Anexo.

A.-Instalación de Ubuntu 9.04 Server.

El primer paso que se debe dar para tener un servidor web seguro es la instalación del sistema operativo, en nuestro caso será Ubuntu 9.04 Server. Este sistema operativo se puede descargar de la página oficial de Ubuntu o bien se puede pedir por correo postal en la misma página.

Una vez que se cuenta con un CD de instalación de Ubuntu 9.04 Server, para instalarlo hay que poner en la unidad de CD ROM el disco de instalación y reiniciar el equipo con el CD dentro. Si la BIOS del equipo se encuentra configurada para que el sistema arranque desde el CD, se accederá a la pantalla principal de la instalación, si no, se deberá configurar la BIOS para ello.

La pantalla principal que aparece al iniciar la instalación, es la siguiente:

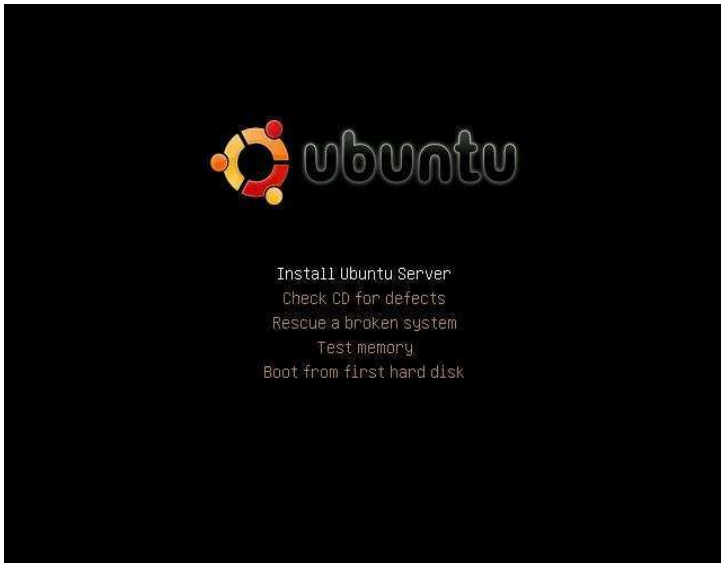


Figura A1.-Pantalla de instalación de Ubuntu 9.04 Server.

Aquí se seleccionará la opción *Install Ubuntu Server*, luego de esto aparecerán pantallas de configuración en donde se podrán elegir el idioma de instalación, el idioma del sistema operativo, el nombre de la máquina y la hora. Después de esto aparecerá un asistente que nos permitirá particionar el disco duro para la instalación del sistema operativo.

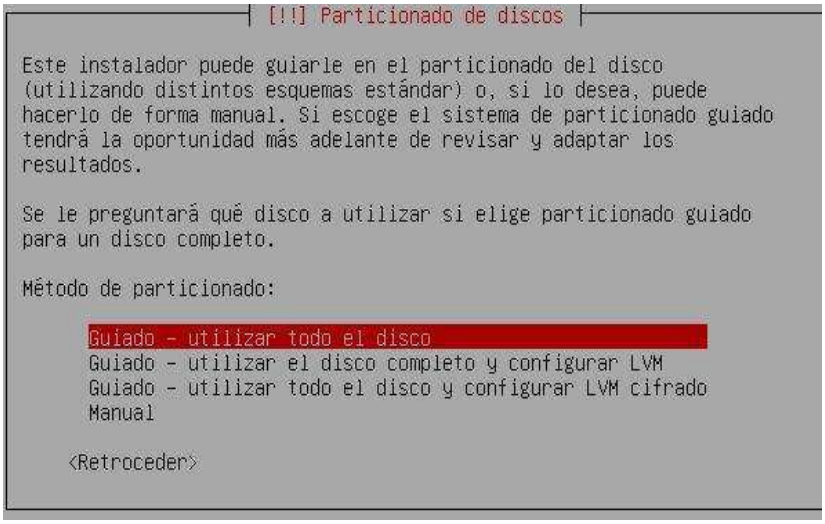


Figura A2.-Asistente de particionado de Ubuntu 9.04 Server.

En nuestro caso utilizaremos el particionado guiado. Al seleccionar esta opción, el asistente de instalación nos pedirá seleccionar el disco duro en el que deseamos instalar Ubuntu 9.04 Server, se seleccionará el disco duro y el sistema propondrá una estructura de particionado, la cual nosotros aceptaremos.

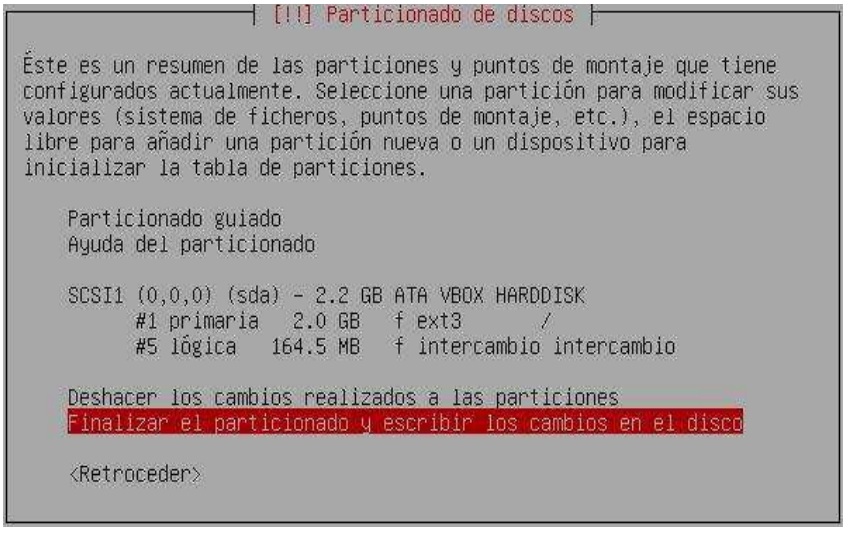


Figura A3.-Creación de particiones para Ubuntu 9.04 Server.

Después de seleccionar la opción de finalizar el particionado, aparecerá una ventana de confirmación, en la que se seleccionará aceptar. Posterior a esto, el disco será formateado y se iniciará la instalación de Ubuntu 9.04 Server.

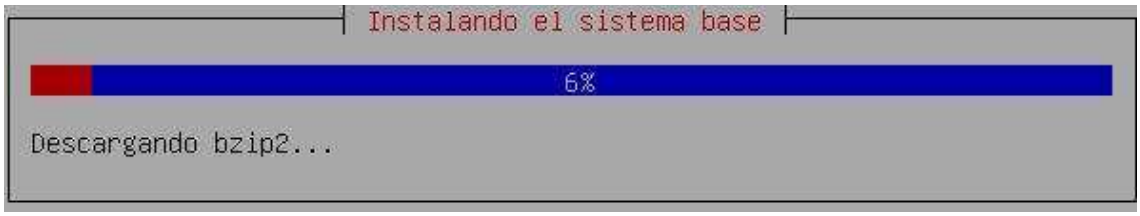


Figura A4.-Proceso de instalación de Ubuntu 9.04 Server.

Al terminar el proceso de instalación, se nos pedirá crear una cuenta de usuario, en donde se requerirá escribir el nombre completo del usuario, el nombre de usuario para la cuenta y la contraseña de la cuenta en dos ocasiones. Finalmente aparecerá una ventana en la que se nos permitirá seleccionar algunos paquetes de software a instalar. En dicha ventana no se seleccionará paquete alguno y se elegirá la opción continuar. El asistente de instalación dará aviso de que la instalación ha terminado y el equipo será reiniciado. Con esto habrá terminado la instalación de Ubuntu 9.04 Server.

Figura A5.-Selección de software de Ubuntu 9.04 Server.

B.-Instalación de OpenSSH utilizando aptitude.

Para poder administrar de manera remota nuestro servidor web o nuestras cuentas, es necesario hacer uso de algún protocolo que nos permita hacerlo (Telnet, RSH, FTP), en nuestro caso hemos optado por utilizar el protocolo SSH que es una buena alternativa.

Como nuestro sistema operativo no cuenta con este servicio, se deberá instalar, para ello utilizaremos un paquete de código abierto llamado OpenSSH, que cuenta con un cliente y con un servidor SSH. Esto lo haremos con la ayuda del gestor de paquetes de Ubuntu *aptitude*.

Primero se deberá iniciar sesión y luego de ello ejecutar una serie de comandos como superusuario.

Para instalar el cliente SSH, deberemos teclear el siguiente comando:

```
sudo aptitude install openssh-client
```

Con esto el paquete será descargado e instalado en el sistema.

Lo siguiente será instalar el servidor, como sigue:

```
sudo aptitude install openssh-server
```

Y por último se reiniciará el servicio:

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

Para verificar que el servicio SSH funciona correctamente, nos conectaremos de manera remota al servidor.

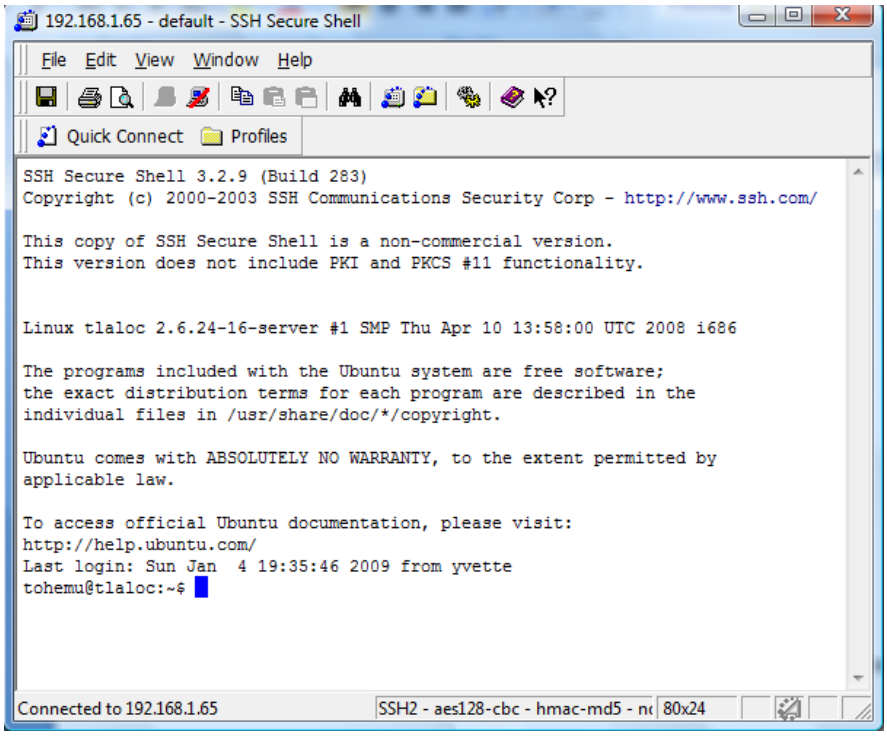


Figura A6.-Conexión remota al servidor por SSH.

C.-Instalación de Apache utilizando aptitude.

Para instalar Apache, deberemos descargarlo por medio de la herramienta de gestión de paquetes con la que cuenta nuestra distribución de Linux (Ubuntu 9.04). Basta con abrir una terminal y teclear el comando que se muestra (como superusuario, utilizando el comando sudo):

```
sudo aptitude install apache2
```

Cuando haya finalizado la descarga e instalación de Apache, se deberá reiniciar el servicio para que nuestro servidor web entre en funcionamiento:

```
sudo /etc/init.d/apache2 restart
```

Una vez terminada la instalación, el sistema nos dará aviso y para comprobar que nuestro servidor web funciona correctamente hay que abrir un navegador y teclear la IP de nuestro servidor. Si todo se hizo correctamente aparecerá en el navegador una pantalla como la que sigue:

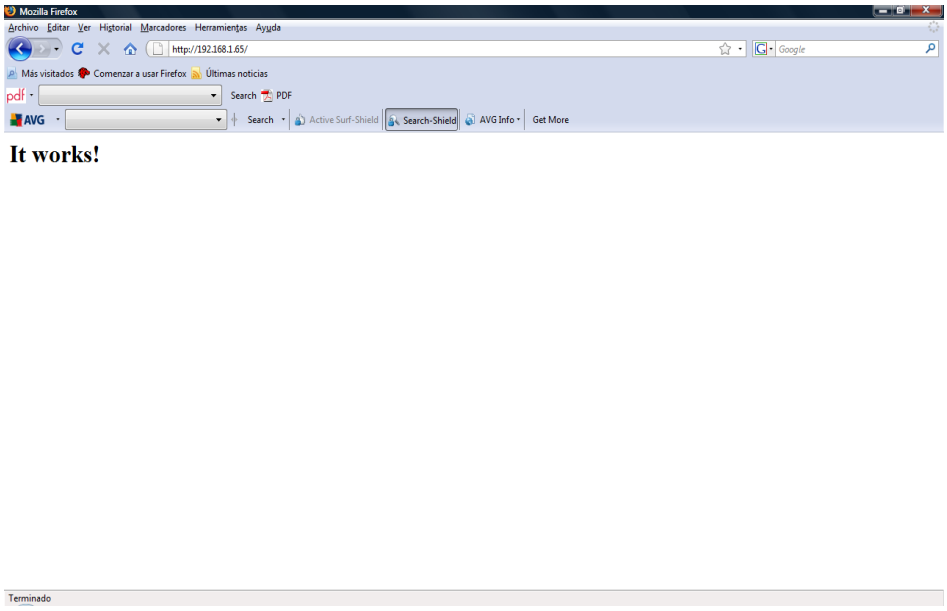


Figura A7.-Comprobación del Servidor web Apache.

D.-Instalación de MySQL utilizando aptitude.

En esta sección instalaremos el sistema manejador de bases de datos MySQL en nuestro servidor, para ello haremos uso del gestor de paquetes de Ubuntu 9.04 Server. Como en

el caso de OpenSSH, aquí también requerimos de un cliente y un servidor. El procedimiento es como sigue:

Primero descargaremos e instalaremos el cliente:

```
sudo aptitude install mysql-client
```

Seguido de ello instalaremos el servidor:

```
sudo aptitude install mysql-server
```

Cuando la instalación haya terminado, el sistema solicitará que se ingrese la contraseña de *root* en dos ocasiones.

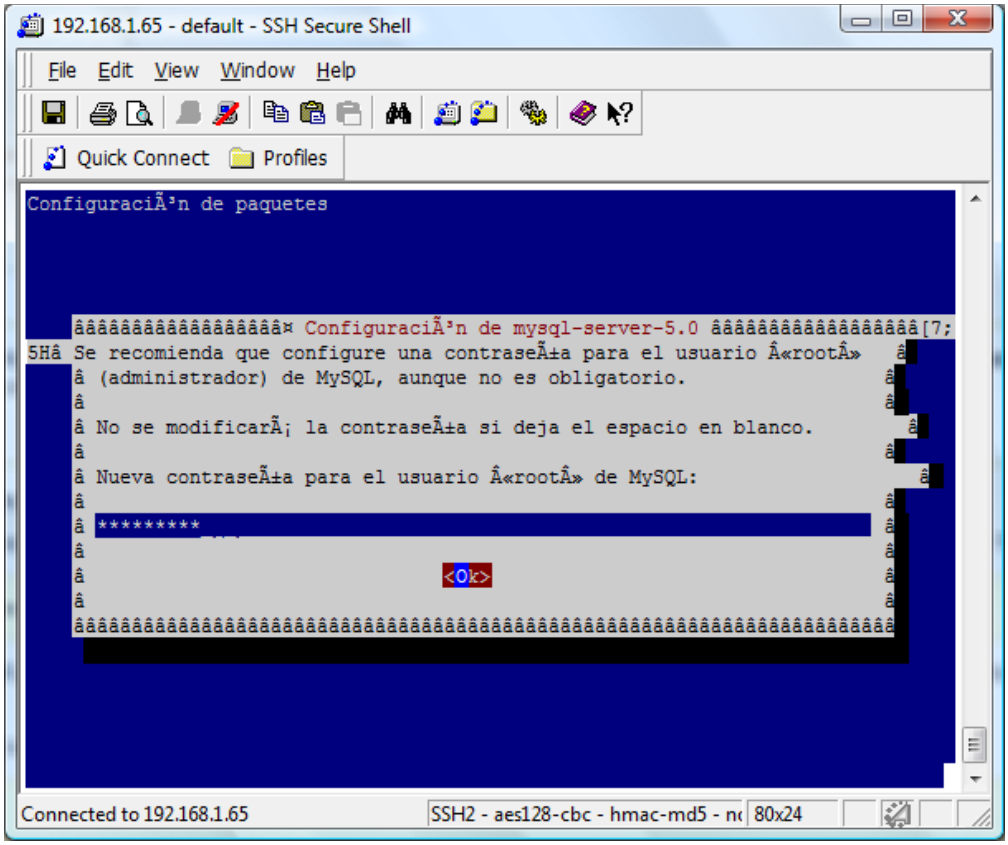


Figura A8.-Elección de contraseña de administrador para MySQL.

Posteriormente reiniciaremos el servicio:

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

Para comprobar que MySQL se ha instalado correctamente en nuestro servidor, iniciaremos sesión remotamente en el servidor MySQL y nos será mostrada la consola del servidor.

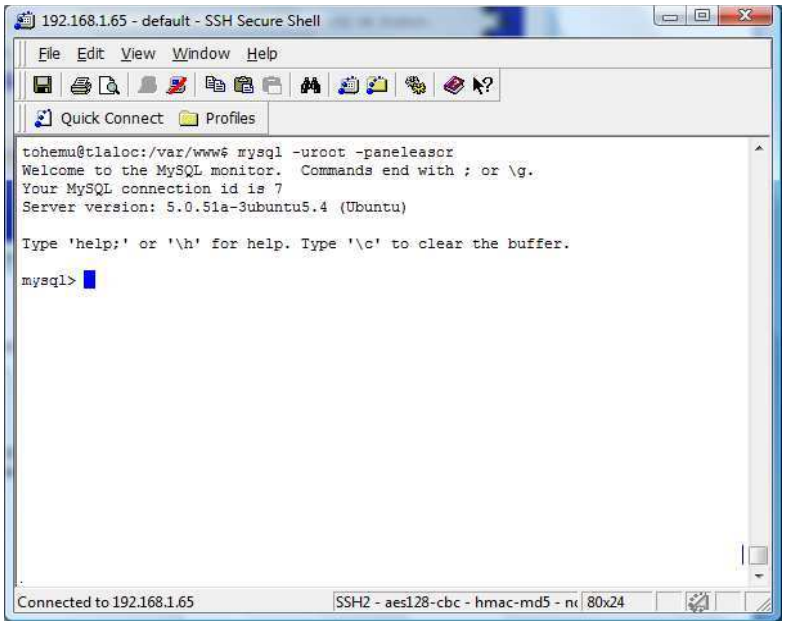


Figura A9.-Inicio de sesión en el servidor MySQL.

E.-Instalación de PHP utilizando aptitude.

Lo siguiente que haremos es instalar el soporte para PHP como módulo de Apache, para ello haremos uso del gestor de paquetes de Ubuntu 9.04 Server de nueva cuenta.

Para descargar e instalar este paquete, se tecleará el siguiente comando en consola:

```
sudo aptitude install php5
```

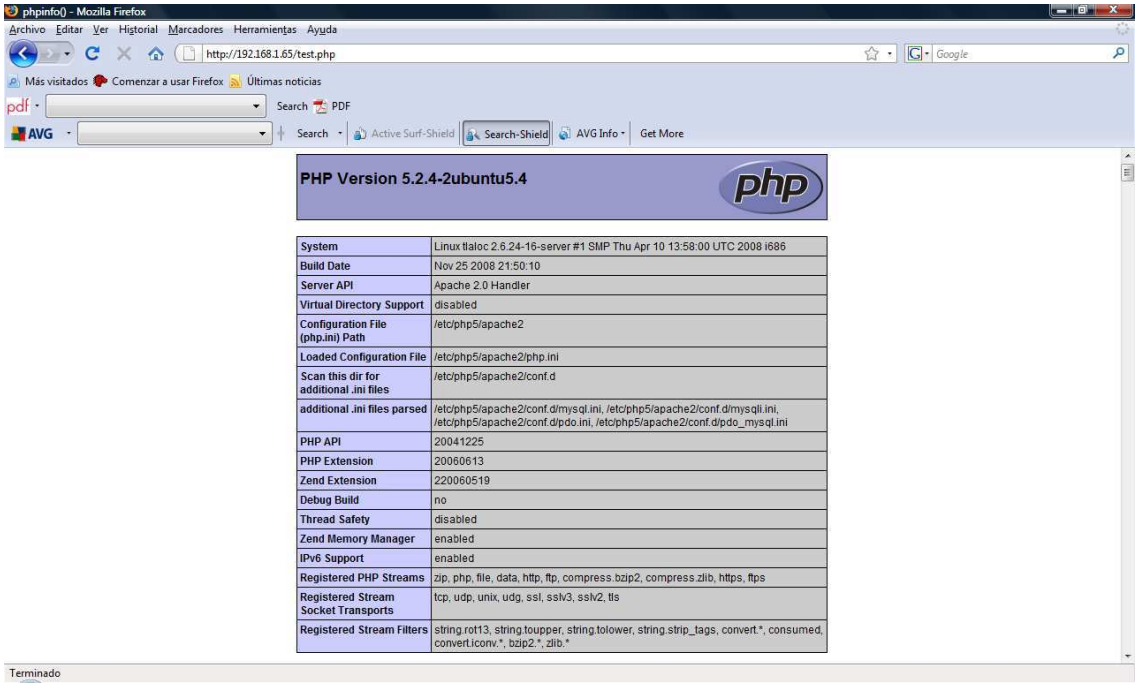
Seguido de esto debemos instalar otros dos paquetes que nos permitirán ligar a PHP con MySQL y con Apache. Esto se hace con los siguientes comandos:

```
sudo aptitude install libapache2-mod-auth-mysql
sudo aptitude install php5-mysql
```

Lo siguiente es reiniciar Apache:

```
sudo /etc/init.d/apache2 restart
```

Y por último verificaremos que PHP funciona, escribiendo un *script* sencillo en PHP y corriéndolo desde un navegador web.



System	Linux tlaloc 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686
Build Date	Nov 25 2008 21:50:10
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
additional .ini files parsed	/etc/php5/apache2/conf.d/mysql.ini, /etc/php5/apache2/conf.d/mysqli.ini, /etc/php5/apache2/conf.d/pdo.ini, /etc/php5/apache2/conf.d/pdo_mysql.ini
PHP API	20041225
PHP Extension	20060613
Zend Extension	220060519
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
IPv6 Support	enabled
Registered PHP Streams	zip, php, file, data, http, ftp, compress.bzip2, compress.zlib, https, ftps
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, sslv2, tls
Registered Stream Filters	string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, convert.iconv.*, bzip2.*, zlib.*

Figura A10.-Prueba de funcionamiento de PHP.

F.-Instalación de phpMyAdmin utilizando aptitude.

En esta parte instalaremos una interfaz web escrita en PHP que nos permitirá administrar nuestras bases de datos de manera remota, se trata de phpMyAdmin. Para utilizarlo se deben tener PHP y MySQL funcionando.

Primero se debe descargar este paquete:

```
sudo aptitude install phpmyadmin
```

Cuando se ejecute este comando, se nos pedirá la autorización para descargar e instalar el paquete, seleccionamos aceptar. Cuando termine de instalarse, un asistente de configuración aparecerá, en el cual se nos preguntará el servidor web que utilizamos, seleccionamos apache 2 y pulsamos Enter.

Lo que sigue es crear una liga simbólica al directorio web:

```
sudo ln -s /usr/share/phpmyadmin /var/www/
```

Posteriormente a esto reiniciamos Apache:

```
sudo /etc/init.d/apache2 restart
```

Posteriormente abrimos un navegador en cualquier cliente web y tecleamos la dirección URL: http://IP_SERVIDOR/phpmyadmin/ y verificamos que phpmyadmin se encuentra funcionando.

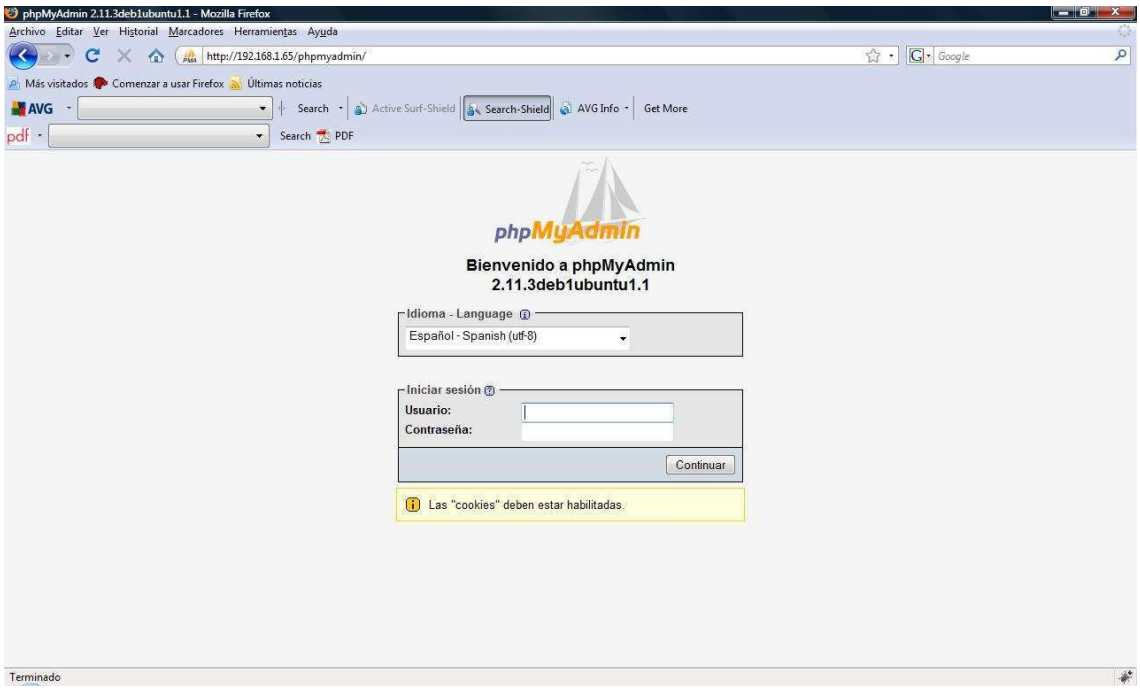


Figura A11.-Cliente MySQL phpmyadmin.

G.-Configuración de UFW.

Lo que haremos en esta parte será configurar una utilería con la que cuenta el sistema operativo Ubuntu 9.04 Server, esta utilería es UFW (*Uncomplicated Firewall*), que es un firewall que viene por defecto con este sistema operativo y es bastante sencillo de utilizar. UFW se basa en *iptables* y se utiliza principalmente como firewall basado en un *host*.

Como UFW ya viene instalado por defecto en nuestro sistema operativo, sólo hay que activarlo, ya que inicialmente viene desactivado. Para activar este firewall se debe utilizar el siguiente comando:

```
sudo ufw enable
```

Una vez que el firewall ha sido activado, aparecerá el mensaje que nos dará aviso de ello.

Ahora que nuestro firewall ha sido activado, nos daremos a la tarea de restringir los puertos de los servicios que no utilizaremos y sólo dejar abiertos los puertos para los servicios que nos interesan. Estos puertos son: 80-http, 22-ssh, 3306-MySQL y el puerto 443-https.

Primero que nada, denegaremos todas las conexiones:

```
sudo ufw default deny
```

Ahora deberemos abrir los puertos de los servicios que son de nuestro interés, en este caso: http, ssh, MySQL y https:

```
sudo ufw allow 80/tcp
sudo ufw allow 80/udp
sudo ufw allow 22/tcp
sudo ufw allow 22/udp
sudo ufw allow 3306/tcp
sudo ufw allow 3306/udp
sudo ufw allow 443/tcp
sudo ufw allow 443/udp
```

Luego de esto reiniciamos el firewall para que los cambios surtan efecto:

```
sudo ufw disable
sudo ufw enable
```

Para verificar el status del firewall, tecleamos el siguiente comando:

```
sudo ufw status
```

Con el comando anterior, se mostrará en pantalla información de los puertos que se encuentran abiertos en nuestro servidor.

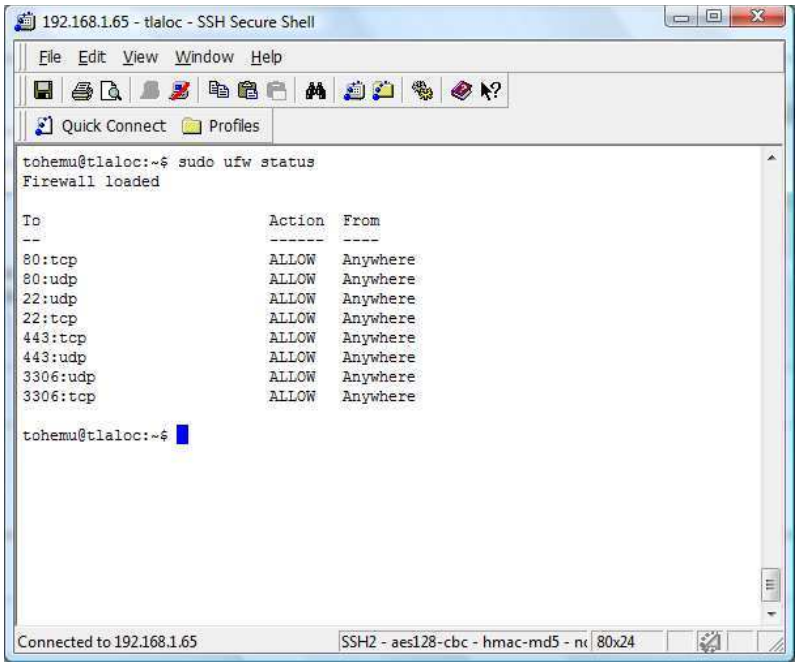


Figura A12.-Puertos permitidos por el Firewall.

Se han realizado una serie de pruebas que nos permitieron verificar el correcto funcionamiento de nuestro firewall, estas pruebas consistieron en cerrar y abrir puertos, verificando que los servicios se encontraban disponibles o negados.

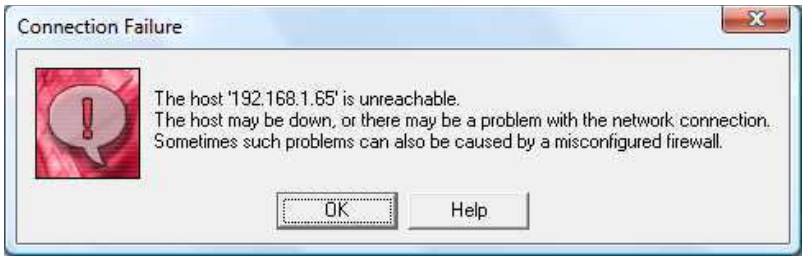


Figura A13.-Puerto 22 cerrado por el Firewall.

H.-Configuración segura de Apache por medio de HTTPS.

En esta parte haremos la configuración de nuestro servidor web para que se encuentre configurado de una manera más segura, instalaremos y configuraremos el módulo SSL/TLS para Apache (mod_ssl), que nos brindará soporte para HTTPS. Dicho módulo es de gran importancia, ya que le añade a nuestro servidor web la capacidad de cifrar los datos que intercambia y procesa, haciendo uso de algoritmos de cifrado como 3DES, RSA y otros.

Para instalar este módulo en nuestra distribución Linux (Ubuntu 9.04 Server), hay que teclear los siguientes comandos:

```
sudo aptitude install apache2 libapache-mod-ssl
sudo aptitude install openssl
```

Una vez instalado este módulo, se debe habilitar para su uso:

```
sudo a2enmod ssl
```

Después de esto, es posible generar un CRS (*Certificate Signing Request*), para lo cuál debemos crear nuestra propia clave, dicha clave se crea con el siguiente comando:

```
sudo openssl genrsa -des3 -out server.key 1024
```

Con el comando anterior estamos generando una clave privada RSA de 1024 bits para el servidor, dicha clave se guardará en el archivo server.key. Después de teclear el comando anterior, se nos pedirá una contraseña (debe ser robusta, por lo menos 8 caracteres, aunque cuando se especifica el argumento –des3, permite elegir de 4 caracteres en adelante), se la proporcionamos y con esto hemos terminado de crear la clave privada.

Lo siguiente es generar el CRS y para ello utilizaremos el siguiente comando:

```
sudo openssl req -new -key server.key -out server.crs
```

Después de teclear este comando se nos pedirá la contraseña que elegimos cuando creamos la clave privada y además se nos pedirán una serie de datos como el nombre de la compañía, la ciudad, el estado, el país, nuestro nombre, correo electrónico, etc. Luego de esto, el CRS se habrá creado y se habrá almacenado en el archivo server.crs.

Una vez que creamos nuestro CRS, deberemos crear un certificado autofirmado utilizando el CRS, para lo cual el siguiente comando es el apropiado:

```
sudo openssl x509 -req -days 3650 -in server.crs -signkey server.key -out server.crt
```

Aquí estamos especificando que el certificado autofirmado es un certificado del tipo x509, con una vigencia de 10 años (3650 días). Este certificado autofirmado se almacenará en el archivo server.crt.

Ya que hemos creado el certificado, es hora de instalarlo. Para ello ejecutaremos los siguientes comandos, con los que copiamos el certificado autofirmado y la clave privada del servidor a los directorios /etc/ssl/certs/ y /etc/ssl/private respectivamente, como se muestra:

```
sudo cp server.crt /etc/ssl/certs/
sudo cp server.key /etc/ssl/private/
```

Ahora deberemos crear el archivo /etc/apache2/sites-available/ssl. Este archivo será una copia del archivo /etc/apache2/sites-available/default, la copia la haremos de la siguiente manera:

```
sudo cp default ssl
```

Las primeras líneas de este archivo serán modificadas de manera que queden como sigue:

```
NameVirtualHost *:443
<VirtualHost *:443>
ServerAdmin webmaster@localhost
SSLEngine On
SSLCertificateFile /etc/ssl/certs/server.crt
SSLCertificateKeyFile
/etc/ssl/private/server.key
DocumentRoot /var/www-ssl/
<Directory />
```

Con esto, los archivos que se encuentren en el directorio /var/www-ssl/ serán los que utilizarán HTTPS (este directorio deberá ser creado).

Lo que deberemos hacer luego de esto, es editar el archivo /etc/apache2/ports.conf. Este archivo es muy pequeño, por lo que a continuación mostramos el archivo completo y como debe quedar una vez que se ha modificado:

```
Listen 80
<IfModule mod_ssl.c>
Listen 443
</IfModule>
```

Después de esto hay que crear una liga simbólica del archivo /etc/apache2/sites-available/ssl en /etc/apache2/sites-enabled/ssl:

```
ln -s /etc/apache2/sites-available/ssl /etc/apache2/sites-enabled/ssl
```

Para finalizar, sólo resta reiniciar nuestro servidor web Apache y todo ha quedado listo.

```
sudo /etc/init.d/apache2 restart
```

Con esto hemos configurado nuestro sitio web con el soporte para HTTPS.

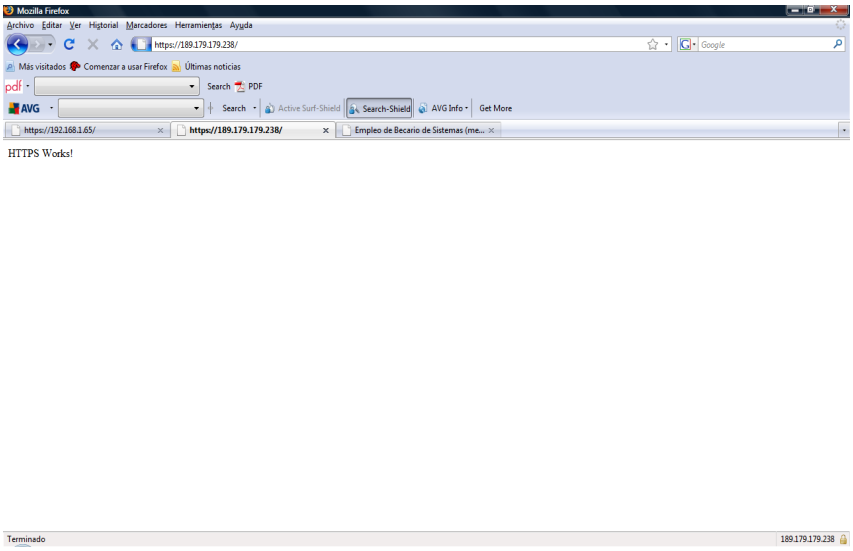


Figura A14.-HTTPS funcionando.

I.-Restricción de acceso de un usuario al home de otros usuarios.

A nosotros nos interesa que el directorio web de un usuario se encuentre en su directorio dentro de /home y que el usuario no tenga la posibilidad de ingresar al directorio de otros usuarios dentro de /home.

Para comenzar debemos crear un usuario, en nuestro caso lo hemos llamado “jaula”:

```
sudo adduser jaula
```

Después de esto se nos pedirá cierta información como la contraseña para el usuario y otros datos, tales como su nombre, su teléfono y otros.

Después de esto teclearemos el siguiente comando:

```
sudo dpkg-reconfigure -plow adduser
```

Al teclear este comando, aparecerá un ventana que nos explicará para qué sirve dicho comando, responderemos que No.

El comando anterior sirve para permitir o denegar el acceso al directorio de cada usuario dentro del directorio /home. Al teclear dicho comando y seleccionar la opción no, hemos hecho que cada usuario pueda solamente modificar el contenido de su directorio en /home y no pueda visualizar ni modificar el contenido de los directorios de otros usuarios en /home.

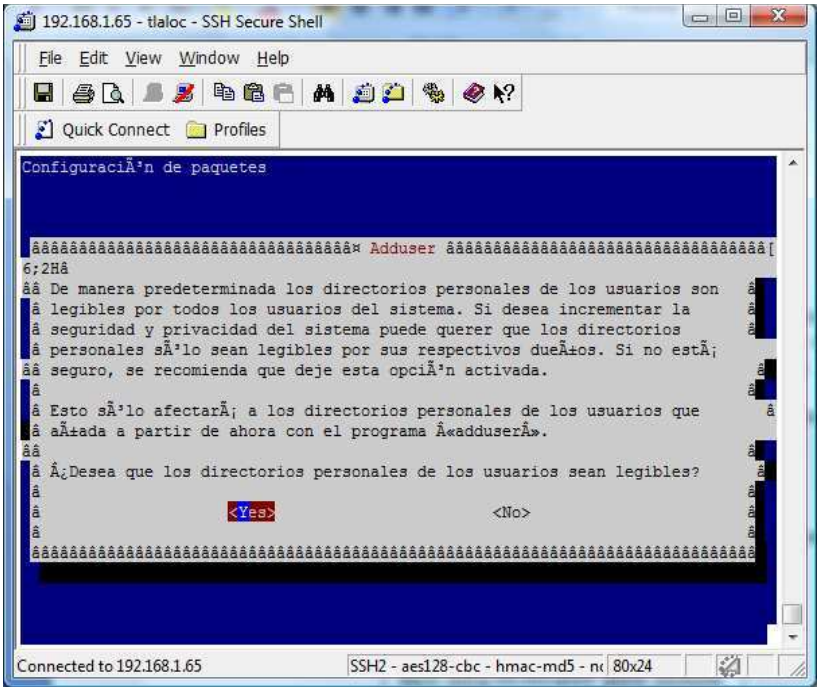


Figura A15.-Denegar el acceso de usuarios a directorios de otros usuarios.

Por último asignaremos permisos a los directorios de los usuarios dentro de /home:

```
sudo chmod o-r /home/*
```

Con este último comando estamos haciendo lo anterior para todos los archivos y subdirectorios del directorio de los usuarios dentro de /home.

Con lo anterior hemos logrado nuestro cometido de denegar el acceso de un usuario a un directorio web que no le pertenece.

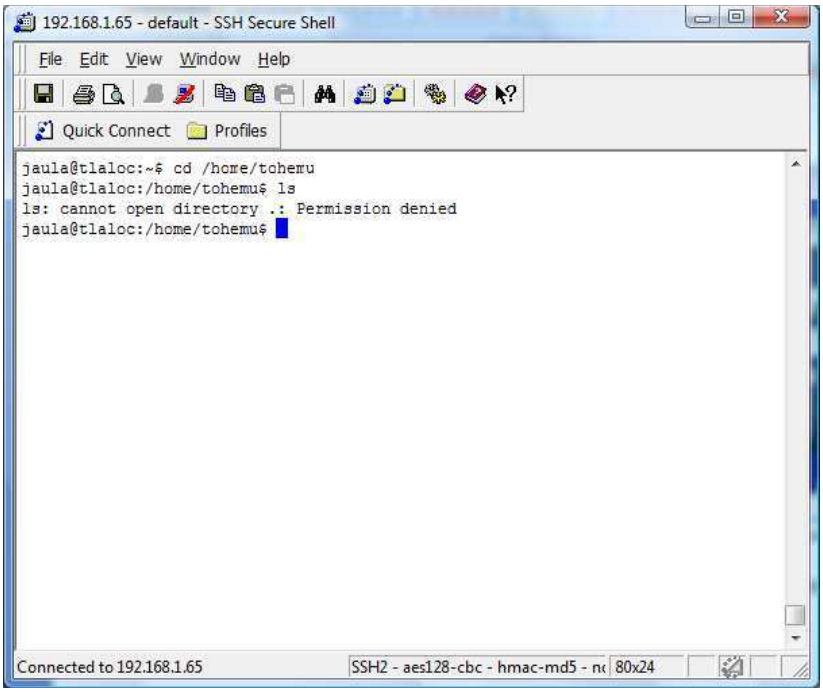


Figura A16.-Prueba de denegación de acceso al directorio de otro usuario.

Para hacer que el directorio de un usuario dentro de /home sea su directorio web, es necesario crear una liga simbólica hacia el directorio web:

```
sudo ln -s /var/www/tohemu /home/tohemu
```

Con el comando anterior, se logra que los archivos que el usuario ponga en su directorio, puedan ser leídos desde Internet. En este caso el directorio /home/tohemu hace referencia al directorio /var/www/tohemu.

J.-Respaldo de archivos del sistema.

Una buena práctica de seguridad es realizar respaldos de los archivos de los usuarios del servidor web. Muchos administradores realizan tareas programadas, que ejecutan un programa que hace esta tarea cada determinado tiempo. Nosotros realizaremos esta tarea con la ayuda de un demonio llamado cron, que nos permitirá hacer la ejecución de programas en ciertos periodos de tiempo definidos por nosotros.

Primero hemos escrito un script en bash que nos permite hacer un respaldo de los directorios web (/var/www y /var/www-ssl), este script copia estos directorios en un directorio llamado backupaammdd (aa-año,mm-mes,dd-día), luego de esto comprime dicho directorio y lo envía a un ordenador remoto utilizando un programa llamado scp.

El código de dicho script es el que se muestra abajo:

```
#
#Script para respaldar los directorios web de este servidor
#

cd /root

DATE=$(date +%y%m%d)

#Creamos el directorio de respaldos
sudo mkdir /root/backup

#copiamos el directorio /var/www a /root/backup
#y el directorio /var/www-ssl a /root/backup
sudo cp -r  /var/www /root/backup/www
sudo cp -r /var/www-ssl /root/backup/www-ssl

#Comprimimos el directorio backup
sudo tar -cvvf backup$DATE.tar backup/

#Copiamos el archivo de respaldo en un host remoto con ssh
sudo scp /root/backup$DATE.tar tohemu@192.168.1.64:/home/tohemu/www/backups


#borramos el directorio /root/backup y el archivo backup$DATE
sudo rm -rf backup
sudo rm  backup$DATE.tar
```

Para automatizar esta tarea se hará uso de cron, pero antes de esto se debe hacer una configuración en el ordenador remoto para que no solicite contraseña al iniciar sesión en él y poder enviar el archivo de respaldo. Esto sólo sirve mientras el servidor web se encuentra encendido.

Lo primero es crear un par de llaves pública y privada en el servidor web, como sigue:

```
sudo ssh-keygen -t dsa
```

Se nos solicitará el nombre del archivo donde se almacenarán las llaves, en nuestro caso, lo hemos denominado llaves, luego de esto se nos pedirá una contraseña y su confirmación, en donde no ingresaremos nada.

Una vez que se ha creado el archivo llaves, hay que copiarlo al ordenador remoto, en el que se almacenarán nuestros respaldos:

```
sudo scp llaves.pub tohemu@192.168.1.64:/home/tohemu/.ssh/authorized_keys
```

Con esto será posible transmitir archivos desde el servidor web a un ordenador remoto que almacenará nuestros respaldos.

Ahora si podremos automatizar esta tarea con cron, para lo cual ejecutaremos el comando:

```
sudo crontab -e
```

Este comando nos permitirá editar el archivo crontab, que es el archivo que nos permitirá programar tareas cada cierto tiempo.

En este archivo haremos la configuración para que nuestro script de respaldos se ejecute diario a las 6 a.m.

Luego de editar el archivo crontab, este debe quedar como sigue para que los datos del servidor web se respalden automáticamente en el ordenador remoto:

```
# m h dom mon dow command
  0  6  *    *    *    /root/backup-script
```

K.-Configuración segura de PHP.

El archivo de configuración de PHP se llama php.ini y en nuestro servidor web se encuentra localizado dentro del directorio de apache (/etc/apache2/php.ini*). En este archivo se encuentran todas las configuraciones de PHP. A nosotros nos interesa editar este archivo para darle una mayor seguridad a nuestro servidor web.

Básicamente en este archivo existen varias opciones de configuración que vienen establecidas y que nosotros modificaremos, ya que pueden representar un hoyo de seguridad en algún momento.

A continuación mencionamos las opciones de configuración que consideramos importantes para tener una seguridad mayor a la hora de servir páginas web dinámicas con PHP.

- Es recomendable deshabilitar el acceso remoto a archivos, ya que con funciones como fopen, file_get_contents, include, entre otras, permiten el acceso a archivos que se encuentran en otros hosts, además se les permiten a los

programadores considerar a las URLs como archivos. Para lograr deshabilitar el acceso remoto a archivos editamos la siguiente bandera del archivo de configuración de PHP como sigue:

```
allow_url_fopen = Off
```

De ser necesario el acceso a archivos remotos, se recomienda utilizar funciones de CURL (Client URL Library) o funciones como fsockopen.

- La transformación de parámetros en las peticiones HTTP que utiliza PHP es insegura, por lo que se debe estar deshabilitado el uso de variables globales. En la versión 4 de PHP y en versiones posteriores, esta opción viene deshabilitada por defecto, pero no está por demás verificar que la bandera se encuentre apagada:

```
register_globals = Off
```

- Es recomendable restringir los archivos a los que puede acceder PHP. La directiva open_basedir nos permite hacer lo anterior limitando el árbol de directorios a un directorio al que se puede acceder. Quizás esta sea la bandera más importante en relación a la seguridad de PHP. Su funcionamiento es como sigue:

```
open_basedir = /var/www/
```

El valor de esta bandera en el ejemplo nos indica que PHP únicamente podrá acceder a los archivos del directorio /var/www/.

- PHP también cuenta con un modo seguro en su archivo de configuración, sin embargo, el activarlo puede traer inconvenientes, ya que en el modo seguro permite a Apache acceder sólo a archivos de los cuales este sea dueño. Es recomendable activar esta bandera si no se configuró a PHP como módulo de Apache. La mejor forma de utilizarlo es configurando las banderas como sigue:

```
safe_mode = Off
safe_mode_gid = On
```

- Si se está trabajando con PHP en modo seguro, la ejecución de archivos binarios no está permitida, pero existe una bandera que nos permite especificar un

directorio en el cual se permite que los archivos binarios en el directorio especificado se puedan ejecutar, la bandera es la siguiente:

```
safe_mode_exec_dir = /www/ejecutables
```

- Al trabajar en el modo seguro de PHP, el acceso a variables de entorno tampoco se encuentra permitido, si se requiere el uso de algunas banderas, estas se pueden utilizar especificándolas en una lista (separada por comas) de prefijos que se permiten para las variables necesarias. Con la siguiente bandera podemos especificar las variables de entorno a las cuales se puede tener acceso:

```
safe_mode_allowed_env_vars = PHP_VARIABLE
```

En el ejemplo de arriba, la variable de entorno dada por VARIABLE, será la que contará con permiso para ser utilizada.

- Es importante controlar los archivos que se van a subir a un servidor. Si no es necesario subir archivos al servidor, se recomienda que esto no se permita o bien que se utilice un filtro para permitir subir solamente algunos tipos de archivos (no se recomiendan archivos binarios). Para impedir que se suban archivos al servidor basta con que la siguiente sentencia se encuentre como sigue:

```
file_uploads = Off
```

- Es importante limitar el tamaño de los archivos si se requiere permitir subirlos, esto lo podemos controlar con la siguiente bandera:

```
upload_max_filesize = 2M
```

- El límite en el tiempo de ejecución de un script es otro de los factores a tomar en cuenta, así como el tamaño de memoria máximo que tiene un script para ejecutarse, esto se controla con las siguientes banderas respectivamente:

```
max_execution_time = 30
memory_limit = 16M
```