SSH keygen y fail2ban.

Publicado el lunes 28 de septiembre de 2015.

Actualizado el jueves 23 de julio de 2020

Con SSH podremos conectarnos de manera segura a nuestras máquinas remotas y ejecutar o automatizar tareas con la certeza de que nadie podrá indagar qué estamos haciendo, con privacidad absoluta.

En un entrada anterior vimos y aprendimos nuestros primeros pasos para trabajar por línea de comandos (*shell*) en una ventana terminal. Si no lo recordáis o queréis refrescar la memoria pues visitadlo y regresad por acá para continuar nuestro aprendizaje (que yo también escribo esto para que no se me olvide -y aprovecho de ayudar a otros también-).

Pues el tema que nos trae hoy consta de dos publicaciones de la cual esta es la primera. Ya sabemos cómo manejar la terminal por atajos de teclado para listar, copiar archivos, renombrarlos, etc. Todo esto lo hacíamos así, directamente sentados en nuestro ordenador, en contacto físico con nuestro equipo. ¿Pero que tal si nosotros necesitamos conectarnos a alguno de los ordenadores de nuestro trabajo? ¿O, si estamos de viaje, necesitamos conectarnos a nuestra computadora en casa?

Apartando el tema que necesitamos conocer la dirección IP que nos asigna nuestro proveedor de internet para conectarnos (eso ya es tema para una nueva publicación) en el mundo GNU/Linux hay software antiquísimo diseñado especialmente para esto. En un principio nos ocasionará temor el dejar nuestros equipos así, abiertos al mundo y con tanto "*cracker*" por allí suelto, que ni siquiera el mayor imperio conocido en la historia de la humanidad ha podido capturar a Edward Snowden o ~~a~~ Julian Assange~~,~~ ¿qué quedará para nosotros? Al final de estas líneas os mostraremos al menos una herramienta para ayudar a protegernos de accesos no autorizados.

La buena noticia -y que tal vez los tranquilice un poco- es que estos dos señores (y muchos más) utilizan lo que nosotros aprenderemos aquí. Se dice rápido y fácil pero detrás de esto mucha gente se devanó los sesos pensando fórmulas matemáticas, estadística y algoritmos para garantizar nuestro acceso a ordenadores remotos. Sin más preámbulos pasemos a describir dicha tecnología.

SSH "Secure SHell"

En informática el acrónimo de tres letras SSH ("*Secure SHell*") describe la conexión entre dos computadores por medio de un canal seguro basado en un protocolo de red que nos permite manejar una línea de comandos como si estuviéramos de cuerpo presente frente al ordenador. La definición en inglés es cortísima, pero bueno cosas de ese idioma, todo es corto y rápido, "como para ayer pues". Una cosa que yo admiro, eso sí, de esa gente es LA PRACTICIDAD. Estas conexiones nos permiten administrar múltiples equipos, ***pero el asunto va mucho más allá.*** Yo me esfuerzo todos los días para llegar a ser un Administrador de sistemas perezoso y ser así algún día UN BUEN ADMINISTRADOR DE SISTEMAS. Podemos crear archivos de procesos por lotes escritos en formato **.sh** (bin bash) y que nuestra máquina local ejecute cada cierto tiempo prefijado con CRON pero lo aplique a una máquina remota (*la idea es poder introducir variables de búsqueda o parámetros a programas desde nuestro equipo local y aplicarlo en la máquina remota*).

Para ello debemos nosotros mismos introducir la contraseña *de la máquina remota* y hasta allí llega nuestro entusiasmo para automatizar tareas. Pero para ello inventaron el ssh-keygen, un algoritmo que nos permitirá conectar automáticamente "sin contraseña" pero con una seguridad muy robusta y comprobada millones de veces diariamente.

Vuelvo y repito, detrás de todo esto hay mucha ciencia y esfuerzo, lo que les describo es una simplificación para aprender el concepto que luego pueden ampliarlo con los variados enlaces que incluyo y que utilicé para escribir esta entrada en mi blog.

Cinco pasos para crear "conexión sin contraseña".

Tanto como "sin contraseña" no es el asunto, realmente tendremos que colocar una sola vez nuestra contraseña para poder acceder a nuestra página remota. Luego son cinco los pasos necesarios para crear nuestras conexiones "sin contraseñas" o como dicen en inglés "SSH Passwordless Login Using SSH Keygen in 5 Easy Steps". Dichos pasos, panorama general, son los siguientes:

1. En nuestra computadora local generaremos una llave digital, para ser exacto dos llaves, una pública y otra privada, ambas relacionadas matemáticamente la una con la otra.
2. Luego crearemos un directorio en la máquina remota donde reposará dicha llave digital.
3. Acto seguido copiaremos la llave en sí (local->remota).
4. Asignaremos en la máquina remota permisos de lectura a dicha llave.
5. Nos conectaremos como prueba de que la llave funciona.

Fue más fácil decirlo que hacerlo, como siempre. Ahora paso a describir esto con un ejemplo, en mi caso tengo una computadora real corriendo Ubuntu 14 de 64 bits (máquina local llamada "KEVIN" cuya dirección IP local es 192.168.1.47) la cual ejecuta un software llamado VirtualBox que me permite correr máquinas con diferentes sistemas operativos. Para nuestras pruebas nos vienen como anillo al dedo pues ahorramos en hardware y en tiempo. La máquina virtual que

utilizaré tiene corriendo Debian 7 "wheezy" la cual será la **máquina remota**, su nombre es "**postgresql**" y su dirección ip de área local es, para este ejemplo, 192.168.1.27 (una máquina virtual con motor de base de datos que utilizo para pruebas de programación).

Instalando SSH en la máquina remota (Debian).

Debido a la sencillez de este paso no se los había mencionado pero es que tampoco podemos obviarlo porque es crucial para aplicar los 5 pasos que nos fijamos como meta. Por seguridad (pienso yo) el SSH **no viene instalado de manera prefijada en las distribuciones Linux.** Pensemos esto como que si no existe una puerta nunca podremos entrar. Lo que vamos a hacer ni siquiera es abrir una puerta, *lo que vamos a realizar es romper la pared y empotrar nuestra puerta.*

Es así que PRIMERO debemos estar sentados en la que será la máquina remota (-si se pierden retrocedan un párrafo y se ubican-) y abrir una ventana terminal y ejecutar la siguiente sentencia:

```
apt-get install ssh
```

Debemos estar conectados como superusuario, en esta oportunidad utilicé el comando **su** introduje mi contraseña de usuario y ordené la instalación del software necesario para hacer la conexión (recordar que la máquina remota ejecuta Debian).

Como ya estaba instalado me reporta que no fue instalado ni modificado nada y nos recuerda que hay 2 actualizaciones por realizar (de nuevo esta parte da material para otra entrada aparte en mi blog). Ya con esto establecido podremos continuar ahora sí con los cinco pasos siguientes.

Bono adicional: instalando SSH en Ubuntu.

Pues eso, tengo una máquina virtual para probar Ubuntu de 64 bits con la versión 16.04 **Xenial Xerus** [Ubuntu da nombres rimbombantes a sus distros (abrid este enlace si queréis saber su significado en inglés), pero ¿quienes somos nosotros para criticarlos? *Hagamos nuestra propia distro para ponerle un nombre que nos guste).*

En dicha máquina virtual abrimos una ventana terminal con **CTRL+T (recordad de nuevo que es Ubuntu)** e introducimos el correspondiente comando:

En este ejemplo si que vemos cómo se instala completo, y notad la generación de certificados

digitales mínimos necesarios para funcionar. También, en la figura 2, veréis como la última línea se generan disparadores para **ufw**, el cortafuegos que utiliza Ubuntu, al final ampliaremos esto. Luego de realizado esto podemos probar la conexión y revisar si conecta correctamente:

Paso N° 1: creación de llave digital.

Tal vez esta sea la sección más llamativa del asunto. Necesitamos crear una llave digital que consiste en un montón de caracteres organizados según un algoritmo que hace esta llave única. Incluso tiene opciones para hacerla más segura aún, pero les dejo eso como tarea en otro artículo. Dicho esto usaremos la opción "simple", *sentados en la máquina local abrimos una ventana de terminal y tecleamos lo siguiente:*

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

Luego presionamos la tecla Intro (Enter) cuatro veces, osea, empleamos los valores que trae por defecto el programa generador de llaves. Aquí debemos anotar dónde guarda la llave que acabamos de generar: **en nuestra carpeta personal dentro de un directorio oculto llamado ssh,** en nuestro ejemplo es "/home/jimmy/.ssh/". Sabiendo esto nos vamos al siguiente paso.

Paso N° 2: directorio remoto para guardar llave.

Para copiar nuestra nueva llave digital en nuestra máquina remota necesitaremos introducir los siguientes comandos desde la máquina local:

```
ssh jimmy@192.168.1.27 mkdir -p .ssh
```

Como ven aquí ya comenzamos a correr comandos en la máquina remota (llamada "postgresql" y cuya dirección IP es "192.168.1.27"). Para ser más exactos ordenamos por medio del comando **mkdir** crea un directorio oculto (observar el punto delante) y con la opción **-p** le decimos que obvie el error *si ya existe el directorio.* Con esa opción la línea de comandos no devolverá respuesta alguna si todo va bien, ***el*** estoicismo ***de Linux es heredado del Unix.*** Ya habrán podido notar que la sintaxis del ssh consiste a dónde se va a conectar y si le colocamos algún otro comando seguido pues simplemente ejecuta ese comando en la máquina remota y "se desconecta".

Si solamente colocamos **"ssh usuario@máquina" (y por supuesto introducimos nuestra contraseña de usuario *de la máquina remota*)** quedará la ventana terminal abierta para teclear

los comandos que necesitemos, eso sí, cambiando el *prompt* o indicador (para nuestro ejemplo "jimmy@posgresql:~$") para que sepamos dónde estamos ubicados.

Paso N° 3: copiando la llave a la máquina remota.

En el paso N° 1 anotamos dónde se guardó nuestra llave digital en nuestro ordenador local (opción por defecto, presionamos Intro y escribir otra ruta). Ahora la copiaremos mediante:

```
cat .ssh/id_rsa.pub | ssh jimmy@192.168.1.27 'cat >> .ssh/authorized_keys
'
```

De nuevo introducimos nuestra contraseña de usuario en la máquina remota con la aclaratoria del uso del signo "tubería" **|** que se denota así, una barra vertical que indica pasar los resultados de un comando al otro.

Vemos que usamos **cat**,un comando que nos permite visualizar por pantalla el contenido de un archivo de texto (o cualquier otro archivo) sólo que con el uso del signo "tubería" lo que íbamos a ver por pantalla (contenido de la llave digital "**id_rsa.pub**") se lo enviamos al comando **ssh** quien a su vez, como tiene un comando a su derecha, lo pasa al interprete de comando de la máquina remota. *Aquí hagamos una pausa para asimilar el resto de la linea (3 comandos)*.

Así como el comando **cat** nos permite extraer y visualizar -por pantalla, por defecto-, *también* nos permite INTRODUCIR texto en un archivo con el uso de los dos signos "**>>**" pero cambiando el nombre a "**authorized_keys**"; si el archivo existiera pues lo agregaría al final (que una máquina remota puede servir para conectarnos desde varias máquinas locales diferentes, cada una con su propia llave digital). **De nuevo el estoicismo y sencillez de Linux se hace notar.**

Paso 4: asignando permisos a la llave copiada.

Para que nosotros y sólo nosotros seamos los únicos que podamos usar esa llave digital en la máquina remota, debemos asignarnos permisos para nosotros, *es decir, los otros usuarios registrados en esa máquina remota les cerramos la posibilidad de leer la llave que acabamos de copiar.*

Para ello ejecutaremos el comando **chmod** a la derecha del comando ssh, ya conocemos bien la sintaxis:

```
ssh jimmy@192.168.1.27 "chmod 700 .ssh; chmod 640 .ssh/authorized_keys"
```

Lo único nuevo aquí es el uso del punto y coma para separar un comando de otro dentro del conjunto delimitado por las comillas dobles. En la máquina remota se ejecutará tal cual, lee el primer comando de la izquierda hasta que encuentre el punto y coma, ejecuta y luego sigue leyendo hasta conseguir otro punto y coma (o se acabe la cadena con las comillas que sirve para delimitar).

Paso N° 5: comprobar que funciona.

El paso más sencillo: ejecutar la conexión y comprobar "que no nos pide contraseña", tecleando en la terminal lo siguiente:

Actualizado el sábado 24 de octubre de 2015:

Si quisiéramos conectarnos como **usuario raíz** o "**root**" el procedimiento necesitaría unos comandos -y ediciones- adicionales. ***No obstante*** la comunidad Linux recomienda NO hacerlo ***por razones de seguridad, lo mejor es*** crear un usuario en el grupo de "sudoers"***.***

Fail2ban para fortalecer servidor.

En esta segunda parte de la entrada nos dedicaremos a estudiar el programa **Fail2ban** para proteger nuestro servidor **SSH** revisando periódicamente los registros de acceso a nuestro servidor en búsqueda de síntomas que indiquen un ataque. Cuando un ataque es detectado, que cumplan los parámetros establecidos, Fail2ban actualizará nuestro **iptables** para bloquear la dirección IP del atacante (o al menos la dirección IP que dice ser) por un período de tiempo o incluso permanentemente. Fail2ban incluso nos avisará por correo electrónico del suceso, si le instalamos algunos programas adicionales. Por último tampoco podemos perder de vista el *Ubuntu Fire Wall* (**ufw**).

Instalación de Fail2ban.

Primero debemos verificar si nuestro servidor está actualizado con los últimos parches de seguridad (línea N° 1), luego instalaremos el Fail2ban en sí (línea N° 2), el programa **sendmail** si queremos recibir correo con informes de sucesos (línea N° 3) y configuraremos el cortafuegos (en este ejemplo, de Ubuntu) en las líneas N° 4 y 5:

```
apt-get update && apt-get upgrade -y apt-get install fail2ban apt-
```

```
get install sendmail ufw enable ufw allow ssh
```

Y las capturas de pantalla para que veáis el resultado de las líneas 2° a la 5°:

Al instalar sendmail el problema no es la descarga del programa, que es rápido, sino el tiempo que tarda instalándose ya que revisa todo el sistema para instalar certificados digitales para la seguridad TLS y muchos otros valores, así que paciencia con esto.

Configurando fail2ban.local

Fail2ban, al inciar en nuestro equipo remoto (que ya tenemos listo para conectarnos "sin contraseña") primero lee todos los archivos con extensión **.conf** (que trae por defecto) y luego lee los archivos con extensión **.local** que están ubicados en la carpeta de sistema **/etc/fail2ban.** Nosotros podemos aprovechar este comportamiento para nuestro beneficio copiando el archivo fail2ban.conf a fail2ban.local y editar la copia, dejando intacto nuestro archivo original (por si acaso algo sale mal, borramos fail2ban.local y repetimos el proceso):

Ahora abrimos con el editor de texto favorito (nosotros usamos **nano** en este caso) para ajustar los valores necesarios:

Podéis hacer clic en la imagen para ampliarla en otra pestaña (si su navegador lo permite). Los valores son los siguientes:

- **loglevel**: nos permite habilitar el nivel de detalle que nos presentará fail2ban: eventos críticos ("CRITICAL"), de error ("ERROR"), de advertencia ("WARNING"), nota ("NOTICE"), informativo ("INFO"), depuración ("DEBUG"). Por defecto está en "INFO", cambiad a vuestro gusto.
- **logtarget**: nos permite decidir a donde vamos a guardar el informe de fail2ban, por defecto en el archivo **/var/log/fail2ban.log** . Las otras opciones son STDOUT, STDERR y SYSLOG, esta última puede ser útil si queremos manejar por mensajes del sistema.
- **syslogsocket**: como dijimos en el apartado anterior, si seleccionamos SYSLOG para que fail2ba.log nos envíe el informe de sucesos, podemos configurar para que automáticamente utilice la vía por defecto o si queremos ubicar este en una ruta distinta (se puede dar el caso).
- **socket**: nos permite comunicarnos con el "demonio" que corre el servicio fail2ban.

Dejamos intacto esta línea, de lo contrario no podremos pasar las órdenes. En el mundo del software libre todo es cristalino y transparente, esta línea es útil a la hora de depurar errores de programación del fail2ban.

- **pidfile**: cada vez que se inicie, en este archivo se guardará el PID del servicio (Process Identification Number), esto es un identificador único a nivel del sistema oeprativo y sirve, por ejemplo, si necesitamos eliminar el proceso de la memoria.
- **dbfile**: fail2ban utiliza SQLite, un manejador ligero de base de datos, para guardar sus procesos en disco duro de manera indexada, lo cual nos permite buscar rápidamente el historial de eventos, así se detenga fail2ban. Permite establecer el proceso en memoria para rapidez -pero se perderán los datos al apagar el equipo- o simplemente no indexaremos nada (tal vez si nos conformamos con los avisos por correo electrónico pero pensamos que siempre necesitaremos saber más, por eso recomendamos dejarlo en su valor por defecto **/var/lib/fail2ban/fail2ban.sqlite3**). De este archivo podemos sacar datos si usamos la línea de comandos SQLite3.
- Si establecimos en el apartado anterior el uso del motor ligero de base de datos, podemos establecer en esta línea el tiempo máximo (en segundos) que se deben conservar los registros. Recomendamos establecerlo a 7 días en vez de las 24 horas que trae por defecto.

Configurando fail2ban.jail

En este archivo **fail2ban.jail** haremos lo mismo que hicimos con **fail2ban.conf**.: copiaremos el contenido de **fail2ban.jail** a **fail2.local** para conservar el archivo original. *Nota: los desarrolladores advierten que fail2ban.conf será cambiado en cualquier momento en que actualicemos a una nueva versión. De antemano ya habíamos previsto utilizar los archivos .local y así nos lo especifican en una nota al comienzo del archivo.*

Los valores que contiene son los siguientes (usamos de nuevo el editor nano con derechos de administrador):

- En la sección **[INCLUDES]** la etiqueta **before** indica que se lean primero los archivos .conf (ya describimos este comportamiento).
- En la sección **[DEFAULT]** la etiqueta **ignoreip** permite colocar las direcciones IP *a las cuales no le aplicaremos regla alguna.* Por defecto trae en formato CIDR a la dirección especial de equipo local 127.0.0.1/8 y si queremos o necesitamos podemos colocar direcciones ip específicas e incluso podemos bloquear servidores de nombres de dominio DNS. Debemos separar con un espacio los diferentes valores. Así podríamos, por ejemplo, especificar que no aplicase regla alguna a nuestros fallidos intentos de conexión SSH desde nuestra red de área local (generalmente 192.168.1.X) con la siguiente notación CIDR: 192.168.1.0/24 (una subnet y 255 direcciones IP).
- **maxretry** y **findtime**: el primer parámetro indica cuántos intentos fallidos y en cuál período de tiempo han de haber sucedido para bloquear la dirección IP del atacante. Por defecto el

número máximo de intentos son 5 en 600 segundos (10 minutos) o menos.
- **bantime**: si cumple con el punto anterior, en este parámetro especificamos por cuánto tiempo bloquearemos la dirección ip del atacante. Por defecto serán 600 segundos sin atender llamada desde esa dirección IP involucrada.
- **usedns**: esto lo explicaremos por la utilidad más notable, ver si la dirección IP es dinámica por medio del reverso del DNS. Al llegarnos un ataque podemos usar su dirección ip al DNS bajo la cual está bajo control a fin de determinar si es una dirección ip fija a un nombre de dominio o si es una dirección ip dinámica que cambia constantemente entre los clientes de un proveedor de internet (ISP). La segunda utilidad es poder marcar en un formato legible para nosotros los humanos en el archivo de registro de eventos. Por defecto viene marcada como advertencia.
- **logencoding**: permite codificar el archivo de registro, viene marcado como "auto" y esto quiere decir que se utilizará lo que tenga asignado el sistema operativo, Posibles valores: ASCII, UTF-8 (este último formato es útil para nosotros los latino hablantes: castellano, francés, italiano, etcétera).
- **enable**: por defecto es falso "false" y permite habilitar el archivo jail correspondiente.
- **filter**: permite que el archivo "interactúe" con otros mensajes del sistema, dándole el nombre propio del archivo para pasarlo como variable. No tocar esta línea para nada.

Hacemos una pausa para separar los valores destinados al envío de correos del sistema:

- **destemail**: por defecto root@localhost será quien esté informado de lo que esté sucediendo.
- **sender**: quien envía el correo, se repite el valor root@localhost, como vemos "yo con yo" no es útil así que en **destemail** colocaremos nuestra dirección de correo electrónico.

Fuentes consultadas.

Aquí les coloco los que visité y leí para poder escribir esta entrada "con sabor venezolano":

En idioma castellano.

- "GNU/Linux: linea de comandos (shell)".
- "Error conectando con un servidor remoto por SSH" por Alex.
- "Evitar ataques de fuerza bruta con fail2ban" por DavidOchoBits.
- «Evitar ataques de fuerza bruta en Nginx con Fail2ban» por DavidOchobits.

En idioma inglés.

- "ssh-keygen" at Wikipedia.
- "SSH Passwordless Login Using SSH Keygen in 5 Easy Steps" at Tecmint.com
- "How to generate ssh-key in ubuntu/CentOS" at SysAdminTutorial.Blogspot

- "How to setup passwordless SSH access for root user"at AskUbuntu.com
- "Using Fail2ban to Secure Your Server" by Linode.com
- "SQlite Home Page".
- "Certificate expiry: how to monitor the expiration of SSL certificates with Pandora FMS" at PandoraFMS.
- "SSH Essentials: Working with SSH Servers, Clients, and Keys" by Justin Ellingwood.
- "Understanding the SSH Encryption and Connection Process" by Justin Ellingwood.
- "How To Configure SSH Key-Based Authentication on a Linux Server" by Justin Ellingwood.
- "How To Use SSH to Connect to a Remote Server in Ubuntu" by Justin Ellingwood.

En idioma francés.

- "Se connecter en ssh avec une clé publique" Renaud M.G. (Responsive-Mind.fr).

Nuestro "tuiteo" acerca de este tema:

Tweets about #SSH from:@ks7000
!function(d,s,id){var js,fjs=d.getElementsByTagName(s)[0],p=/^http:/.test(d.location)?'http':'https';if(!d.getElementById(id)){js=d.createElement(s);js.id=id;js.src=p+"://platform.twitter.com/widgets.js";fjs.parentNode.insertBefore(js,fjs);}}(document,"script","twitter-wjs");