

Práctica 2: Instalación y configuración de servicios

Alonso Bueno Herrero

11 de agosto de 2020

Índice

1. Introducción

2. Sesión 1ª: Instalación del servicio SSH en Ubuntu

- 2.1. Instalación y pruebas del servicio
- 2.2. Configuración del servidor SSH mediante el fichero `/etc/ssh/sshd_config`
- 2.3. El cifrado por clave público-privada
 - 2.3.1. Contexto
 - 2.3.2. Gestión en Linux

3. Sesión 2ª (parte I): Instalación de SSH en CentOS

- 3.1. Instalación y puesta en marcha
- 3.2. Configuración del cortafuegos (en CentOS)
- 3.3. RECAPITULACIÓN: Comparativa de servicios frecuentes en Ubuntu Server *versus* CentOS 7

4. Sesión 2ª (parte II): Copias de seguridad y control de versiones

5. Sesión 3ª (parte I): Servidores Web. Instalación y configuración de HTTP en Ubuntu y CentOS.

- 5.1. Conceptos previos
- 5.2. Instalación del servidor web en Ubuntu Server
- 5.3. Instalación del servidor web en CentOS Server.

6. Sesión 3ª (parte II): Seguridad y análisis en servidores: fail2ban y rkhunter

- 6.1. Introducción a **fail2ban**
 - 6.1.1. Tareas básicas con fail2ban.
- 6.2. Introducción a **rkhunter** (*Root Kit Hunter*)

Referencias

- [1] <https://www.redeszone.net/2019/03/27/diferencias-clave-publica-clave-privada/>
- [2] <https://blog.mdcloud.es/para-que-sirve-la-criptografia-de-clave-publica-y-privada/>
- [3] Sobre `systemctl` y el inicio o la habilitación de servicios: <https://www.digitalocean.com/community/tutorials/how-to-use-systemctl-to-manage-systemd-services-and-units>.
- [4] <https://www.redhat.com/es/topics/linux/what-is-selinux>
- [5] <https://linux.die.net/man/8/semanage>
- [6] https://www.tutorialspoint.com/web_developers_guide/web_basic_concepts.htm
- [7] https://www.fail2ban.org/wiki/index.php/Main_Page
- [8] Página del proyecto rkhunter: <https://sourceforge.net/p/rkhunter/wiki/index/>

1. Introducción

Los conceptos previos necesarios en este caso son simplemente los gestores de paquetes (*apt* y *yum*, para Ubuntu y CentOS respectivamente), así como la utilidad básicas de *ssh*.

Tenemos que tener en cuenta los siguientes aspectos:

- Siguiendo las indicaciones del profesor, se ha comprobado qué comandos se pueden ejecutar sin *sudo*, y cuáles no. En este guión, ya viene el *sudo* puesto donde es necesario (previa comprobación manual).

2. Sesión 1ª: Instalación del servicio SSH en Ubuntu

El proceso que describimos a continuación en esta sección se corresponde con la instalación y configuración básica de SSH en Ubuntu Server. En la siguiente sección (ver) veremos el proceso análogo pero para nuestro servidor CentOS.

2.1. Instalación y pruebas del servicio

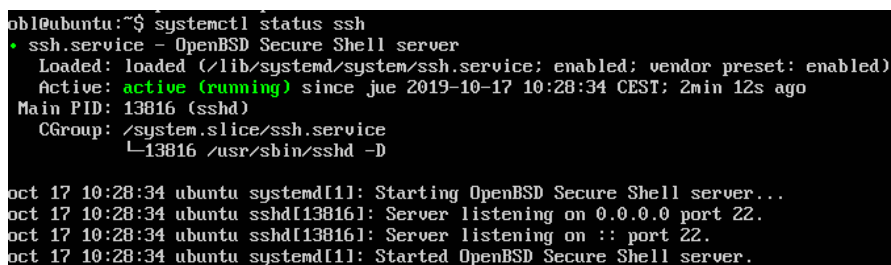
1. Lo primero que hay que hacer es instalar el servicio *ssh* mediante el gestor de paquetes. Para ello, vamos a utilizar las opciones de dicho gestor para hacer una búsqueda correcta de lo que queremos instalar:

```
apt search ssh | grep server      # busco el servicio SSH en apt
sudo apt install openssh-server  # instalar el paquete openssh-server
```

Hecho esto, vamos a ver si está activo el **servicio SSH**. Recordemos que pueden aparecer dos «versiones» o *comandos ssh*: el cliente (no lleva la letra «d» tras el literal «ssh») y el servidor o *daemon*, que sí la lleva: «**sshd**». Para comprobar que en efecto son *procesos* del Sistema Operativo como cualquier otro, usamos una orden conocida

```
systemctl status ssh
```

y el resultado es el de la figura 1.



```
obl@ubuntu:~$ systemctl status ssh
• ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since jue 2019-10-17 10:28:34 CEST; 2min 12s ago
   Main PID: 13816 (sshd)
   CGroup: /system.slice/ssh.service
           └─13816 /usr/sbin/sshd -D

oct 17 10:28:34 ubuntu systemd[1]: Starting OpenBSD Secure Shell server...
oct 17 10:28:34 ubuntu sshd[13816]: Server listening on 0.0.0.0 port 22.
oct 17 10:28:34 ubuntu sshd[13816]: Server listening on :: port 22.
oct 17 10:28:34 ubuntu systemd[1]: Started OpenBSD Secure Shell server.
```

Figura 1: Comprobando el estado de *sshd* tras la instalación

2. Vamos a probar ahora SSH. Para ello, hacemos en modo usuario (no *root*) una conexión al **localhost**, es decir, a la propia máquina en sí misma (ver figura 2):

Y como vemos en la figura 2 la conexión se ha realizado satisfactoriamente. Si lo hiciéramos en modo *root*, daría un **error**, pues nos estamos intentando conectar desde *root* hacia el usuario «remoto» *root*, y el servicio, tal y como está configurado, **no nos permite conectarnos por clave al root del usuario del servidor al que nos conectamos** (en este caso el servidor era localhost, claro).

El servicio, con la opción concreta con la que viene configurado por defecto, tiene un **agujero de seguridad** que permite acceder al usuario *root* remoto de manera ilegítima. Estos y otros parámetros interesantes los vamos a trabajar sobre el fichero fundamental del servicio SSH: */etc/ssh/sshd_config*.

```
obl@ubuntu:~$ ssh localhost
obl@localhost's password:
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.4.0-131-generic x86_64)

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

Pueden actualizarse 103 paquetes.
102 actualizaciones son de seguridad.

New release '18.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Thu Oct 17 10:34:25 2019 from ::1
obl@ubuntu:~$
```

Figura 2: Probando SSH.

2.2. Configuración del servidor SSH mediante el fichero /etc/ssh/sshd_config

Lo primero que hay que tener en cuenta es que para que los cambios realizados sobre el servidor SSH tengan efecto tenemos que tener cuidado de acceder siempre al fichero del servicio **servidor**, es decir, el asociado a sshd: /etc/ssh/sshd_config) y no al del cliente, que se llama igual salvo la «d» final del nombre del fichero¹, que no la tiene (al no ser el proveedor del servicio, es decir, el servidor SSH).

Dicho esto, abriremos el fichero anterior con vi y ajustaremos algunos parámetros interesantes. Para simplificar la realización y comprensión de los pasos realizados, proporcionamos en la figura 3 una captura de la «zona» del fichero referido que nos interesa. Sólo ajustaremos parámetros de los que aparecen en esta zona, por ser los más relevantes para nosotros.

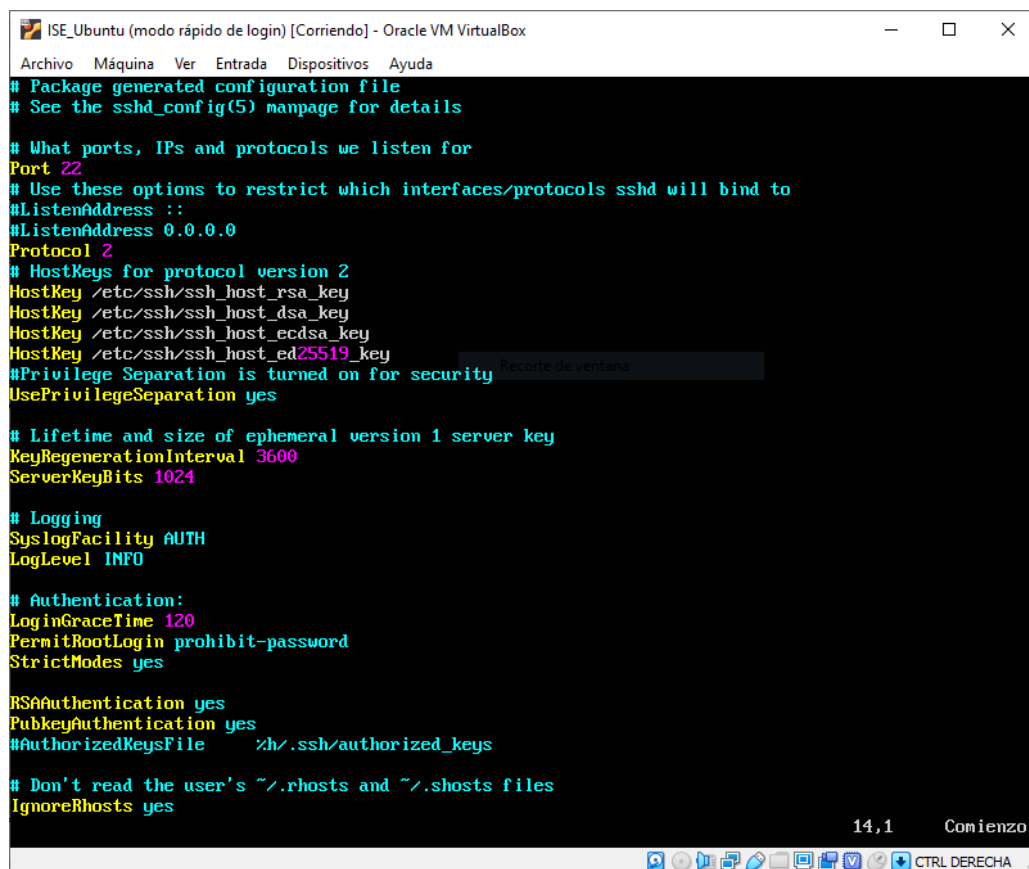


Figura 3: Fragmento de interés del fichero /etc/ssh/sshd_config).

Vamos ya con las modificaciones sobre el fichero /etc/ssh/sshd_config):

¹la letra «d» de sshd se refiere al servidor SSH, y no al cliente

1. En primer lugar configuraremos el parámetro `PermitRootLogin`, y cambiaremos su valor a «**no**», para **solucionar el agujero de seguridad** que había al poder accederse al `root` mediante clave. Tras guardar los cambios, salimos del editor `vi` y reiniciamos el servicio con la orden:

```
systemctl restart sshd      # reiniciar el servicio sshd
```

2. A continuación, vamos a modificar el puerto por defecto (22) y lo pondremos, por ejemplo, que sea el **22022**. Esto lo haremos en la línea (al principio del fichero) `Port`, y cambiaremos «22» por «22022». Guardamos y reiniciamos el servicio con el comando anterior.

Ahora vamos a hacer algunas pruebas:

1. Conectémonos (estando en modo `root`) al servidor (`localhost`). No nos debe de dejar, en función de lo que hemos escrito en el fichero (ver figura 4).

```
root@ubuntu:/home/obl# ssh localhost -p 22022
root@localhost's password:
Permission denied, please try again.
root@localhost's password:
Permission denied, please try again.
root@localhost's password:
Permission denied (publickey,password).
root@ubuntu:/home/obl# _
```

Figura 4: Conexión a `root` (notar que como nos estamos conectando desde el `root`, se intentará conectar también al `root` de la máquina remota).

Y en efecto no nos deja. Salgámonos del modo `root` en nuestra máquina para poder conectarnos correctamente (al usuario «normal», y no al `root`) a partir de ahora.

2. Veamos si nos deja acceder ahora por el puerto 22 con el parámetro `-p` (ver figura 5). Y tras ver el resultado, confirmamos que no nos deja.

```
obl@ubuntu:~$ ssh localhost
ssh: connect to host localhost port 22: Connection refused
obl@ubuntu:~$
```

Figura 5: Conectándonos al puerto por defecto.

3. Ahora vamos a conectarnos con el puerto correcto: «22022» (ver figura 6).

```
obl@ubuntu:~$ ssh localhost -p 22022
obl@localhost's password:
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.4.0-131-generic x86_64)

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

Pueden actualizarse 103 paquetes.
102 actualizaciones son de seguridad.

New release '16.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Thu Oct 17 11:41:56 2019 from ::1
obl@ubuntu:~$ _
```

Figura 6: Conectándonos usando el puerto correcto.

2.3. El cifrado por clave público-privada

2.3.1. Contexto

Tanto la clave pública como la clave privada tienen una finalidad. Un uso concreto para que podamos cifrar y descifrar el contenido, así como los contactos a los que enviamos un determinado archivo.

Podemos decir someramente que una clave pública tiene como única función cifrar un archivo o documento. Por su parte, una clave privada puede tanto cifrar como descifrar.

Si enviamos un documento cifrado a otro contacto, lo cifraremos con la clave pública de dicho contacto. Sin embargo, esa persona tiene que utilizar la clave privada para poder descifrar lo que le hemos enviado.

Al cifrado de clave pública también se le conoce como **cifrado asimétrico**. Es algo que se distribuye de manera pública a ambas partes. La parte «**receptora**» es la que cuenta con una **clave privada** (y evidentemente, también tiene la pública) para poder **descifrar** ese archivo o documento. **La clave privada va asociada a la pública, tal que el usuario receptor es poseedor del par de claves, pública y privada, que son inseparables.**

Hay que mencionar que la clave privada no se comparte, como sí ocurre con la pública. En este caso también se le conoce como cifrado simétrico. Con la clave privada podemos tanto cifrar como descifrar esa información.

Tenga en cuenta:

- Es el destinatario (A) de la información el que tiene que crear su clave público-privada (par de claves), y **enviar la pública al emisor (B) del mensaje para que cifre el mensaje y se lo mande al usuario A.**
- **La clave pública puede ser vista por todos, ya que no descifra, sólo cifra.**
- La clave privada sólo puede tenerla el **destinatario**, el receptor de la información, nadie más.

Más información y aclaraciones sobre el concepto de criptografía, cifrado simétrico, asimétrico y demás conceptos en la referencia (2).

2.3.2. Gestión en Linux

Vamos a cambiar de Sistema Operativo. Nos iremos a CentOS, que actuará como cliente (receptor de información por parte del servidor) y el servidor será Ubuntu. Entonces CentOS, al ser el cliente, será el que *Cree la clave público-privada.* Esto lo hará con el comando siguiente (ver figura 7).

```
ssh-keygen # genera dos archivos, uno con cada clave
```

Y se habrán creado dos archivos: la clave **privada** (fichero llamado **clave**) y la **pública** (fichero **clave.pub**).

Ahora habrá que mandarle al «emisor» la clave pública generada, para que cifre la información que quiere mandar y nos la mande cifrada. Nosotros descifraremos la información con nuestra clave privada.

Para mandarla usaremos el comando:

```
ssh-copy-id IP -p PUERTO # enviar clave publica
```

3. Sesión 2ª (parte I): Instalación de SSH en CentOS

En este caso vamos a instalar el mismo servicio que antes (SSH) pero en nuestro servidor CentOS, para que pueda que pueda recibir peticiones de conexión remota (para que podamos «conectarnos» remotamente a él).

```

[root@localhost abh]# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa): clave
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in clave.
Your public key has been saved in clave.pub.
The key fingerprint is:
ae:15:a7:c3:16:3c:1e:35:25:4d:07:b5:91:06:67:ff root@localhost.localdomain
The key's randomart image is:
+--[ RSA 2048 ]-----+
|      .O=+.      |
|      o.+o+      |
|      o  ..      |
|      . . .      |
|      S .  E!     |
|      + B         |
|      0           |
|      + .         |
|      .           |
+-----+
[root@localhost abh]#
[root@localhost abh]#
[root@localhost abh]#
[root@localhost abh]# ls -lha .
total 24K
drwx----- 2 abh abh 113 oct 22 22:42 .
drwxr-xr-x 3 root root 17 oct 8 20:16 ..
-rw----- 1 abh abh 43 oct 9 01:41 .bash_history
-rw-r--r-- 1 abh abh 18 ago 2 2016 .bash_logout
-rw-r--r-- 1 abh abh 193 ago 2 2016 .bash_profile
-rw-r--r-- 1 abh abh 231 ago 2 2016 .bashrc
-rw----- 1 root root 1,8K oct 22 22:42 clave
-rw-r--r-- 1 root root 408 oct 22 22:42 clave.pub
[root@localhost abh]#

```

Figura 7: Generando la clave público-privada.

3.1. Instalación y puesta en marcha

Lo primero que haremos para configurar el servicio, tras arrancar la máquina virtual, es entrar al modo *root*.

A continuación comprobamos si el servicio está habilitado (e instalado) ahora usando el comando (ver figura 8):

```
systemctl status sshd
```

```

[abh@localhost ~]$ systemctl status sshd
■ sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
   Active: active (running) since sáb 2019-10-26 13:21:11 CEST; 29min ago
     Docs: man:sshd(8)
           man:sshd_config(5)

```

Figura 8: Comprobando el estado previo del demonio/servicio SSH

Bien, ahora que sabemos que el servicio está activo², vamos a hacer una conexión SSH a *localhost*, desde *root*, y veremos si, por defecto, nos permite acceder al *root* de la máquina remota (*localhost*) mediante contraseña (ver figura 9):

```

[root@localhost abh]# ssh localhost
The authenticity of host 'localhost (::1)' can't be established.
ECDSA key fingerprint is b3:f1:ff:c8:61:b9:c9:cc:e7:9d:96:de:83:a7:ad:3c.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
root@localhost's password:
Last login: Sat Oct 26 15:09:53 2019
[root@localhost ~]# _

```

Figura 9: Haciendo un primer acceso a localhost (con el puerto por defecto)

Y vemos que se conecta al *root* del remoto (*localhost* en este caso) sin problema. Esto lo evitaremos,

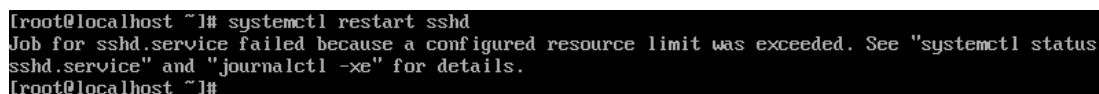
²Hemos descubierto que el demonio de SSH está instalado y activo por defecto en CentOS, sin haber configurado nosotros nada. Más adelante elaboraremos una tabla resumen (interesante para el examen) para resumir y clarificar las situaciones que nos hemos encontrado en Ubuntu frente a CentOS.

por motivos de seguridad, tal y como lo hacíamos en el apartado 2.1 de esta práctica: configurando la opción `PermitRootLogin` del fichero `/etc/ssh/sshd_config`.

Además podemos configurar el puerto también en este mismo fichero, para que no escuche en el puerto por defecto, sino que lo haga en otro diferente, como el **22022**. **Recuerda descomentar esas líneas del fichero para que se apliquen los cambios** ya que en CentOS aparecen *comentadas por defecto*. Ahora nos falta reiniciar el demonio/servicio SSH para que tengan efecto estos cambios, y eso se hace con el comando:

```
systemctl restart sshd
```

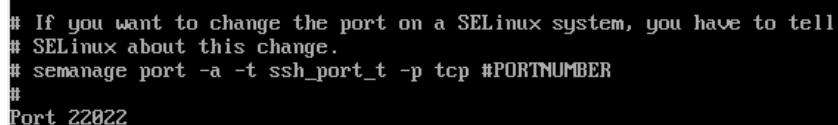
El resultado de reiniciar con esta orden se puede ver en la figura 10.



```
[root@localhost ~]# systemctl restart sshd
Job for sshd.service failed because a configured resource limit was exceeded. See "systemctl status
sshd.service" and "journalctl -xe" for details.
[root@localhost ~]#
```

Figura 10: Reiniciando el servicio SSH (`sshd`)

¿Qué ha ocurrido? Ya adelantamos que el error es por haber modificado el puerto (se puede comprobar cambiando primero `PermitRootLogin` y reiniciando, y no dará problemas, y luego intentar levantar la red, y dará el mismo fallo que ha dado ahora). Podemos entrar en los ficheros/órdenes que nos recomienda en el mensaje de error, pero vamos a ir directamente al fichero de configuración de `sshd` (`/etc/ssh/sshd_config`) en busca de respuestas, y nos encontramos, en la cabecera de dicho fichero la información siguiente (ver figura 11):



```
# If you want to change the port on a SELinux system, you have to tell
# SELinux about this change.
# semanage port -a -t ssh_port_t -p tcp #PORTNUMBER
#
Port 22022
```

Figura 11: Cabecera del fichero de configuración de SSH.

Nos dicen que si queremos **cambiar el puerto** en un sistema controlado por el «policía» de Linux (**SELinux**)³, hay que decirle a ese policía que lo vamos a cambiar, y eso se hace con la orden que nos dan, para la cual necesitamos tener instalado el programa `semanage`. Esta herramienta sirve para la gestión de servicios de puertos en Linux bajo la supervisión de SELinux.

Si buscamos con

```
yum provides semanage
```

el paquete que nos puede proporcionar la herramienta `semanage`, nos encontramos con lo siguiente (ver figura 12):

Entonces el paquete que tenemos que instalar es: `policycoreutils-python-2.5-33.el7.x86_64`. Lo instalamos con la orden de la figura 13.

Y una vez instalado correctamente, vamos a ejecutar la orden que nos indicaban en el fichero de configuración de `sshd` para poder cambiar correctamente el puerto. Esta orden estaba en la figura 11 y tras ejecutarla (sustituyendo `#PORTNUMBER` por `22022`) y que acabe podemos comprobar que el puerto deseado (22022) ahora está controlado por SELinux (ver figura 14):

Vamos a **reiniciar el servicio, para que todos los cambios tengan efecto**. Y después nos conectaremos a `localhost` por el puerto correcto (22022) y fuera del `root` (evidentemente) para ver si todo funciona bien (ver figura 15):

Se ha conectado, y podríamos decir que funciona bien, pues lo ha hecho además mediante contraseña hacia el usuario sin privilegios de `root`, que es el agujero de seguridad del que llevamos hablando a

³Security-Enhanced Linux (SELinux) es una arquitectura de seguridad para los sistemas Linux(R) que otorga a los administradores mayor control sobre las personas que pueden acceder al sistema. Originalmente, la Agencia de Seguridad Nacional (NSA) de Estados Unidos desarrolló este producto como una serie de parches para el kernel de Linux utilizando los módulos de seguridad de Linux (LSM). Más información en (4) y (5)


```
[root@localhost ~]# yum provides semanage
Complementos cargados:fastestmirror
Loading mirror speeds from cached hostfile
 * base: mirror.tedra.es
 * extras: mirror.tedra.es
 * updates: mirror.tedra.es
base/7/x86_64/filelists_db                | 7.3 MB  00:00:03
extras/7/x86_64/filelists_db             | 207 kB  00:00:00
updates/7/x86_64/filelists_db            | 2.1 MB  00:00:01
polycoreutils-python-2.5-33.el7.x86_64 : SELinux policy core python utilities
Repositorio                               : base
Resultado obtenido desde:
Nombre del archivo                        : /usr/sbin/semanage

[root@localhost ~]#
```

Figura 12: Buscando el paquete *semanage*.

```
[root@localhost ~]# yum install polycoreutils-python-2.5-33.el7.x86_64
```

Figura 13: Instalando el paquete *semanage*.

```
[root@localhost ~]# semanage port -l | grep ssh
ssh_port_t                tcp      22022, 22
```

Figura 14: Comprobando que el puerto 22022 aparece en la lista de puertos controlados por SELinux. Nota: en la imagen también aparece el puerto 22, porque es el puerto por defecto y por tanto ya estaba controlado por SELinux por defecto.

```
[root@localhost abh]# systemctl restart sshd
[root@localhost abh]# exit
exit
[abh@localhost ~]# # ahora estoy fuera del root, puedo hacer el ssh a localhost
[abh@localhost ~]#
[abh@localhost ~]# ssh localhost -p 22022
abh@localhost's password:
Last login: Mon Oct 28 01:25:24 2019 from localhost
[abh@localhost ~]#
```

Figura 15: Reiniciando el servicio, saliendo de modo root y conectándonos a *localhost* por el puerto 22022.

```
obl@ubuntu:~$ ssh 192.168.56.110 -p 22022
ssh: connect to host 192.168.56.110 port 22022: No route to host
obl@ubuntu:~$ _
```

Figura 16: Comprobando la configuración realizada.

```
[root@localhost abh]# firewall-cmd --add-port=22022/tcp
success
[root@localhost abh]# _
```

Figura 17: Habilitando el puerto 22022 para el cortafuegos de centOS.

lo largo de la práctica; pero, ¿realmente lo hemos hecho bien? Hagamos una prueba definitiva: arranquemos la MV Ubuntu, que hará de cliente y conectémonos a nuestro servidor (que será la máquina centOS en la que estamos ahora) aprovechando que están conectadas en red (lo hicimos en la práctica 1). Comprueba la conexión entre las máquinas haciendo un ping desde una a otra. Al conectarnos desde Ubuntu a centOS nos aparece el resultado de la figura 16.

Tras observar la figura 16 vemos que no se ha podido conectar. Vamos a intentar solucionar el problema revisando el estado del cortafuegos, que es algo que aún no hemos contemplado.

3.2. Configuración del cortafuegos (en centOS)

Una de las posibles causas que se nos podrían ocurrir al problema sería el cortafuegos en nuestro servidor **centOS**, del cual aún no hemos hablado. Veamos el estado del mismo en **nuestro servidor centOS**, con el comando:

```
firewall-cmd --state # consultar 'man firewall-cmd' para mas informacion
```

y nos indicará que está activo. ¿Qué falta, entonces? Sencillo, añadir el puerto 22022 para que el cortafuegos de centOS no lo bloquee, y permita tráfico por ahí. Esto lo haremos con la opción `--add-port` de la misma herramienta (`firewall-cmd`) e indicándole el protocolo de comunicación y el número de puerto de la forma que se indica en la figura 17.

Volvamos a probar a intentar conectarnos desde Ubuntu (ver figura 18):

Y vemos que **se conecta satisfactoriamente**. Aún hay un pequeño problema, y es que necesitamos que esa adición del puerto 22022 a la lista del *firewall* de centOS sea **definitiva**, para evitar tener que ejecutar el comando cada vez que arranquemos la máquina. Haremos lo siguiente: (1) vamos a reiniciar nuestra máquina centOS, (2) cuando centOS esté arrancado y hayamos entrado, nos conectamos desde Ubuntu y no nos dejará. Ahora ejecutaremos (en centOS, evidentemente) el mismo comando que habíamos usado para añadir el puerto 22022, pero indicándole que sea un cambio «permanente», y para ello usaremos la opción `--permanent` al final del comando referido. El comando y su resultado se pueden ver en la figura 19.

Y como muy bien dicen los comentarios de la figura 19, si volvemos a conectarnos ahora desde Ubuntu server, vemos que nos dejará. A continuación mostramos en el par de figuras 20-21 un cambio+prueba en tiempo real (ver la fecha en el *prompt* de cada captura).

El lector más elocuente se habrá percatado de algo «incoherente» en nuestra explicación: ¿por qué hemos necesitado el cortafuegos en centOS y no nos hizo falta cuando configuramos el mismo servicio en Ubuntu? La razón no es evidente, se trata de una de las características que diferencia a Ubuntu

```
obl@ubuntu:~$ ssh 192.168.56.110 -p 22022 -l abh
The authenticity of host '192.168.56.110:22022 ([192.168.56.110]:22022)' can't be established.
ECDSA key fingerprint is SHA256:BNkSgQsxsyY22q0CJnJDEBTH3jW1K4bisEWM17T2dw.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.56.110:22022' (ECDSA) to the list of known hosts.
abh@192.168.56.110's password:
Last login: Mon Oct 28 02:02:38 2019
labh@localhost ~1$
```

Figura 18: Reintentando la conexión desde Ubuntu tras habilitar el puerto 22022 en el cortafuegos de centOS.

```

[root@localhost abh]# firewall-cmd --add-port=22022/tcp --permanent
success
[root@localhost abh]# # para que se active desde ya, ejecutamos el comando inicial:
[root@localhost abh]# firewall-cmd --add-port=22022/tcp
success
[root@localhost abh]# #volvemos a conectarnos desde ubuntu, y nos dejará SIEMPRE
[root@localhost abh]#

```

Figura 19: Habilitando de forma permanente el puerto 22022 para el *firewall*. Nota: se han añadido algunos comentarios en la propia imagen para aclarar lo que se está haciendo

```

root 02:31:45 @localhost abh
$ firewall-cmd --add-port=22022/tcp
success
root 02:31:54 @localhost abh
$_

```

Figura 20: Añadiendo el puerto antes de conectarse desde Ubuntu.

Server de CentOS: en CentOS el cortafuegos viene activo por defecto (es decir, controlando el tráfico donde debe) y por tanto hay que «decirle» que «abra el tráfico» por el puerto 22022; por el contrario, Ubuntu Server no tiene su cortafuegos activo, tal que es como si no estuviera trabajando.

Si hacemos ahora una prueba desde CentOS para **intentar conectarnos a Ubuntu Server a través de SSH**, veremos que nos dejará, pero eso es un potencial agujero de seguridad, por lo que vamos, en primer lugar, a **ver el estado del cortafuegos del servidor Ubuntu** con el comando:

```
ufw status          # ver estado del cortafuegos de Ubuntu
```

Y como vemos que en efecto, tal como hemos dicho, está inactivo («no trabaja»), vamos a activarlo:

```
ufw enable          # activar cortafuegos
```

Si volvemos a intentar conectarnos (no sirve hacerlo desde localhost) desde CentOS o desde la máquina nativa a través del puerto que usamos para SSH (**22022**), veremos que no nos deja, porque no hemos añadido este puerto a la lista del cortafuegos. La filosofía es la misma que en CentOS, tal que basta añadirlo a la lista de **ufw** con el comando:

```
ufw allow 22022      # permitir el trafico por el puerto 22022
```

y se añadirá el puerto **de forma permanente** (otra diferencia con respecto a CentOS).

3.3. RECAPITULACIÓN: Comparativa de servicios frecuentes en Ubuntu Server *versus* CentOS 7

La comparativa pretende arrojar luz sobre los servicios que hemos estado manejando a lo largo de la práctica 2 hasta ahora, pero también de los usados durante la práctica 1. **Fue redactada y completada EN CLASE, junto al profesor, y es muy útil, casi obligatoria, para el supuesto examen que se vaya a realizar.** La comparativa se puede ver en la tabla 1.

4. Sesión 2ª (parte II): Copias de seguridad y control de versiones

Toda la información necesaria relativa a este apartado se puede encontrar en el material oficial de la asignatura (se puede consultar también este material en <https://github.com/alonso-bh/>

```

obl 02:32:48 @ubuntu ~
$ ssh 192.168.56.110 -p 22022 -l abh
abh@192.168.56.110's password:
Last login: Mon Oct 28 02:18:54 2019
[abh@localhost ~]$ _

```

Figura 21: Conectándonos a CentOS justo después de ejecutar la orden la figura 20.

	¿SSH activo por defecto?	Nomenclatura para SSH	Archivo de configuración...	Acceso root por defecto permitido	Firewall activo por defecto
Ubuntu Server	No	ssh y sshd (para el servicio)	No comentado	No, si es por contraseña.	No.
centOS 7	Sí	sshd	Comentado	Sí	Sí

Cuadro 1: Tabla comparativa de servicios en centOS 7 y Ubuntu Server.

Ingenier-a-de-Servidores-UGR/blob/master/Material%20de%20pr%C3%A1cticas/Guiones%20de%20pr%C3%A1cticas/ISE-P2-Lec2.pdf).

5. Sesión 3ª (parte I): Servidores Web. Instalación y configuración de HTTP en Ubuntu y centOS.

5.1. Conceptos previos

Antes de adentrarnos en la ejecución de la práctica conviene tener claros algunos conceptos básicos relacionados con la Web:

1. Página web
2. Documento web/documento HTML
3. Protocolo HTML
4. URL y URI
5. WWW
6. Lenguajes compilados e interpretados
7. Concepto de dinamismo de la web
8. Servidor web

Puede repasar estos conceptos en la referencia (6).

5.2. Instalación del servidor web en Ubuntu Server

El caso de Ubuntu es el más simple de los dos. Podemos usar el software de la **pila LAMP (Linux + Apache + MySQL [MariaDB] + PHP)**, un conjunto de recursos para el andamiaje de servidores web y desarrollo.

Una vez instalado ese software, que se hace de forma guiada y muy simple con Ubuntu, habrá que hacer algunos ajustes mínimos y pruebas.

Entramos en una terminal de Ubuntu Server, en **modo root**, y ejecutamos:

```
tasksel
```

para iniciar el asistente de instalación de la pila LAMP.

Instalamos la pila software y cuando terminemos haremos algunas comprobaciones:

1. Comprobar que el servicio Apache está activo con la orden:

```
systemctl status apache2
```

```

[root@localhost abh1# systemctl status httpd
■ httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset: disabled)
   Active: inactive (dead)
     Docs: man:httpd(8)
           man:apachectl(8)
[root@localhost abh1# system

```

Figura 22: Consultando el estado del servicio httpd.

Y probamos que el servidor está **accesible** (funcionando, a todos los efectos) desde nuestro navegador insertando en la barra de búsqueda la URL: 192.168.56.105, o ejecutando desde la propia terminal del servidor (Ubuntu) el comando:

```
curl localhost
```

con el cual usamos el navegador en línea Curl para conectarse a sí mismo (**localhost**, equipo local), es decir, al propio servidor.

2. Probar PHP: simplemente intentaremos, desde el terminal de Ubuntu, abrir el intérprete de PHP, que es el lenguaje estrella (por ahora) para la programación de aplicaciones en el servidor. Ejecutamos pues la orden:

```
php -a    # abrir el intérprete
```

3. Probar la base de datos MariaDB [MySQL]:

```
mysql -u root -p
```

y se meterá en el intérprete de MariaDB si todo ha ido bien. Los manuales de MariaDB estás disponibles para cualquier duda. Podemos probar que las BD funciona correctamente ejecutando alguna consulta básica en SQL en el intérprete.

Completados estos pasos, ya tendríamos listo nuestro servidor web, sólo faltaría añadirle el contenedor, aplicaciones web que se quieran diseñar...

5.3. Instalación del servidor web en CentOS Server.

En CentOS no existe un software preparado para la pila LAMP. Tenemos que instalar manualmente, y uno por uno, cada uno de sus componentes necesarios: el servidor web Apache, la base de datos MariaDB Y el intérprete PHP.

Los **pasos** son los siguientes:

1. **Instalar Apache en CentOS.** Usaremos la orden

```
yum search apache | grep server
```

y vemos en el resultado de esta orden que el nombre del software de Apache para CentOS se llama **httpd**, luego vamos a intentar instalar ese paquete software con la orden

```
yum install httpd
```

Una vez que se ha instalado, y así se nos indica en la terminal, vamos a ver el **estado del servicio** con una orden conocida (ver figura 23).

Y como vemos que está inactivo, podemos iniciarlo y habilitarlo con unas órdenes conocidas⁴:

```

systemctl start httpd      # iniciar el servicio
systemctl enable httpd     # habilitarlo siempre al arranque del servidor (boot)

```

⁴cfr. (3)

```

[root@localhost abh1# systemctl status httpd
■ httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset: disabled)
   Active: active (running) since mar 2020-06-30 23:50:51 CEST; 42s ago
     Docs: man:httpd(8)
           man:apachectl(8)
  Main PID: 2363 (httpd)
   Status: "Total requests: 0; Current requests/sec: 0; Current traffic:  0 B/sec"
    CGroup: /system.slice/httpd.service
            └─2363 /usr/sbin/httpd -DFOREGROUND
              └─2364 /usr/sbin/httpd -DFOREGROUND
                └─2365 /usr/sbin/httpd -DFOREGROUND
                  └─2366 /usr/sbin/httpd -DFOREGROUND
                    └─2367 /usr/sbin/httpd -DFOREGROUND
                      └─2368 /usr/sbin/httpd -DFOREGROUND

jun 30 23:50:51 localhost.localdomain systemd[1]: Starting The Apache HTTP Server...
jun 30 23:50:51 localhost.localdomain httpd[2363]: AH00558: httpd: Could not reliably determine...ge
jun 30 23:50:51 localhost.localdomain systemd[1]: Started The Apache HTTP Server.
Hint: Some lines were ellipsized, use -l to show in full.
[root@localhost abh1#

```

Figura 23: comprobación del estado del servicio httpd tras habilitarlo e iniciarlo.

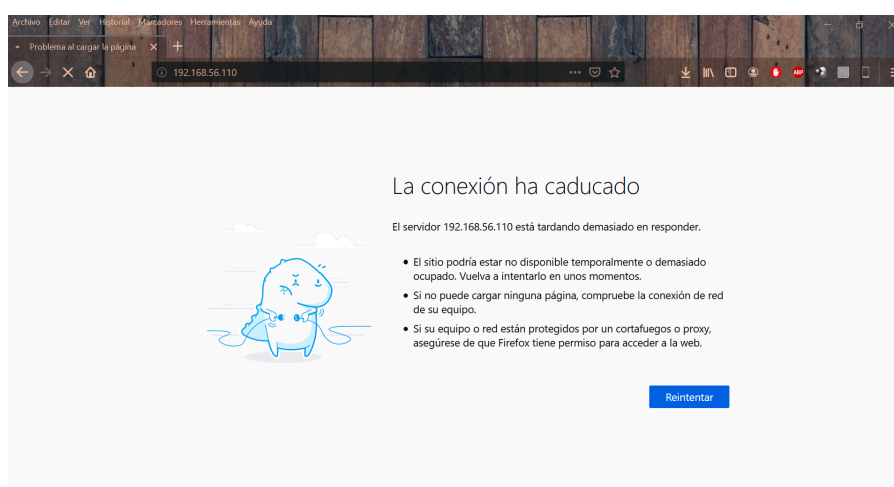


Figura 24: Comprobando el servidor web recién instalado (servicio httpd).

Volvemos a comprobar su estado: figura 23, y en ella ya vemos que el servicio está activo, es decir, nuestro servidor ya puede, en principio servir páginas HTML y responder a peticiones sobre este protocolo. Vamos a comprobarlo: iniciamos el navegador de nuestro ordenador (anfitrión) y en la barra de búsqueda introducimos: 192.168.56.110/. El resultado se puede ver en la figura 24.

Y se muestra un error, no hemos podido acceder al servidor. Revisemos el estado del cortafuegos de CentOS, y añadamos el puerto 80 para que permita el tráfico por ese puerto, que es el puerto por defecto de HTTP (servidor web). Esto lo hacemos como se indica en la figura 25.

Vamos a intentarlo de nuevo en el navegador. Tras recargar la pagina (F5) nos aparece lo de la figura 26.

Y tal y como se demuestra en la figura 26, tenemos el servicio de páginas HTML listo. Pero para tener toda la infraestructura del servidor web montada (pila LAMP) aún nos falta el intérprete, el que aporta el dinamismo a las páginas web: PHP, y también el Sistema Gestor de Bases de Datos: MariaDB.

```

[root@localhost abh1# firewall-cmd --add-port=80/tcp --permanent
success
[root@localhost abh1#

```

Figura 25: Añadiendo al control el puerto 80 por el cortafuegos de CentOS.

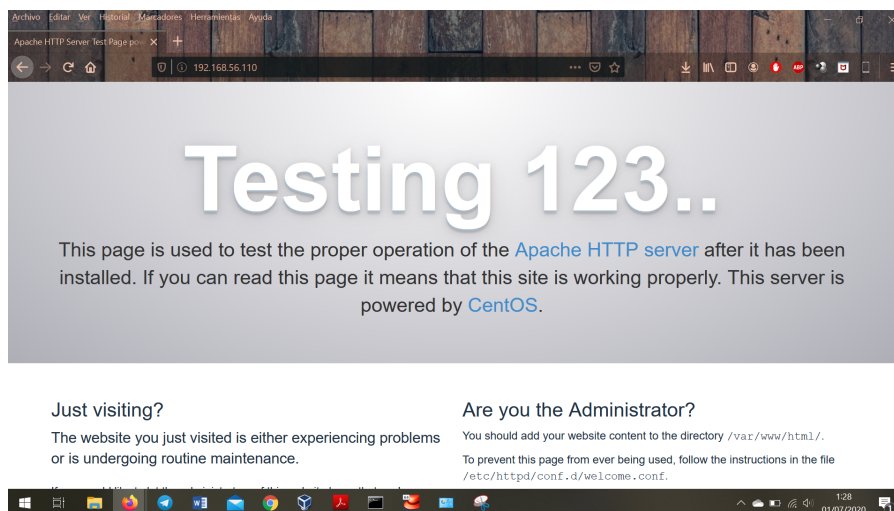


Figura 26: Acceso satisfactorio al servidor web de Apache.

2. Instalación del intérprete PHP. Ejecutamos:

```
yum install php -y
```

Si todo ha ido bien, nos saldrá lo siguiente (ver figura 27).

Vamos a probar que el intérprete se ha instalado correctamente: abrimos el intérprete, para lo cual (si miramos el manual) usaremos el comando:

```
php -a
```

se abre el intérprete de PHP y podemos ejecutar la siguiente sentencia PHP:

```
echo "Hola_a_todos\n";
```

y al pulsar Intro se imprimirá en la siguiente línea el texto “Hola a todos”.

6. Sesión 3ª (parte II): Seguridad y análisis en servidores: fail2ban y rkhunter

6.1. Introducción a fail2ban

Esta herramienta es un «baneador» de IP que son sospechosas de acciones maliciosas contra el sistema donde se instala. Permite, entre otras cosas, definir **cárceles** para estas IP's, bloqueándolas e impidiendo su acceso al sistema.

Una descripción más detallada se da en el sitio oficial (ver (7)):

Fail2ban scans log files (e.g. /var/log/apache/error_log) and bans IPs that show the malicious signs -too many password failures, seeking for exploits, etc. Generally Fail2Ban is then used to update firewall rules to reject the IP addresses for a specified amount of time, although any arbitrary other action (e.g. sending an email) could also be configured. Out of the box Fail2Ban comes with filters for various services (apache, courier, ssh, etc).

Fail2Ban is able to reduce the rate of incorrect authentication attempts however it cannot eliminate the risk that weak authentication presents. Configure services to use only two factor or public/private authentication mechanisms if you really want to protect services.

```

ise_centOS [Corriendo] - Oracle VM VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
(4/6): php-5.4.16-48.el7.x86_64.rpm                | 1.4 MB  00:00:00
(5/6): php-cli-5.4.16-48.el7.x86_64.rpm            | 2.7 MB  00:00:00
(6/6): php-common-5.4.16-48.el7.x86_64.rpm          | 565 kB  00:00:00
-----
Total                                              3.3 MB/s | 6.4 MB  00:00:01
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Actualizando   : 1:openssl-libs-1.0.2k-19.el7.x86_64      1/8
  Instalando     : libzip-0.18.1-8.el7.x86_64               2/8
  Instalando     : php-common-5.4.16-48.el7.x86_64          3/8
  Instalando     : php-cli-5.4.16-48.el7.x86_64            4/8
  Instalando     : php-5.4.16-48.el7.x86_64                5/8
  Actualizando   : 1:openssl-1.0.2k-19.el7.x86_64         6/8
  Limpieza      : 1:openssl-1.0.1e-60.el7.x86_64          7/8
  Limpieza      : 1:openssl-libs-1.0.1e-60.el7.x86_64     8/8
  Comprobando    : php-cli-5.4.16-48.el7.x86_64           1/8
  Comprobando    : 1:openssl-1.0.2k-19.el7.x86_64        2/8
  Comprobando    : libzip-0.18.1-8.el7.x86_64             3/8
  Comprobando    : php-common-5.4.16-48.el7.x86_64       4/8
  Comprobando    : php-5.4.16-48.el7.x86_64              5/8
  Comprobando    : 1:openssl-libs-1.0.2k-19.el7.x86_64   6/8
  Comprobando    : 1:openssl-libs-1.0.1e-60.el7.x86_64   7/8
  Comprobando    : 1:openssl-1.0.1e-60.el7.x86_64       8/8

Instalado:
  php.x86_64 0:5.4.16-48.el7

Dependencia(s) instalada(s):
  libzip.x86_64 0:0.18.1-8.el7  php-cli.x86_64 0:5.4.16-48.el7  php-common.x86_64 0:5.4.16-48.el7

Dependencia(s) actualizada(s):
  openssl.x86_64 1:1.0.2k-19.el7          openssl-libs.x86_64 1:1.0.2k-19.el7

¡Listo!
root@localhost abh1#

```

Figura 27: Instalando el intérprete de PHP.

6.1.1. Tareas básicas con fail2ban.

6.2. Introducción a rkhunter (*Root Kit Hunter*)

rkhunter, o *Root Kit Hunter*, es una herramienta que permite detectar elementos maliciosos en el sistema. Sus principales características se indican a continuación (extraídas de (8)):

Rootkit Hunter (commonly abbreviated as RKH) is a security monitoring and analyzing tool for POSIX compliant systems, to help you detect known rootkits, malware and signal general bad security practices. Rootkits have a certain structure and files in certain areas, known to the Rootkit Hunter team. This is similar to virus signatures. RKH offers additional scans that may assist you.

One of the features RKH offers is a scan for changed file properties similar to some criteria that file integrity checkers use. It is completely dependent on ensuring you have a correct database to scan from. In general this can be achieved by installing Rootkit Hunter right after a clean Operating System installation.

Para instalar y empezar a usar a nivel básico rkhunter se puede usar la referencia (8), aunque solamente nos interesan por ahora dos comandos muy sencillos:

1. Instalación en CentOS:

```
yum install rkhunter          # instalar rkhunter
```

2. Análisis (*profiling*) del sistema:

```
rkhunter -c
```