: Please note that IPv6 queries are still not functional. We're trying to fix this. Another issue is querying servers with only chacha20-poly1305 and aes-gcm algorithms. We're

El proceso de comprobar la seguridad del servidor SSH tarda unos 10 segundos aproximadamente, y nos informará de todos los algoritmos que podemos usar, y si son o no seguros. Por ejemplo, en nuestro caso con el servidor SSH de XigmaNAS con los valores por defecto, obtenemos que la seguridad es sobresaliente, ya que hace uso de todos los algoritmos de cifrado simétrico y asimétricos seguros, descartando la gran mayoría de los antiguos.

Rebex SSH Test result for

General information

| | |
|------------------------|---|
| Server Identification: | SSH-2.0-OpenSSH_8.2 FreeBSD-openssh-portable-8.2.p1_1.1 |
| IP Address: | |
| Generated at: | 2020-08-24 16:28:14 UTC (4 seconds ago) |

Key Exchange Algorithms

| | | |
|--------------------------------------|---|--------|
| diffie-hellman-group14-sha256 | Diffie-Hellman with 2048-bit Oakley Group 14 with SHA-256 hash ⓘ Oakley Group 14 should be secure for now. | Secure |
| diffie-hellman-group15-sha512 | Diffie-Hellman with 4096-bit MODP Group 15 with SHA-512 hash ⓘ | Secure |
| diffie-hellman-group18-sha512 | Diffie-Hellman with 8192-bit MODP Group 18 with SHA-512 hash ⓘ | Secure |
| diffie-hellman-group-exchange-sha256 | Diffie-Hellman with MODP Group Exchange with SHA-256 hash ⓘ | Secure |
| curve25519-sha256 | Elliptic Curve Diffie-Hellman on Curve25519 with SHA-256 hash ⓘ | Secure |

Esta herramienta también es capaz de comprobar los algoritmos de clave de servidor, tal y como podéis ver, nos informa que el utilizado con SHA-1 es «seguro» pero ya nos informa que está obsoleto, y que es recomendable utilizar ya siempre SHA2-256 o superior, por lo que en el fichero de configuración podríamos retirar este algoritmo.

También tendremos los algoritmos de cifrado simétrico disponibles, a nosotros solo nos aparece uno porque así lo tenemos definido en el fichero de configuración. Esta herramienta es muy útil también para verificar que efectivamente la configuración de seguridad es como nosotros deseamos.

Server Host Key Algorithms

| | | |
|--------------|--|--------|
| ssh-rsa | RSA with SHA-1 hash ⓘ SHA-1 is becoming obsolete. | Secure |
| ssh-rsa | RSA with SHA-1 hash ⓘ SHA-1 is becoming obsolete. | Secure |
| rsa-sha2-256 | RSA with SHA-256 hash ⓘ | Secure |
| rsa-sha2-256 | RSA with SHA-256 hash ⓘ | Secure |
| rsa-sha2-512 | RSA with SHA-512 hash ⓘ | Secure |
| rsa-sha2-512 | RSA with SHA-512 hash ⓘ | Secure |

Encryption Algorithms

| | | |
|------------|--|--------|
| aes256-cbc | AES with 256-bit key in CBC mode ⓘ CBC mode is not perfect, but still not "unsafe". | Secure |
|------------|--|--------|

Por último, también nos indica los algoritmos MAC que tenemos disponible en el servidor SSH, algunos de ellos tal y como podéis ver son considerados como inseguros, por lo que es recomendable retirar su soporte lo antes posible:

MAC Algorithms

| | | |
|-------------------------------|---|--------|
| umac-128-etm@openssh.com | 128-bit Universal Hashing MAC (Encrypt-then-MAC) by OpenSSH | Secure |
| hmac-sha2-256-etm@openssh.com | Hash-based MAC using SHA-256 (Encrypt-then-MAC) by OpenSSH | Secure |
| hmac-sha2-512-etm@openssh.com | Hash-based MAC using SHA-512 (Encrypt-then-MAC) by OpenSSH | Secure |
| umac-128@openssh.com | 128-bit Universal Hashing MAC by OpenSSH | Secure |
| hmac-sha2-256 | Hash-based MAC using SHA-256 | Secure |
| hmac-sha2-512 | Hash-based MAC using SHA-512 | Secure |
| umac-64-etm@openssh.com | 64-bit UMAC (Universal Hashing MAC) (Encrypt-then-MAC) by OpenSSH 64-bit UMAC is no longer considered secure enough. Recommended tag size should be at least 128 bits. | Weak |
| hmac-sha1-etm@openssh.com | Hash-based MAC using SHA-1 (Encrypt-then-MAC) by OpenSSH SHA-1 is becoming deprecated - consider replacing with SHA-256 or SHA-512. | Weak |
| umac-64@openssh.com | 64-bit UMAC (Universal Hashing MAC) by OpenSSH 64-bit UMAC is no longer considered secure enough. | Weak |
| hmac-sha1 | Hash-based MAC using SHA-1 SHA-1 is becoming deprecated - consider replacing with SHA-256 or SHA-512. | Weak |

Otros datos que es capaz de proporcionarnos este servicio es, si utilizamos algún tipo de compresión, y si la utilizamos, nos dirá de qué tipo es la compresión que tenemos activada en el fichero de configuración del servidor. Por último, también nos informará sobre la clave pública del servidor, incluyendo el fingerprint MD5, SHA2-256 e incluso la clave pública que utilizamos:

Server Public Keys

| | |
|----------------------|---------|
| ssh-rsa | |
| Key size: | 2048bit |
| MD5 Fingerprint: | |
| SHA-256 Fingerprint: | |
| Public key: | |

Tal y como habéis visto, gracias a esta gran herramienta online completamente gratuita, podremos comprobar de manera fácil y rápida la seguridad de nuestro servidor SSH.

REVISAR TU SERVIDOR CON SSH-AUDIT

ssh-audit es una herramienta totalmente gratuita, escrita en Python y que se encargará de escanear la configuración de nuestro servidor SSH, en esta ocasión, no tenemos una herramienta online, sino que la tendremos que ejecutar nosotros en el propio servidor donde queramos comprobar la seguridad de SSH. ssh-audit nos indicará si las diferentes configuraciones que hemos aplicado son seguras, inseguras, o tienen alguna debilidad, ideal para posteriormente realizar cambios en dicho servidor SSH.

Algunas de las principales características de esta herramienta gratuita, es que nos permitirá detectar el banner de inicio de sesión, si estamos usando un protocolo inseguro como SSH1, e incluso si estamos utilizando compresión con librería zlib. También será capaz de verificar los algoritmos de intercambio de claves, la clave pública del host, el cifrado simétrico cuando ya se ha establecido la comunicación, y también los mensajes de autenticación de la información.

Cuando ssh-audit haya analizado todos estos parámetros de manera totalmente automatizada, nos sacará un completo informe indicándonos desde cuándo está disponible una determinada opción, si ha sido eliminada, deshabilitada, si es inseguro, débil o si es seguro. Dependiendo de la severidad de la configuración realizada, podremos ver diferentes colores en los avisos.

```
ar@secbsd:~ % ./ssh-audit.py -v localhost | grep 'key.*info'
(key) ssh-rsa -- [info] available since OpenSSH 2.5.0, Dropbear SSH 0.28
(key) rsa-sha2-512 -- [info] available since OpenSSH 7.2
(key) rsa-sha2-256 -- [info] available since OpenSSH 7.2
(key) ssh-ed25519 -- [info] available since OpenSSH 6.5
ar@secbsd:~ % ./ssh-audit.py github.com
# general
[info] banner: SSH-2.0-libssh-0.7.0
[info] compatibility: OpenSSH 6.5-6.6, Dropbear SSH 2013.62+ (some functionality from 0.52)
[info] compression is enabled (zlib, zlib@openssh.com)

# key exchange algorithms
(key) curve25519-sha256@libssh.org -- [info] available since OpenSSH 6.5, Dropbear SSH 2013.62
(key) ecdh-sha2-nistp256 -- [fail] using weak elliptic curves
-- [info] available since OpenSSH 5.7, Dropbear SSH 2013.62
(key) diffie-hellman-group14-sha1 -- [warn] using weak hashing algorithm
-- [info] available since OpenSSH 3.9, Dropbear SSH 0.53
(key) diffie-hellman-group1-sha1 -- [fail] removed (in server) since OpenSSH 6.7, unsafe algorithm
-- [fail] disabled (in client) since OpenSSH 7.0, logjam attack
-- [warn] using small 1024-bit modulus
-- [warn] using weak hashing algorithm
-- [info] available since OpenSSH 2.3.0, Dropbear SSH 0.28

# host-key algorithms
(key) ssh-dss -- [fail] removed (in server) and disabled (in client) since OpenSSH 7.0
-- [warn] using small 1024-bit modulus
-- [warn] using weak random number generator could reveal the key
-- [info] available since OpenSSH 2.1.0, Dropbear SSH 0.28
(key) ssh-rsa -- [info] available since OpenSSH 2.5.0, Dropbear SSH 0.28

# encryption algorithms (ciphers)
(enc) chacha20-poly1305@openssh.com -- [info] available since OpenSSH 6.5
-- [info] default cipher since OpenSSH 6.9
-- [info] available since OpenSSH 3.7, Dropbear SSH 0.52
(enc) aes256-ctr -- [info] available since OpenSSH 3.7
-- [info] available since OpenSSH 3.7, Dropbear SSH 0.52
(enc) aes128-ctr -- [fail] removed (in server) since OpenSSH 6.7, unsafe algorithm
-- [warn] using weak cipher mode
-- [info] available since OpenSSH 2.3.0, Dropbear SSH 0.47
(enc) aes192-cbc -- [fail] removed (in server) since OpenSSH 6.7, unsafe algorithm
-- [warn] using weak cipher mode
-- [info] available since OpenSSH 2.3.0
(enc) aes128-cbc -- [fail] removed (in server) since OpenSSH 6.7, unsafe algorithm
-- [warn] using weak cipher mode
-- [info] available since OpenSSH 2.3.0, Dropbear SSH 0.28
(enc) blowfish-cbc -- [fail] removed (in server) since OpenSSH 6.7, unsafe algorithm
-- [fail] disabled since Dropbear SSH 0.53
-- [warn] disabled (in client) since OpenSSH 7.2, legacy algorithm
-- [warn] using weak cipher mode
-- [warn] using small 64-bit block size
-- [info] available since OpenSSH 1.2.2, Dropbear SSH 0.28

# message authentication code algorithms
(mac) hmac-sha1 -- [warn] using encrypt-and-MAC mode
-- [warn] using weak hashing algorithm
-- [info] available since OpenSSH 2.1.0, Dropbear SSH 0.28
(mac) hmac-sha2-256 -- [warn] using encrypt-and-MAC mode
-- [info] available since OpenSSH 5.9, Dropbear SSH 2013.56
(mac) hmac-sha2-512 -- [warn] using encrypt-and-MAC mode
-- [info] available since OpenSSH 5.9, Dropbear SSH 2013.56
ar@secbsd:~ %
```

Esta herramienta también nos mostrará la versión de SSH utilizada, además, es compatible con OpenSSH y Dropbear, los dos servidores SSH más utilizados en sistemas operativos y en dispositivos como routers, switches etc. Esta herramienta es más avanzada que la anterior, ya que nos proporcionará una mayor información.

Para usarla, lo único que tenemos que hacer es [bajarnos el archivo .py del GitHub oficial de ssh-audit](#), a continuación lo ejecutaremos como cualquier otra herramienta Python de la siguiente manera:

```
python ssh-audit.py [-nv] host[:port]
```

El argumento `-n` deshabilitará los diferentes colores en la salida de toda la información, y el `-v` nos mostrará absolutamente toda la información que nos brinda la herramienta.

MONITORIZA LOS LOGS DE OPENSSH CON SSHGUARD

SSHGuard es un programa que nos permitirá monitorizar los logs de nuestro servidor SSH, para detectar posibles ataques de fuerza bruta contra los usuarios que tengamos permitidos para conectarse a nuestro servidor. Para una buena administración de los sistemas informáticos de una organización, es muy importante monitorizar los logs de diferentes servicios, y gracias a SSHGuard, podremos monitorizarlos para posteriormente detectar ataques y bloquear a los atacantes haciendo uso del firewall del propio sistema operativo.

Este software se encargará de monitorizar los logs en diferentes formatos, concretamente, es capaz de reconocer estos logs en diferentes formatos:

- macOS
- metalog
- multilog
- raw log files
- syslog
- syslog-ng
- systemd journal

SSHGuard no solamente permite proteger nuestro servidor SSH, sino que también está preparado para escanear, detectar y proteger otros servicios fundamentales en cualquier sistema informático. En un principio solamente era compatible con OpenSSH, pero ha ido evolucionando y actualmente disponemos de todos estos servicios:

- OpenSSH (Servidor SSH)
- Sendmail (Servidor de correo)
- Exim (Servidor de correo)
- Dovecot (Servidor de correo)
- Cucipop (Servidor de correo)
- UWimap (Servidor de correo)
- vsftpd (Servidor FTP/FTPES)
- proftpd (Servidor FTP/FTPES)
- pure-ftpd (Servidor FTP/FTPES)
- FreeBSD ftpd (Servidor FTP/FTPES)

Cuando lee los logs de los diferentes servicios del sistema, y detecta algún tipo de ataque, automáticamente lo bloqueará utilizando el firewall del sistema operativo. Lógicamente, dependiendo del sistema operativo tendremos un firewall en concreto instalado, actualmente es compatible con los siguientes cortafuegos de sistemas basados en Unix y Linux:

- FirewallD
- ipfw
- IPFILTER

- netfilter/iptables
- netfilter/ipset
- PF
- tcpd's hosts.allow
- IBM AIX's firewall

Otras opciones interesantes son que podremos crear una lista negra de direcciones IP automáticamente, además, también es capaz de supervisar varios archivos de registro simultáneamente. En caso de utilizar el protocolo IPv6 en tu red profesional o doméstica, estás enhorabuena porque SSHGuard tiene soporte completo para el protocolo IPv6.

INSTALACIÓN Y PUESTA EN MARCHA

Esta gran herramienta para proteger nuestro servidor SSH, está disponible en los principales repositorios de software de distribuciones Linux como Debian, ArchLinux, Ubuntu, OpenSUSE, y también en el sistema operativo FreeBSD basado en Unix. La instalación se debe realizar a través del gestor de paquetes de tu distribución, en caso de no existir, siempre te podrás descargar el software y compilarlo tú mismo, podéis acceder a la [web oficial de SSHGuard](#) para acceder a su descarga.

```
bron@redeszone:~$ sudo apt install sshguard
[sudo] password for bron:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Los paquetes indicados a continuación se instalaron de forma automática y ya no son necesarios.
  libmemcached11 libmemcachedutil2
Utilice «apt-get autoremove» para eliminarlos.
Se instalarán los siguientes paquetes NUEVOS:
  sshguard
0 actualizados, 1 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 103 kB de archivos.
Se utilizarán 342 kB de espacio de disco adicional después de esta operación.
Des:1 http://ftp.es.debian.org/debian/ jessie/main sshguard amd64 1.5-6 [103 kB]
Descargados 103 kB en 0s (1.030 kB/s)
Seleccionando el paquete sshguard previamente no seleccionado.
(Leyendo la base de datos ... 126549 ficheros o directorios instalados actualmente.)
Preparando para desempaquetar .../sshguard_1.5-6_amd64.deb ...
Desempaquetando sshguard (1.5-6) ...
Procesando disparadores para man-db (2.7.0.2-5) ...
Procesando disparadores para systemd (215-17+deb8u6) ...
Configurando sshguard (1.5-6) ...
Procesando disparadores para systemd (215-17+deb8u6) ...
bron@redeszone:~$ █
```

Para la puesta en marcha de SSHGuard lo primero que debes hacer es configurar el sistema de logs en tu servidor. Si no sabes hacerlo, exista una magnífica [documentación en la web oficial del software](#). Después tendrás que configurar ciertos parámetros en el firewall, para que SSHGuard sea capaz de bloquear las direcciones IP de los posibles atacantes que tengamos de manera completamente automática y sin intervención del administrador de redes o sistemas.