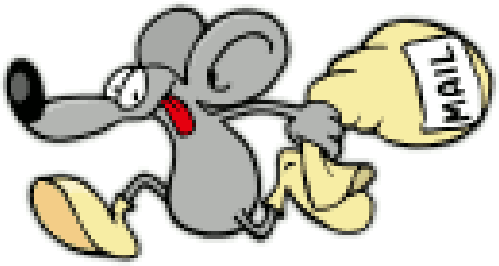
Instalación y bastionamiento de un servidor de correo bajo GNU/Linux



Lorena Fernández a.k.a. Loretahur

www.loretahur.es

INDICE

1. INTRODUCCIÓN

Esta documentación recoge la instalación de un servidor de correo usando la distribución **Debian** **Sarge** con un kernel 2.4.27-2-386 y usando exclusivamente software libre.

Para el ejemplo usaremos el dominio *prueba.com* gestionado por un servidor DNS (con bind 9) que reside en la propia máquina. En él hemos creado un registro de tipo MX para el servidor de correo con el siguiente nombre: *mail.prueba.com*.

Se ha elegido **Postfix** en su versión 2.1.5-9 como MTA. A Postfix se le han agregado los siguientes mecanismos de seguridad: TLS (Transport Layer Security) para cifrar las conexiones y SASL (Simple Authentication and Security Layer) como sistema de autentificación.

Todo el correo que pase a través del servidor SMTP será revisado en busca de virus y SPAM. Para llevar a cabo esta tarea se utilizará **AMaViSd-new** (en su versión 20030616p10-5) como interfaz entre el servidor de correo SMTP y las aplicaciones **ClamAV** (v. 0.84-2.sarge.10) y **Spamassassin** (v. 3.0.3-2sarge1), las cuales analizarán el correo en busca de virus y SPAM respectivamente.

Para la consulta de mensajes por parte de los usuarios se dispondrá de un servidor POP3 e IMAP, con sus respectivas versiones seguras con protocolo SSL, para lo cual se hará uso de **Courier** (courier-pop y courier-pop-ssl versión 0.47-4sarge5, courier-imap y courier-imap-ssl versión 3.0.8-4sarge5).

Los usuarios del correo no serán usuarios físicos de la máquina sino que serán usuarios virtuales almacenados en una Base de Datos **MySQL** (v. 4.0.24-10sarge2). Ha sido seleccionada esta tecnología para evitar que el archivo /etc/passwd se haga enorme. Sin embargo, ha sido descartada la idea de montar un directorio ldap puesto que el número de usuarios no es muy grande.

Para crear una autoridad certificadora y generar certificados propios se ha dispuesto de **OpenSSL** 0.9.7e.

2. CARACTERÍSTICAS DE POSTFIX

Postfix es un MTA que funciona sobre sistemas tipo UNIX. Se creó como alternativa a Sendmail y fue escrito en C por Wietse Zweitze Venema cuando trabajaba para IBM. Es un servidor de código abierto y además gratuito.

Tiene una arquitectura y diseño muy modular lo que hace que sea muy rápido y tenga un gran rendimiento. Es fácil de administrar y configurar.

Además ofrece soporte para las tecnologías más usadas hoy día: LDAP, Bases de datos (MySQL y PostgreSQL), autenticación mediante SASL, LMTP, etc.

Se pueden lanzar varias instancias de Postfix en la misma máquina con distintas configuraciones, usando cada una distinta dirección IP, distintos puertos, etc. con la única limitación de que ambas no compartan el directorio de colas y que usen distintos valores para myhostname.

Las ventajas que presenta Postfix frente a Sendmail son las siguientes:

- Postfix es más **seguro** que Sendmail. Los procesos que conforman Postfix se comunican a través de sockets que se crean en un directorio de acceso restringido (trabajan en modo chroot, por tanto, cualquier directorio o fichero que esté fuera les quedará inaccesible). La información que intercambian los diversos procesos es la mínima posible y cada proceso corre con los mínimos permisos necesarios para realizar su tarea.

- Mientras que Sendmail es monolítico, Postfix es modular y esto le hace ser más liviano puesto que sólo carga los módulos que necesita en cada momento. Por tanto, presenta un mejor **rendimiento** y una mayor **rapidez**.

- Postfix es mucho más fácil de **configurar** y **administrar** que Sendmail.

- Postfix evita utilizar **buffers** de tamaño fijo, evitando que tengan éxito ataques buffer overflow.

- Los **procesos** que no se necesitan se pueden **desactivar**. Por ejemplo, en una máquina dial-up que sólo envía correo podemos desactivar el proceso servidor SMTPD.

- Postfix complementa su seguridad con un uso sencillo de **listas negras** y una fácil integración con **antivirus**.

3. PROTOCOLOS

A continuación se va a describir los protocolos más relevantes que intervienen en las comunicaciones de correo.

POP o POP3 (Post Office Protocol)

Es el protocolo de nivel de aplicación que se usa para que los clientes locales de correo se descarguen los mensajes almacenados en un buzón de un servidor de correo. A diferencia de IMAP, no requiere estar conectado permanentemente al servidor, sino que podemos descargarnos los correos y trabajar en local. Es un protocolo poco pesado para la máquina servidora (se produce una conexión, una descarga y luego una desconexión) y además es muy simple: tan sólo 13 comandos que pueden responder con +OK o –ERR.

Normalmente, las conexiones con este protocolo se realizan a través del puerto TCP 110.

Este protocolo tiene su versión cifrada: POP3S cuyo puerto well known es el 995/tcp.

Los principales comandos que se usan son los siguientes:

- USER <nombre>: identificación de usuario (sólo se realiza una vez).
- PASS <password>: envías la clave del servidor.
- STAT: da el número de mensajes no borrados en el buzón y su longitud total.
- LIST: muestra todo los mensajes no borrados con su longitud.
- RETR <número>: solicita el envío del mensaje especificando el número (no se borra del buzón).
- TOP <número> <líneas>: muestra la cabecera y el número de líneas requerido del mensaje especificando el número.
- DELE <número>: borra el mensaje especificando el número.
- RSET: recupera los mensajes borrados (en la conexión actual).
- QUIT: salir.

Ejemplo:

```
telnet mail.prueba.com 110
+OK Hello there.
USER si@prueba.com
+OK Password required.
PASS ****
+OK logged in.
STAT
+OK 2 2397
QUIT
+OK Bye-bye.
```

IMAP (Internet Message Access Protocol)

Protocolo de recepción de correos similar a POP pero con la diferencia de que con POP, el cliente de correo se descarga los mensajes para que el usuario interaccione con ellos y con IMAP, se accede directamente al buzón de correo del servidor, por lo que, cualquier modificación que haga el usuario, la está haciendo sobre el servidor. Por ejemplo, si con IMAP marcamos un correo como leído, lo marcamos en el servidor mientras que con POP, lo marcamos en el cliente utilizando un ID. Además, este protocolo soporta crear, modificar y/o eliminar buzones.

El puerto TCP por el que escucha por defecto es el 143. La versión cifrada de IMAP: IMAPS, escucha por defecto en el 993/tcp.

A la hora de recomendar el uso de un protocolo de recepción de mensajes al usuario, será mejor POP frente a IMAP puesto que consume menos recursos del servidor de correo y menos espacio en disco.

SMTP (Simple Mail Transfer Protocol)

Protocolo de red basado en texto utilizado para el intercambio de mensajes de correo electrónico entre dispositivos sin ningún tipo de autenticación. Es un protocolo muy simple pero a la vez muy inseguro:

- Las cabeceras son altamente spoofeables, lo que deriva en la aparición de SPAM.
- Habitualmente las comunicaciones mediante protocolo SMTP se realizan sin utilizar mecanismos de cifrado, por lo que toda la información viaja en claro por Internet.

El Well Known port para SMTP es el 25/tcp.

Los comandos básicos de SMTP son:

- EHLO/HELO: el servidor emisor comunica al receptor quién es. Esto es fácilmente spoofeable.

- MAIL FROM: dirección del emisor (no confundir con la cabecera *From:*).

- RCPT TO: dirección del destinatario (no confundir con las cabeceras *To:* y *CC:*).

- DATA: contenido del mensaje. Dentro se puede definir el from y el subject. Se da por terminado cuando se escribe un punto en una nueva línea.

- QUIT: cortar la conexión.

Ejemplo:

```
telnet mail.prueba.com 25
220 mail.prueba.com ESMTP
HELO mail.deusto.es
250 mail.prueba.com
MAIL FROM: pepito@deusto.es
250 Ok
RCPT TO: jaimito@prueba.com
250 Ok
DATA
354 Please start mail input.
FROM: Pepito <pepito@deusto.es>
SUBJECT: Esto es una prueba
Hola, esto es una prueba.
.
250 Mail queued for delivery.
QUIT
221 Closing connection. Good bye.
```

TLS (Transport Layer Security o Seguridad para Capa de Transporte)

En 1999, con aplicación no sólo a SMTP, se definió el protocolo TLS basado en SSL (Secure Socket Layers), y cuya definición formal puede encontrarse en el *RFC-2246*.

TLS básicamente proporciona cifrado en las comunicaciones y autentificación entre ambos corresponsales mediante el uso de certificados X.509. Mejora la comunicación TCP añadiendo cifrado e integridad en los correos. No protege el contenido de los mails (para eso está PGP o S/MIME) sino que lo que hace es proteger la comunicación completa. Necesita para ello un par de claves pública y privada emitidas por una autoridad certificadora (CA).

ESMTP (Enhanced Simple Mail Transfer Protocol)

La integración del protocolo TLS y SMTP se define en el *RFC-2487*, y se implementa en ESMTP, en concreto en la negociación inicial (EHLO en lugar de HELO).

El servidor ofrece la prestación de TLS mediante la opción STARTTLS, invitando al cliente a enviar la directiva STARTTLS y pasar a un estado de comunicaciones cifradas.

ESMTP tiene otras extensiones a parte de STARTTLS como son 8BITMIME, AUTH,…

Ejemplo:

```
telnet mail.prueba.com 25
220 mail.prueba.com ESMTP
ehlo mail.deusto.es
250-mail.prueba.com
250-PIPELINING
250-SIZE 10485760
250-ETRN
250-STARTTLS
250-AUTH LOGIN PLAIN
250-AUTH=LOGIN PLAIN
250 8BITMIME
```

Como puede apreciarse, una vez realizada la identificación inicial mediante la directiva EHLO, el servidor proporciona la relación de funciones que soporta, entre las que aparece el protocolo TLS mediante la opción STARTTLS.

SASL (Simple Authentication and Security Layer)

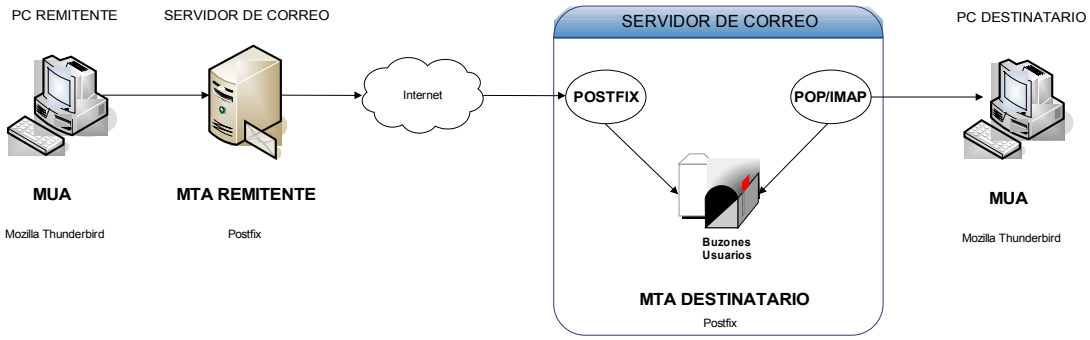
SASL es un entorno de trabajo para autenticación y autorización en protocolos de Internet. Separa los mecanismos de autenticación de los protocolos de la aplicación permitiendo, en teoría, a cualquier protocolo de aplicación que use SASL usar cualquier mecanismo de autenticación soportado por SASL.

Mediante SASL sólo se maneja la autenticación y se requieren de otros mecanismos, como por ejemplo TLS, para cifrar el contenido que se transfiere. Si no, la password que va en claro por la red podría ser interceptada. Así, añadiendo TLS desde el inicio, ciframos las conexiones y no importa que la contraseña vaya en texto claro.

¿Por qué es necesario SASL? Como se ha comentado antes, SMTP es un protocolo orientado a red sin ningún tipo de autenticación. Podríamos introducir en una base de datos las IP's de los clientes, pero si las conexiones son con IP dinámica y el cliente itinerante, puede ser un caos además de imposible de mantener.

4. FUNCIONAMIENTO DE UN SISTEMA DE CORREO

El servicio de correo electrónico consta de dos partes bien diferenciadas: aquella con la que trata el usuario, y aquella que se encarga de transportar los mensajes del origen al destino. A menudo hay un componente adicional encargado de distribuir el correo que llega a la máquina destino a una ubicación especial dentro de ésta, propia de cada usuario. Los agentes que intervienen son los siguientes:



1) **MTA (Mail Transfer Agent)**: agente de transporte de correo encargado de recoger mensajes y enviarlos, comunicando para ello con otros MTA según sea preciso. Se encuentra en el servidor. Puede hacer dos cosas:

   a. Si el mensaje es para un usuario local, se lo pasará a un MDA para que lo guarde en el buzón correspondiente.

   b. Si el mensaje es para un usuario remoto, se lo entregará al MTA que le corresponda.

   Ejemplos: Sendmail, Postfix, Exim, Microsoft Exchange, QMail.

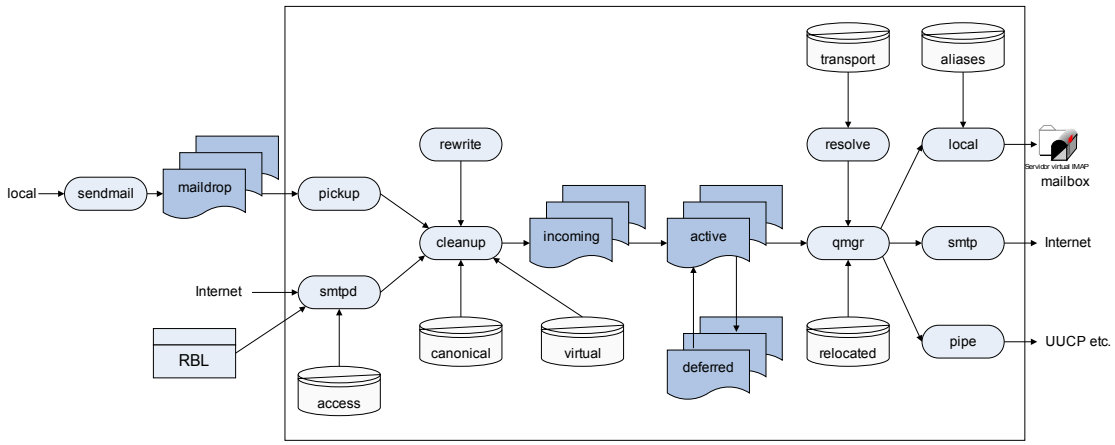
2) **MDA (Mail Delivery Agent)**: es el encargado de depositar el correo en los buzones de los usuarios. Suele hacer varios tipos de filtrado y clasificación antes de esto. También se encuentra en el servidor.

   Ejemplos: procmail, maildrop.

3) **MUA (Mail User Agent)**: cliente de correo que usa el usuario para escribir/leer el correo. Se encuentra en el cliente

   Ejemplos: Mozilla Thunderbird, Microsoft Outlook, Pegasus Mail, Mutt, etc...

En nuestro caso, hemos seleccionado Postfix como MTA. La arquitectura de Postfix es la siguiente:



Postfix funciona gestionando cuatro colas: *maildrop*, *incoming*, *active* y *deferred.*

- **Maildrop**: es la cola donde se procesan los mensajes generados y/o entregados localmente, mediante la versión de Postfix de /usr/lib/sendmail.
- **Incoming**: aquí van los mensajes recibidos por SMTP de otros hosts (tras recibir cierto procesamiento) y los que han pasado por la cola maildrop. Si llegan correos y Postfix no puede atenderlos se quedan esperando en esta cola.
- **Active**: almacena los mensajes que se están intentando enviar en un momento dado. Tiene un espacio limitado.
- **Deferred**: los mensajes que por diversas causas no se pueden encaminar o están pendientes de reintentar su encaminamiento.

Postfix gestiona estas colas mediante procesos independientes. El proceso *pickup* toma los mensajes procedentes de la cola *maildrop* y los pasa a *cleanup*, que analiza las cabeceras de los mensajes y deposita éstos en la cola *incoming*.

El proceso *qmgr* es el encargado de tratar los mensajes que llegan a la cola *incoming*, depositarlos en *active* y lanzar el proceso adecuado para su encaminamiento, como pueden ser *local*, *smtp* o *pipe*.

El proceso *local* es el encargado de depositar el correo en el buzón del usuario receptor. *Smtp* es el proceso que envía el correo al host destino mediante protocolo SMTP.

El correo procedente de otros sistemas se atiende a través del proceso *smtpd*, utilizando el protocolo SMTP.

Además de las colas, Postfix trabaja con unas tablas que, creadas por el administrador con el comando **postmap**, sirven a los diferentes procesos para concretar el tratamiento que debe darse a cada mensaje. Se usan seis tablas: *access*, *aliases*, *canonical*, *relocated*, *transport* y *virtual* (no es obligatoria la existencia ni utilización de todas ellas).

La tabla *access* permite definir una relación explícita de sistemas a los que se les deben aceptar o rechazar sus mensajes. La utiliza el proceso *smptd*. Un ejemplo de

entrada en esta tabla en la que se rechaza los mensajes procedentes del dominio todocoleccion.com es:

```
@todocoleccion.com      REJECT
```

La tabla *aliases*, al igual que en Sendmail, define una serie de nombres alternativos a usuarios locales, y la consulta el proceso *local*. Para crearla o actualizarla se utiliza el comando de Postfix **newaliases**.

El proceso *cleanup*, mediante la tabla *canonical* establece relaciones entre nombres alternativos y nombres reales, ya sean usuarios locales o no.

El proceso *qmgr* utiliza la tabla *relocated* para devolver los mensajes de usuarios que han cambiado de dirección: “User has moved to *new-email*”.

Con la tabla *transport*, que es utilizada por el proceso *trivial-rewrite*, se define la política de encaminamiento por dominios, subdominios e incluso por dirección concreta de usuario.

Para la gestión y soporte de dominios virtuales el proceso *cleanup* utiliza la tabla *virtual*. En ella se establecen las relaciones entre usuarios virtuales y reales, e incluso de dominios completos.

Postfix soporta muy diversos soportes de backend para las tablas. Algunos de ellos son:

- **Hash**: el archivo generado es un hash. Está disponible para sistemas con soporte BD db (así viene por defecto en la instalación de postfix).
- **MySQL**: mapeo de las tablas de postfix a MySQL. Actualmente no es la mejor solución aunque es bastante sencilla de implementar.
- **PostgreSQL**: mapeo de las tablas de postfix a PostgreSQL. Esto tiene una dificultad mediana.
- **LDAP**: mapeo de las tablas de postfix a LDAP. Es la mejor solución aunque la más complicada de implementar, sobre todo contra Active Directory. No hay esquemas propios para LDAP.

Proceso de envío de mensajes

Cuando un usuario envía un correo electrónico desde un programa cliente de correo electrónico (MUA), éste conecta con un servidor SMTP, al que le deja el mensaje para su envío a una o varias cuentas de correo destinatarias.

Este servidor (MTA del remitente) comprueba si el destinatario es local. Si es así, el MDA deposita el mensaje en el buzón correspondiente. Por el contrario, si el destinatario no pertenece a ese MTA, éste se conectará con el servidor donde esté alojada la cuenta de correo del destinatario (MTA del destinatario) y transmitirá el mensaje para su almacenamiento y posterior descarga por el destinatario.

Proceso de lectura de mensajes

El usuario, para leer sus mensajes, se conectará al servidor a través de un MUA mediante POP o IMAP.

Los servidores POP e IMAP no forman parte de Postfix sino que hay que recurrir a paquetes específicos como es el caso de courier-pop y courier-imap.

Nuestro cliente conectará al puerto 110 del servidor en caso de POP o al 143 en caso de IMAP.

Si el método usado es POP, los mensajes por defecto se borrarán del servidor una vez termine la conexión a no ser que se indique lo contrario. En cambio, mediante IMAP los mensajes se leen en el servidor directamente.

Buzones

Los buzones de los usuarios en el servidor de correo pueden tener dos formatos diferentes: mbox y maildir.

Mbox es el método tradicional de almacenar correos en servidores UNIX. Los mensajes se guardan en un único archivo en el que se van encolando según van llegando al buzón. Para indicar el comienzo de un mail dentro del archivo se utiliza la palabra *From* y para indicar el fin, una línea en blanco.

El formato **Maildir** consiste en un directorio con tres subdirectorios (new, cur y tmp) en el que se van guardando los mensajes en diferentes archivos. En new se almacenan los mensajes nuevos, en cur los mensajes ya leídos pero que no han sido descargados del servidor y en tmp los mensajes recién recibidos que aún no han sido pasados a new (una vez depositado en tmp, se moverá a new).

Para esta instalación se ha seleccionado como formato Maildir por varias razones:

- Courier-IMAP necesita este formato.
- Es más seguro puesto que si un mensaje se corrompe no afecta a todos los demás, como es el caso del formato mbox.
- Maildir no bloquea los ficheros para mantener la integridad del mensaje, porque los mensajes se almacenan en ficheros distintos con nombres únicos. Con mbox, sin embargo, sólo un proceso puede abrir el archivo en modo lectura/escritura.
- Mbox es muy poco óptimo para buzones grandes.

5. INSTALACIÓN Y CONFIGURACIÓN DE POSTFIX

Instalaremos los paquetes de Postfix de los repositorios de Debian con:

```
#apt-get install postfix
```

También debemos instalar el soporte que tiene Postfix para MySQL:

```
#apt-get install postfix-mysql
```

Los archivos más importantes de configuración de Postfix son los siguientes:

- **main.cf** (/etc/postfix/main.cf): archivo de configuración principal de Postfix donde se encuentran los parámetros que controlan el comportamiento del MTA.
- **master.cf** (/etc/postfix/master.cf): archivo de configuración del demonio master. Determina qué procesos serán arrancados y con qué opciones.
- **aliases** (/etc/aliases): archivo de alias. Equivalencia entre una dirección ficticia y una dirección local real.

6. MAIN.CF

En la ruta /etc/postfix, por defecto, tenemos el archivo principal de configuración de Postfix denominado main.cf.

La estructura de este archivo es el de una línea por cada parámetro con la siguiente sintaxis:

```
parámetro = valor
```

Cada parámetro debe empezar en la primera columna, es decir, no puede haber espacios al principio de cada línea porque si no, postfix pensará que se trata de la continuación de la anterior.

Los comentarios empiezan por el carácter # y un comentario no puede comenzar en medio de una línea.

Cuando a un parámetro le asignamos un valor, se puede hacer referencia más adelante como si fuera una variable. Por ejemplo, si declaramos: `mydomain = dominio.com` puede ser usado el valor en otras líneas como `$mydomain`

Los parámetros no especificados cogen su valor por defecto.

Algunas direcciones IP pueden encubrirse para evitar que Postfix resuelva el nombre haciendo peticiones al DNS (naked ip addresses). Para introducir una direccion IP desnuda se usan los corchetes cuadrados [ ]. Ejemplo: `parametro = [192.168.1.1]`

Pasaremos a continuación a describir los campos más relevantes y la configuración que se le ha dado para este proyecto.

- **myhostname**: indica a Postfix el nombre del servidor (FQDN). Si no lo ponemos lo cogerá de la información que esté almacenada en /etc/hostname.

```
myhostname = mail.prueba.com
```

- **mydomain**: con este parámetro postfix puede servir a varios dominios. Si no lo ponemos, cogerá el parámetro myhostname quitando la primera parte hasta el punto.

```
mydomain = prueba.com
```

- **myorigin**: será, para el correo local, el origen del correo cuando en el mail sólo se indica el usuario.

```
myorigin = $mydomain
```

- **mydestination**: aquí se declaran todos los dominios que deben considerarse locales al sistema a efectos de encaminamiento del correo electrónico.

```
mydestination = $myhostname, localhost.$mydomain,
                localhost, $mydomain
```

- **home_mailbox**: la siguiente opción nos permite elegir en qué tipo de "formato" se van a guardar los mensajes en el buzón de cada usuario: mbox (fichero) o maildir (directorio).

```
home_mailbox = Maildir/
```

  Además de modificar este parámetro, no debemos olvidarnos de cambiar también el fichero /etc/login.defs. Aquí tenemos que descomentar la línea `QMAIL_DIR Mailidir` para indicar que todos los usuarios que hagan login en nuestro sistema usarán también el formato Maildir (por tanto, debemos comentar las dos que aparezcan a su lado).

- **mail_spool_directory**: indica el directorio donde se almacenan los buzones de correo.

```
mail_spool_directory = /var/spool/mail/
```

- **notify_classes**: en ocasiones, algunos mensajes no llegan porque se producen errores. Postfix puede notificar automáticamente, mediante un mensaje al usuario Postmaster, de aquellas incidencias que se han declarado de interés para su administración. Con el parámetro notify_classes indicamos el nivel de error a notificar. Los valores que puede tomar son:

- bounce: si un mensaje no puede ser encaminado, se envía otro mensaje al remitente y una copia al Postmater incluyendo el original. En el caso del Postmaster sólo se incluyen las cabeceras por razones de privacidad.
- 2bounce: en el caso en que la notificación de un mensaje de error de encaminamiento genere también un error de encaminamiento.
- delay: se informa a Postmaster de que hay mensajes pospuestos por problemas en su encaminamiento.
- policy: informa a Postmaster de las peticiones rechazadas por políticas UCE de otros servidores. Llega una copia de la transacción.
- protocol: informa a Postmaster de cualquier error de protocolos, cliente o servidor, o intentos de algún cliente de ejecutar comandos no implementados. Se recibe una copia de la transacción completa.
- resource: informa a Postmaster de los mail no entregados por algún problema de recursos (errores read/write, queue, etc).
- software: informa a Postmaster que un mensaje no ha sido encaminado por problemas de software.

```
notify_classes = resource, software
```

- **relay_domains**: aquí se declaran los dominios a los que se autorizará mandar mensajes a través de nuestro servidor. Deberemos prestar especial atención a este parámetro para evitar así que nuestro servidor se convierta en un open relay (open relay es cuando desde nuestro servidor se permite mandar mensajes a otras direcciones. Esto nos convierte en blancos fáciles de los generadores de SPAM y por tanto pronto pasaríamos a formar parte de las RBLs).

```
relay_domains = $mydestination
```

- **mynetworks**: el rango de direcciones IP que van a poder enviar a través de este MTA, es decir, redes a las que permito hacer relay puesto que son locales a Postfix. Las redes deberán ser especificadas en formato Classless Internet Domain Routing (CIDR).

```
mynetworks = 192.168.1.0/24, 127.0.0.0/8
```

- **smtpd_banner**: texto que acompaña a la respuesta 220 de bienvenida. A la hora de seleccionar el banner que aparecerá al conectarse a nuestro servidor de correo, no deberemos dar excesiva información como por ejemplo, el Sistema Operativo o que se trata de un servidor de correo Postfix. De esta forma evitaremos que si alguien nos hace un fingerprinting o escaneo lo tenga relativamente fácil para futuros ataques e identificación de vulnerabilidades. Por eso sólo mostraremos el nombre del servidor de correo y el protocolo: *220 mail.prueba.com ESMTP*

```
smtpd_banner = $myhostname ESMTP
```

- **setgid_group**: identificador del grupo destinado exclusivamente a la gestión de correo y de colas.

```
setgid_group = postdrop
```

- **queue_directory**: especifica el lugar de la cola de Postfix. Es también el directorio raíz de los demonios de Postfix (que corren enjaulados).

```
queue_directory = /var/spool/postfix
```

- **command_directory** y **daemon_directory**: contienen la ruta donde están los comandos y los demonios de Postfix, respectivamente.

```
command_directory=/usr/sbin
daemon_directory=/usr/lib/postfix
```

- **mail_owner**: indica el usuario que es propietario de la cola de Postfix. Se debe usar un usuario dedicado y como la instalacion de Postfix crea el usuario y el grupo postfix, usaremos éste.

```
mail_owner = postfix
```

- **alias_maps** y **alias_database**: archivos para los alias.

```
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases
```

- **virtual_mailbox_base**: este parámetro es lo que se le agrega al valor que tenemos en la Base de Datos MySQL para conseguir llegar hasta el buzón del usuario. Vamos a dejarlo con "/"

```
virtual_mailbox_base=/
```

- **virtual_uid_maps** y **virtual_gid_maps**: señalamos a Postfix que los UserIDs y GroupIDs de los usuarios de correo los obtendrá por medio del archivo indicado, que accederá a MySQL.

```
virtual_uid_maps=mysql:/etc/postfix/ids.cf
virtual_gid_maps=mysql:/etc/postfix/gids.cf
```

- **virtual_mailbox_maps**: con este parámetro indicamos que mediante el archivo mysql_virt.cf vamos a acceder a MySQL para ver dónde están los buzones de los usuarios:

```
virtual_mailbox_maps=mysql:/etc/postfix/mysql_virt.cf
```

- **local_transport**: le indicamos que deberá entregar el correo a usuarios virtuales y no a locales.

```
local_transport = virtual
```

- **disable_vrfy_command**: para no permitir la verificación que hacen los spamers buscando usuarios válidos, deberemos deshabilitar el verify (que se usa para saber si un usuario existe o no en nuestro sistema).

```
disable_vrfy_command = yes
```

- **message_size_limit**: para evitar que nuestros propios usuarios provoquen un DoS tratando de enviar un mensaje excesivamente grande, limitaremos el tamaño máximo a 10 Mb (incluidas cabeceras). Si no, podría suceder que Postfix se bloquease al intentar procesar este tipo de mensajes.

```
message_size_limit = 10485760
```

  Debe tenerse en consideración que si se incrementa mucho el *message_size_limit*, llegando a superar el valor de *mailbox_size_limit*, Postfix dará un error de configuración al arrancar.

- **mailbox_size_limit**: con este parámetro indicamos el límite del tamaño de los buzones de correo. En nuestro caso, no pondremos límite:

```
mailbox_size_limit = 0
```

- **smtpd_delay_reject**: aunque postfix determine que el correo no es válido en alguna de sus comprobaciones, lo deja pasar por el resto (con un consumo de CPU). Para evitar esto podemos poner a *no* el parámetro smtpd_delay_reject. Sin embargo, hay que tener cuidado con esta directiva porque si te rechaza la conexión por ejemplo a nivel smtpd_client_restrictions no contarás con logs ni del from, ni del to, etc... con lo que la búsqueda de correos "perdidos" en logs es peor (no compensa por tanto el ahorro de recursos).

```
smtpd_delay_reject = no
```

6.1. MASTER.CF

El programa *master* es el encargado de orquestar a los diferentes procesos de Postfix, arrancando en cada momento el necesario. Para ello sigue las indicaciones de su fichero de configuración: *master.cf* que normalmente se encuentra en el directorio */etc/postfix*. En él se especifican los procesos y sus parámetros.

Cada entrada en el fichero es un conjunto de ocho campos separados por blancos o tabuladores:

```
service type private unprivileged chroot wakeup maxprocess command
```

- **Service**: nombre del servicio que se está configurando. Ejemplos de servicios: cleanup, pickup, qmgr,…

- **Type**: tipo de comunicación de transporte utilizado por el servicio para comunicarse con otros módulos. Puede ser de tres tipos: internet sockets (inet), Unix sockets (unix) y pipes con nombre (fifo).

- **Private**: indica cuándo el canal de comunicación de un proceso debe estar accesible a procesos ajenos a Postfix.

- **Unprivileged**: especifica con qué privilegio de usuario se ejecuta el servicio. Si se especifica "**y**" (que es el valor por omisión), el servicio se ejecuta con los del usuario descrito en la directiva **mail_owner** de *main.cf*, que por defecto es **postfix**. Si se indica "**n**", el servicio se ejecuta con privilegios de **root**.

- **Chroot**: indica si el servicio se ejecuta en un entorno *chroot*, lo que proporciona niveles adicionales de seguridad. Esta opción se activa indicando "**y**". Una única restricción: los procesos *local* y *pipe* no pueden ejecutarse en modo *chroot*.

- **Wakeup**: indica los segundos que deben transcurrir para que el proceso master envíe una señal para despertar el servicio correspondiente. Existe una opción adicional, que es añadir el símbolo **?** al final del valor que señala el intervalo. Con esto se indica al proceso *master* que sólo envíe la señal de despertar al servicio si éste se está ejecutando.

- **Maxprocess**: número máximo de procesos que puede tener en ejecución el servicio. Por defecto, se admiten 50.

- **Command**: nombre del programa a ejecutar y parámetros a pasar. Todos los programas admiten las siguientes opciones:
  -v Habilita mayor detalle de log.
  -D Activa el modo debug.

Nosotros, para que funcionen los usuarios virtuales con MySQL, deberemos añadir el siguiente servicio manualmente:

```
virtual unix - n n - - virtual
```

6.2. MYSQL_VIRT.CF, IDS.CF Y GIDS.CF

Pasamos a continuación a mostrar el contenido de los ficheros antes referenciados en el main.cf llamados mysql_virt.cf, ids.cf y gids.cf. Todos ellos hacen referencia a una Base de Datos implementada en MySQL, que más adelante detallaremos.

mysql_virt.cf

- **user** y **password**: usuario y password para acceder al servidor MySQL:

```
user=postfix
password=postfix
```

- **dbname** y **table**: nombre de la Base de Datos y la tabla que contienen los datos de las cuentas de usuario:

```
dbname=mail
table=passwd
```

- **select_field** y **where_field**: en select_field indicamos que el campo de la base de datos que elegimos para recuperar el buzón del usuario es maildir. En where_field especificamos con el campo id de la tabla passwd la dirección de correo del destinatario del mensaje.

```
select_field=maildir
where_field=id
```

- **hosts**: el host que tiene el MySQL. En esta documentación, tanto Postfix como MySQL están en la misma máquina, por tanto, el primero se conectará a la base de datos por medio de los sockets internos de unix/linux. Para ello ponemos el parámetro hosts con el valor unix:mysqld.sock. Si estuviesen en máquinas distintas, habría que poner el nombre de la máquina en la que se encuentra MySQL.

```
hosts=unix:mysqld.sock
```

ids.cf

- **user** y **password**: usuario y password para acceder al servidor MySQL:

```
user=postfix
password=postfix
```

- **dbname** y **table**: nombre de la Base de Datos y la tabla que contienen los datos de las cuentas de usuario:

```
dbname=mail
table=passwd
```

- **select_field** y **where_field**: en select_field indicamos que el campo de la base de datos que elegimos para recuperar el uid del usuario es uid. En where_field especificamos con el campo id de la tabla passwd la dirección de correo del destinatario del mensaje.

```
select_field=uid
where_field=id
```

- **hosts**: el host que tiene el MySQL.

```
hosts=unix:mysqld.sock
```

gids.cf

- **user** y **password**: usuario y password para acceder al servidor MySQL:

```
user=postfix
password=postfix
```

- **dbname** y **table**: nombre de la Base de Datos y la tabla que contienen los datos de las cuentas de usuario:

```
dbname=mail
table=passwd
```

- **select_field** y **where_field**: en select_field indicamos que el campo de la base de datos que elegimos para recuperar el gid del usuario es gid. En where_field especificamos con el campo id de la tabla passwd la dirección de correo del destinatario del mensaje.

```
select_field=gid
where_field=id
```

- **hosts**: el host que tiene el MySQL.

```
hosts=unix:mysqld.sock
```

Deberemos cambiar los permisos de estos ficheros para que sólo el grupo postfix pueda leerlos puesto que contienen las contraseñas para la conexión a la BD.

7. INSTALACIÓN Y CONFIGURACIÓN DE MYSQL

Instalaremos los paquetes de MySQL de los repositorios de Debian con:

```
#apt-get install mysql-server
```

La primera medida a tomar para preservar la seguridad de nuestras bases de datos es poner contraseña al usuario root puesto que su contraseña por defecto está en blanco.

```
#mysqladmin password Contraseña
```

No deberemos olvidar tampoco habilitar el archivo de log (que por defecto no está indicado). Para esto, descomentaremos en /etc/mysql/my.cnf la siguiente línea:

```
log = /var/log/mysql/mysql.log
```

Ahora deberemos crear la Base de Datos y la tabla necesaria para guardar toda la información de nuestros usuarios virtuales.

Nos conectamos como root indicando con el parámetro –h el nombre del host en el que reside la Base de Datos, con –u el usuario y con –p le decimos que vamos a introducir contraseña.

```
#mysql -h localhost -u root -p
```

Una vez dentro de MySQL, creamos una Base de Datos llamada mail:

```
mysql>create database mail;
```

Seleccionamos la Base de Datos recién creada para trabajar con ella:

```
mysql>use mail;
```

Creamos la tabla en la que guardaremos la información de los usuarios virtuales.

```
mysql> create table passwd (
id char(128) DEFAULT ' ' NOT NULL,
pass char(128) DEFAULT ' ' NOT NULL,
name char(128) DEFAULT ' ' NOT NULL,
uid int(10) unsigned NOT NULL,
gid int(10) unsigned NOT NULL,
home char(255) DEFAULT ' ' NOT NULL,
maildir char(255) DEFAULT ' ' NOT NULL,
KEY id (id(128))
);
```

Esta tabla tiene los siguientes campos:

PASSWD		
id	char(128)	Dirección de correo completa del usuario. **Primary Key**
pass	char(128)	Contraseña del usuario para autentificarse
name	char(128)	Login del usuario (nombre hasta la @)
uid	int(10)	uid del usuario. Iremos asignándole números secuencialmente a partir del primer userid que no esté ocupado
gid	int(10)	gid del grupo mail (lo miramos en /etc/group) para que puedan escribir en el directorio /var/spool/mail que tiene como grupo propietario a mail
home	char(255)	home que por ser usuarios "virtuales" lo dejamos a "/" mismo
maildir	char(255)	Lugar donde se almacenará el buzón de correo. A esta ruta se le añade por delante el valor que habíamos puesto en /etc/postfix/main.cf como virtual_mailbox_base

También debemos crear un usuario postfix y darle todos los permisos en esta tabla:

```
mysql>GRANT ALL ON mail.passwd TO postfix@localhost IDENTIFIED BY
"postfix";
```

Ahora ya podemos ir introduciendo los datos de nuestros usuarios. Ejemplo:

```
mysql>insert into passwd (id,pass,name,uid,gid,home,maildir)
values("postmaster@prueba.com","contrasenia","postmaster","1003","8","
/","/var/spool/mail/postmaster/");
```

Como indicamos en el campo home de la Base de Datos, deberemos crear la ruta para almacenar el correo de nuestros usuarios (/var/spool/mail/). Dentro de esta carpeta Postfix creará los buzones de manera automática (cuando llegue el primer mensaje para un usuario, será el propio demonio el encargado de recrear la jerarquía).

También tenemos el comando *maildirmake* para crear buzones. Este comando se encarga de crear la estructura interna (subdirectorio new, cur y tmp). Ejemplos de uso de este comando:

- Creación de un buzón con quota:
  ```
  maildirmake -q quota
  ```
- Creación de una carpeta dentro del propio buzón:
  ```
  maildirmake -f carpeta
  ```

Después de esto debemos hacer que el grupo mail y el usuario postfix sean los dueños de los directorios y darles permisos de escritura, lectura y ejecución.

Resumiendo, el proceso es el siguiente: cuando mandamos un mail por SMTP a un usuario, Postfix lo recibe y lo primero que hace es ver si dicho usuario existe en la base de datos de MySQL. Para ello, se conecta a la BD y lanza una consulta SQL. Si no obtiene resultado, es que dicho user no existe, por lo que no es posible enviar el mail. Si la consulta tiene éxito, coge el campo maildir cuyo id corresponde con el destinatario del mail, y mira a ver si existe el directorio Maildir en /var/spool/mail/. Si no es así lo crea y deposita el mail en dicho directorio, estando ya a disposición del usuario para que lo consulte por IMAP o POP a través de Courier.

Existe un problema al usar MySQL con Postfix. Postfix corre enjaulado como chroot en /var/spool/postfix. Para acceder a la base de datos, lo hace a través de un socket (por defecto /var/run/mysqld/mysqld.sock) pero éste está fuera del entorno chroot en el cual se ejecuta Postfix (por defecto /var/spool/postfix). Entonces no tenemos acceso a dicho socket. La solución consiste en crear el directorio dentro del chroot:

```
#mkdir -p /var/spool/postfix/var/run/mysqld
#chown mysql /var/spool/postfix/var/run/mysqld
```

Y crear un hard link al socket actual:

```
#ln /var/run/mysqld/mysqld.sock /var/spool/postfix/var/run/mysqld/mysqld.sock
```

Vamos a probar la conexión, para asegurarnos de que está bien hecho:

```
# mysql -p -S /var/spool/postfix/var/run/mysqld/mysqld.sock
```

Una vez que tenemos montada la Base de Datos e instalado Postfix, ya podemos arrancarlos para tener un MTA totalmente funcional:

```
#/etc/init.d/mysql start
#/etc/init.d/postfix start
```

Comprobamos que tenemos el demonio de Postfix escuchando y los procesos ejecutándose:

```
#netstat –an | grep :25 | grep LISTEN

tcp        0      0 0.0.0.0:25        0.0.0.0:*               LISTEN

#ps aux | grep postfix

root     9963  0.0 0.3  3656 1328 ?  Ss  Sep25  0:02 /usr/lib/postfix/master

postfix 9967  0.0 0.3  2996 1212 ?  S   Sep25  0:00 qmgr -l -t fifo -u -c

postfix 22046 0.0 0.2  2964 1096 ? S   01:12  0:00 pickup -l -t fifo -u -c

postfix 22060 0.0 0.7  5416 2844 ? S   01:31  0:00 smtpd -n smtp -t inet -u -c

postfix 22061 0.0 0.3 3612 1396 ? S 01:31  0:00 trivial-rewrite -n rewrite -t unix -u –c
```

Miramos también los logs en busca de posibles fallos. Los logs de Postfix son los siguientes:

- **mail.log** (/var/log/mail.log): fichero donde se registran todas las acciones del servidor Postfix.
- **mail.err** (/var/log/mail.err): fichero donde se registran los errores.
- **mail.info** (/var/log/mail.info): fichero donde se registran los eventos que no son errores.

Un comando fácil para buscar problemas es el siguiente:

```
#egrep '(reject|warning|error|fatal|panic):' /var/log/mail.log
```

Otros comandos que nos ofrece Postfix para administrar nuestro servidor son:

- **mailq**: lista el contenido de la cola de correo. Muestra el ID de cola, tamaño del correo, hora llegada, remitente y destinatario/s. Si no se ha podido entregar, muestra la razón.
- **postsuper**: se encarga de realizar operaciones de mantenimiento.
- **postqueue**: comando que sirve de interfaz para la gestión de las colas.
- **postmap**: crea, actualiza o consulta una o más tablas de postfix.
- **postconf**: muestra los valores actuales de los parámetros de postfix (los que se asignan en main.cf).
- **postfix check**: comprueba que la sintaxis de main.cf sea correcta.

Para más información de esos comandos, siempre está man ;-)

Aún faltará por añadir soportes que nos den más seguridad e instalar el Courier para que los usuarios puedan consultar su correo.

8. INSTALACIÓN Y CONFIGURACIÓN DE COURIER

Courier utiliza un servicio de autentificación común para todos sus servicios (POP3, POP3S, IMAP, IMAPS). Este servicio se puede configurar de manera que haga la autentificación desde varias fuentes (PAM, LDAP, MySQL, et.). Para instalarlo desde los respositorios de Debian:

```
#apt-get install courier-authdaemon courier-authmysql courier-base
```

El servicio de autentificación está formado por un demonio llamado *authdaemon*, cuyo fichero de configuración es /etc/courier/authdaemonrc. Por defecto viene configurado para la autentificación vía PAM así que debemos editar el archivo authdaemonrc para sustituir la línea `authmodulelist="authpam"` por `authmodulelist="authmysql"` para que a partir de ahora se use MySQL.

También hay que indicar a courier dónde debe mirar los usernames y passwords; para ello tenemos que modificar el archivo authmysqlrc y tener especial cuidado con los espacios que se dejan entre el nombre de la variable y el valor de la misma, puesto que esos espacios se tomarán como parte del valor a la hora de intentar conectarse a la base de datos (por ejemplo, si dejamos un espacio delante de la contraseña, tomará el valor de la contraseña con el espacio y no podrá acceder a la base de datos).

authmysqlrc

Usuario con el que conectaremos a la Base de Datos:

```
MYSQL_USERNAME postfix
```

Contraseña con la que conectaremos a la Base de Datos:

```
MYSQL_PASSWORD postfix
```

La dirección del servidor MySQL. Como en nuestro caso lo tenemos instalado en nuestra máquina, usaremos localhost:

```
MYSQL_SERVER localhost
```

Puerto de MySQL:

```
MYSQL_PORT 3306
```

Nombre de la Base de Datos:

```
MYSQL_DATABASE mail
```

Tabla de la BD donde están los usuarios, sus contraseñas, la ruta de su buzón de correo, etc:

```
MYSQL_USER_TABLE passwd
```

Le podemos indicar que la contraseña se guarda en texto claro con `MYSQL_CLEAR_PWFIELD pass` o bien cifrada. Nosotros optaremos por tener el MD5 de la contraseña en la BD por si alguien consigue acceder.

```
MYSQL_CRYPT_PWFIELD pass
```

Dominio por defecto que se agregará al login:

```
DEFAULT_DOMAIN prueba.com
```

El campo en la tabla passwd donde estarán los uid de usuarios:

```
MYSQL_UID_FIELD uid
```

El campo en la tabla passwd donde estarán los gid de usuarios:

```
MYSQL_GID_FIELD gid
```

El campo en la tabla passwd donde estarán las direcciones de usuarios:

```
MYSQL_LOGIN_FIELD id
```

El campo en la tabla passwd donde estarán los directorios home del usuario:

```
MYSQL_HOME_FIELD home
```

El campo en la tabla passwd donde estarán los logins de usuarios:

```
MYSQL_NAME_FIELD name
```

El campo en la tabla passwd donde se almacena la ruta del buzón de correo:

```
MYSQL_MAILDIR_FIELD maildir
```

8.1. COURIER POP3 Y COURIER POP3S

El servicio POP3 lo aporta el paquete *courier-pop* y POP3S el paquete courier-pop-ssl, por lo que a de instalarse en el sistema:

```
#apt-get install courier-pop courier-pop-ssl
```

Sus ficheros de configuración residen en /etc/courier/ con el nombre de pop3d y pop3d-ssl,

Pop3d

Pid del demonio:

```
PIDFILE=/var/run/courier/pop3d.pid
```

Número máximo de demonios arrancados:

```
MAXDAEMONS=40
```

Número máximo de conexiones por IP:

```
MAXPERIP=4
```

Con esta opción configuraremos el nivel de debug. Para tenerlo activado lo ponemos a 1 ó 2 (muestra usuario y contraseña en el log). Una vez hechas las pruebas no deberemos olvidar ponerlo a 0 de nuevo para no dejar información delicada en el log.

```
DEBUG LOGIN=0
```

Puerto en el que escuchará el servicio POP:

```
PORT=110
```

Para activar el servidor POP3 de Courier sin necesidad de editar los scripts de arranque del sistema:

```
POP3DSTART=YES
```

Pop3d-ssl

Puerto en el que escuchará el POP3S:

```
SSLPORT=995
```

Ruta al certificado SSL del servidor POP3. Más adelante se explica cómo generar este certificado (*véase Anexo A*):

```
TLS CERTFILE=/etc/courier/pop3d.pem
```

Para activar el servidor POP3S de Courier:

```
POP3DSLLSTART=YES
```

Si queremos que forzar el uso de TLS a nuestros usuarios deberemos poner la variable *POP3_TLS_REQUIRED* a 1:

```
POP3_TLS_REQUIRED=1
```

8.2. COURIER IMAP Y COURIER IMAPS

El servicio IMAP lo aporta el paquete *courier-imap* y IMAPS el paquete *courier-imap-ssl*, por lo que a de instalarse en el sistema desde los repositorios:

```
#apt-get install courier-imap courier-imap-ssl
```

Sus ficheros de configuración residen en /etc/courier/ con el nombre de imapd e imapd-ssl.

Imapd

Puerto en el que escuchará el servicio IMAP:

```
PORT=143
```

Número máximo de demonios arrancados:

```
MAXDAEMONS=40
```

Número máximo de conexiones por IP:

```
MAXPERIP=20
```

Para arrancar el servicio IMAP sin tener que editar los scripts de inicio del nivel de ejecución:

```
IMAPDSTART=YES
```

Imapd-ssl

Ruta al certificado SSL del servidor IMAPS. Más adelante se explica cómo generar este certificado (*véase Anexo A*):

```
TLS_CERTFILE=/etc/courier/imapd.pem
```

Para arrancar el servicio IMAPS:

```
IMAPDSSLSTART=YES
```

Se puede generar un certificado de prueba con el comando *mkimapdcert*. Esto genera un certificado X.509 auto-firmado.

Una vez configurado tanto courier imap como courier pop, arrancamos sus respectivos demonios (también el que se encarga de la autentificación):

```
#/etc/init.d/courier-authdaemon start
#/etc/init.d/courier-pop start
#/etc/init.d/courier-pop-ssl start
#/etc/init.d/courier-imap start
#/etc/init.d/courier-imap-ssl start
```

9. POSTFIX-TLS

Comenzaremos ahora a agregar capas de seguridad a nuestro MTA, empezando por TLS.

Los mensajes que se envían desde un servidor a otro mediante SMTP viajan en texto claro por defecto. Esto hace que cualquier persona que pueda interponerse entre ambos servidores pueda leer el contenido del mensaje. Para evitar esta situación podemos recurrir al cifrado de la conexión mediante TLS.

Para ello, instalamos el soporte para tls en Postfix desde los repositorios:

```
# apt-get install postfix-tls
```

TLS se basa en el uso de certificados X.509, por lo que será necesario disponer de una clave privada y de un certificado firmado por alguna CA (Autoridad de Certificación). Normalmente, y evidentemente por razones de seguridad, la clave privada siempre se almacena protegida por una frase de acceso. Esto implica que cada vez que se accede a dicha clave será necesario proporcionar la frase.

En un servidor de correo esta circunstancia puede ser un inconveniente, por lo que será necesario disponer la clave privada sin protección de frase de acceso. Como contrapartida, es necesario tener un especial cuidado con los permisos del directorio y los ficheros relacionados.

Generamos el certificado y lo firmamos con nuestra CA (*véase Anexo A*).

```
#mkdir /etc/postfix/ssl
#cp demoCA/cacert.pem /etc/postfix/ssl/
#cp newcert.pem /etc/postfix/ssl/
#cp newreq.pem /etc/postfix/ssl/
#chown root /etc/postfix/ssl/newreq.pem
#chmod 400 /etc/postfix/ssl/newreq.pem
```

Añadimos las siguientes líneas al fichero principal de Postfix, main.cf:

Usar TLS siempre que se pueda:

```
smtpd_use_tls = yes
```

Aviso de conexión TLS:

```
smtpd_tls_note_starttls_offer = yes
```

Lugar del certificado:

```
smtpd_tls_cert_file = /etc/postfix/ssl/newcert.pem
```

Lugar de la solicitud de firma:

```
smtpd_tls_key_file = /etc/postfix/ssl/newreq.pem
```

Lugar del certificado de la CA:

```
smtpd_tls_CAfile = /etc/cert/private/cacert.pem
```

Nivel de log:

```
smtpd_tls_loglevel = 1
```

Tiempo de validez de las claves:

```
smtpd_tls_session_cache_timeout = 3600s
```

Fuente de entropía:

```
tls_random_source = dev:/dev/urandom
```

10.POSTFIX-SASL

SASL (Simple Authentication and Security Layer) es una capa software que permite añadir soporte de autentificación en protocolos orientados a conexión, como puede ser SMTP.

SMTP AUTH es una solución donde el cliente se identifica mediante un Usuario y Password ante el servidor SMTP, y si ésta se realiza correctamente se autoriza el uso del servidor como relay. La autorización se hace a la persona y no al sistema informático que en ese momento le proporciona soporte.

Gracias a la combinación que hacemos de SASL con TLS, logramos que al cifrar la conexión con TLS desde el inicio, nuestra contraseña que va en claro por la red no pueda ser interceptada por nadie.

Instalamos las librerías para implementar la API de SASL con:

```
#apt-get install libsasl2 libsasl2-modules
```

Instalamos también las herramientas para administración de usuarios:

```
#apt-get sasl2-bin
```

Añadimos las siguientes líneas al **main.cf**:

- El parámetro *smtpd_sasl_auth_enable* activa la utilización del protocolo SMTP AUTH.

```
smtpd_sasl_auth_enable = yes
```

- La directiva *broken_sasl_auth_clients* se utiliza para dar soporte a antiguos clientes Microsoft que no soportan la versión estándar del protocolo AUTH.

```
broken_sasl_auth_clients = yes
```

- Con la directiva *smtpd_sasl_local_domain* se define el dominio de búsqueda, que en terminología SASL se denomina realm. Este campo, junto con el nombre de usuario, es lo que se utilizará para realizar la búsqueda en el fichero /etc/sasldb. Es por ello muy importante que cuando se creen los usuarios con las herramientas que proporciona SASL se especifique correctamente el realm, pues de lo contrario puede que las búsquedas sean fallidas (en /etc/sasldb2 con usuario@mail.prueba.com).

```
smtpd_sasl_local_domain = $myhostname
```

- En *smtpd_recipient_restrictions* se le indica a Postfix las restricciones que tienen que cumplir los mensajes para que sean aceptados. Las restricciones se analizan secuencialmente, y se detiene el proceso cuando una concuerda. Con la opción *permit_mynetworks* le indicamos que sólo recoja los mensajes procedentes de:

- Clientes cuya dirección IP esté incluida en *mynetworks*.
- Clientes que se hayan identificado correctamente mediante SMTP AUTH.
- Cualquier sistema, cuyo destinatario esté incluido en *relay_domains*, *inet_interfaces*, *mydestination*, *virtual_alias_domains*, *virtual_mailbox_domains*.

```
smtpd_recipient_restrictions =
      permit_mynetworks,
      permit_sasl_authenticated,
      reject_unauth_destination
```

- El mecanismo de autenticación permitido por Postfix es el no anónimo:

```
smtpd_sasl_security_options = noanonymous
```

Existen varios métodos para gestionar las contraseñas:
- saslauth: demonio Cyrus SASL contra cuentas UNIX.
- auxprop: archivo independiente de usuarios y contraseñas.

Nosotros escogemos auxprop y para ello se lo indicamos a Postfix en el archivo smtpd.conf (/etc/postfix/sasl/smtpd.conf):

```
pwcheck_method: auxprop
mech_list: plain login
```

Aquí también le hemos indicado que la contraseña viajará en claro por la red pero esto no supone ningún problema porque tenemos soporte TLS para cifrar la conexión.

Se va a utilizar únicamente el módulo sasldb, y por tanto no es necesario ejecutar el demonio de autentificación. Por tanto, pararemos el demonio de Cyrus saslauthd de la siguiente forma: editando /etc/default/saslauthd y poniendo:

```
START=no
MECHANISMS=”sasldb”
```

La base de datos de usuarios residirá en el fichero /etc/sasldb2. Es importante que este fichero tenga permisos de lectura para todos y de escritura para el grupo sasl. Para ello tendremos que ejecutar:

```
#chmod 664 /etc/sasldb2
```

Para añadir/eliminar los usuarios a la base de datos se utilizará el programa saslpasswd2:

```
#saslpasswd2 -f /etc/sasldb2 -c –u mail.prueba.com usuario
 Password:
 Again (for verification):
```

Para ver el usuario recién creado:

```
#sasldblistusers2 /etc/sasldb2
```

Como Postfix está trabajando en modo chroot es necesario enlazar el archivo de contraseñas al correspondiente en el chroot. Para ello creamos un enlace fuerte:

```
#ln /etc/sasldb2 /var/spool/postfix/etc/sasldb2
```

Una vez realizadas las modificaciones en main.cf se hará que Postfix recargue la nueva configuración mediante la directiva:

```
#/etc/init.d/postfix reload
```

Si luego se establece una conexión telnet al puerto SMTP se obtendrá lo siguiente:

```
220 mail.prueba.com ESMTP

ehlo mail.deusto.es

250-mail.prueba.com

250-PIPELINING

250-SIZE 10485760

250-ETRN

250-STARTTLS

250-AUTH LOGIN PLAIN

250-AUTH=LOGIN PLAIN

250 8BITMIME
```

Se han agregado dos nuevas funcionalidades: AUTH LOGIN, y AUTH=LOGIN (para los viejos clientes de Microsoft).

11.SPAM

El SPAM consiste en el envío de mensajes no solicitados, habitualmente de tipo comercial, remitidos en cantidades masivas. También es conocido como UBE (Unsolicited Bulk Email) o UCE (Unsolicited Commercial Email).

No sólo tenemos que tener en consideración el spam que reciben nuestros usuarios a sus buzones sino también el tener bien configurado nuestro servidor para que no se convierta en un emisor (si está como open relay).

En este ejemplo vemos como nuestro servidor de correo no permite enviar mensajes a una dirección que no sea suya (deusto.es), es decir, no está como open relay:

```
telnet mail.prueba.com 25
220 mail.prueba.com ESMTP
ehlo mail.deusto.es
250-mail.prueba.com
250-PIPELINING
250-SIZE 10485760
250-ETRN
250-STARTTLS
250-AUTH LOGIN PLAIN
250-AUTH=LOGIN PLAIN
250 8BITMIME
mail from: pepito@deusto.es
250 Ok
rcpt to: jaimito@hotmail.com
554 <jaimito@hotmail.com>: Relay access denied
```

Todo software antispam se puede equivocar y caer en falsos negativos o falsos positivos:

- **Falsos negativos**: es cuando el antispam no es capaz de detectar un correo no solicitado.
- **Falsos positivos**: es cuando un correo legítimo es clasificado como spam. Estos son los más graves.

Un buen antispam es el que no tiene falsos positivos y muy pocos falsos negativos.

Para evitar el spam podemos llevar a cabo las siguientes acciones:

- Usar listas negras.
- Poner restricciones en la conexión de los clientes con el servidor: *smtpd_client_restrictions*.
- Controlar el saludo del cliente o servidor que nos entrega el mensaje: *smtpd_helo_restrictions*.
- Controlar la orden MAIL FROM: *smtpd_sender_restrictions*.

11.1. LISTAS NEGRAS

Para detectar el spam podemos hacer uso de las RBL (Realtime Blackhole List): listas negras en tiempo real. En éstas se almacenan las direcciones ip o los nombres de posibles emisores de spam. El funcionamiento es el siguiente: nuestro servidor, cuando un cliente u otro servidor se conecta a él, comprueba que no está en una lista negra. Para ello, puede hacer las comprobaciones por dirección ip o por nombre (RHSbl, DNSbl).

Hay que tener cuidado con ellas puesto que si no están actualizadas pueden provocar pérdidas de correos.

Listas actualizadas constantemente y recomendadas para su uso:

- cbl.abuseat.org → http://cbl.abuseat.org/
- dul.dnsbl.sorbs.net → http://www.dnsbl.nl.sorbs.net/
- sbl+xbl de spamhaus → http://www.spamhaus.org/xbl/
- list.dsbl.org → http://dsbl.org/
- combined.rbl.msrbl.net → http://www.msrbl.com/
- bl.spamcop.net → http://www.spamcop.net/

Postfix permite fácilmente verificar que el cliente que ha establecido la conexión SMTP o el remitente del mensaje no está inscrito en alguna de las listas negras que se designen.

Para ello, en **main.cf**, en la directiva *smtpd_client_restrictions*, activaremos el mecanismo de verificación:

```
smtpd_client_restrictions =

      reject_rbl_client relays.ordb.org

      reject_rhsbl_client relays.ordb.org
```

De esta forma, Postfix verificará que la traducción inversa de la dirección IP del cliente (reject_rbl_client) o el nombre de host del cliente (reject_rhsbl_client) no aparece inscrito en la RBL relays.ordb.org.

Este tipo de restricción también es aplicable en el comando MAIL FROM a través de la directiva *smtpd_sender_restrictions*:

```
smtpd_sender_restrictions =

      reject_rhsbl_sender relays.ordb.org
```

11.2. RESTRICCIONES ORIENTADAS A LA CONEXIÓN DEL CLIENTE

Postfix permite delimitar los clientes a los que permitirá establecer una sesión SMTP. Para ello dispone de la directiva *smtpd_client_restrictions*, que entre otros criterios, permite rechazar las conexiones desde clientes cuya dirección IP no disponga de resolución inversa en el DNS. Esta medida ayuda a frenar la posible entrada de SPAM.

```
smtpd_client_restrictions = reject_unknown_client
```

11.3. RESTRICCIONES ORIENTADAS AL SALUDO INICIAL

Normalmente los sistemas de gestión de correo no obligan a iniciar el establecimiento de sesión mediante la directiva HELO o EHLO en caso de ESMTP. Es más, tampoco suelen proporcionar mecanismos que permitan analizar el nombre del sistema que está identificándose.

Postfix dispone de la directiva *smtpd_helo_required*, que dándole el valor yes obliga a que el sistema remoto inicie la sesión con la directiva HELO o EHLO. Por defecto este parámetro está a no. Obligar a iniciar la sesión con HELO/EHLO ya en sí inutiliza algunas herramientas utilizadas para SPAM y algún que otro virus.

```
smtpd_helo_required = yes
```

En conjunción con la anterior directiva se dispone también de *smtpd_helo_restrictions*, que posibilita establecer criterios que permiten continuar o no con la sesión. Aunque existen hasta doce criterios aplicables, la configuración básica recomendada es:

```
smtpd_helo_restrictions =
      reject_invalid_hostname
      reject_unknown_hostname
      reject_non_fqdn_hostname
```

Mediante el criterio *reject_invalid_hostname* se rechazarán todas las sesiones en las que el nombre proporcionado en la directiva HELO/EHLO esté mal formado.

Con el criterio *reject_unknown_hostname* se rechazarán todas las sesiones en las que el nombre del sistema proporcionado no tenga una entrada A en un DNS o disponga de un registro MX.

Y por último, con el criterio *reject_non_fqdn_hostname* se rechazarán todas las sesiones en las que el nombre del sistema indicado en la directiva HELO/EHLO no esté en forma FQDN, es decir, que sea un nombre completo.

11.4. RESTRICCIONES ORIENTADAS AL REMITENTE

La comprobación *reject_unknown_sender_domain* se basa en verificar que, si nos envían un correo desde, por ejemplo pepito@deusto.es, deusto.es tiene un registro A o MX asociado. Si no lo tiene se rechazará el correo.

```
reject_unknown_sender_domain
```

Con *reject_non_fqdn_sender* se exigirá que el dominio del servidor que nos envía el correo sea FQDN. Por ejemplo, se rechazaría un correo de pepito@deusto pero se aceptaría de pepito@deusto.es

```
reject_non_fqdn_sender
```

12.FILTROS DE CORREO

El propio Postfix implementa filtros de correo que se pueden aplicar a las cabeceras y a los cuerpos de los mensajes.

Las opciones a incluir en el fichero main.cf son **header_checks** y **body_checks**. Como su nombre indica, header_checks filtrará mediante expresiones regulares (tablas de tipo regexp:/) las cabeceras del mensaje, y body_checks se ocupará de filtrar el cuerpo. Esto no debe usarse como antispam ni como antivirus, sino como un método adicional de protección. Sabemos que muchos gusanos que se envían por correo electrónico suelen usar el mismo asunto en el mensaje, o palabras clave en el cuerpo del mensaje, de modo que podríamos utilizar esto para filtrar esos mensajes no deseados.

12.1. FILTRADO POR CABECERAS

La posibilidad de filtrar los mensajes en base a las cabeceras de los mismos y a patrones definidos mediante expresiones regulares logra combatir un gran número de virus y spam.

De esta manera, todo mensaje que contenga una cabecera que cumpla un determinado patrón será rechazado automáticamente. Un ejemplo de fichero de patrones podría ser:

```
/^Subject:.*VIAGRA.*/          REJECT
/^Subject: ILOVEYOU/           REJECT
/^From:.*eresalud.com/         REJECT
```

En él, se filtran mensajes por campo asunto (evitando el spam de viagra y el virus ILOVEYOU) y por campo from.

Suponiendo que este fichero es /etc/postfix/headers_checks, para activar el filtrado por cabeceras será necesario incluir en el fichero main.cf la siguiente directiva:

```
header_checks = regexp:/etc/postfix/header_checks
```

Hacemos un reload de Postfix para que recargue la nueva configuración y ya lo tenemos funcionando.

12.2. FILTRADO POR CONTENIDO

De igual manera que se pueden filtrar mensajes en base a patrones que se definen sobre las cabeceras de los mensajes, también se pueden definir patrones que se aplicarán al contenido de los mensajes.

```
/Any problems with ere[jc][jc]tion\? (try|Get) V[a-zI][a-zI]AGRA/ REJECT
```

En este ejemplo se rechazan todos los mensajes que contengan en cualquier parte del cuerpo alguna de las palabras o frases indicadas. Suponiendo que este fichero es /etc/postfix/body_checks, para activar el filtrado por contenidos en el mensaje será necesario incluir en el fichero main.cf la siguiente directiva:

```
body_checks = regexp:/etc/postfix/body_checks
```

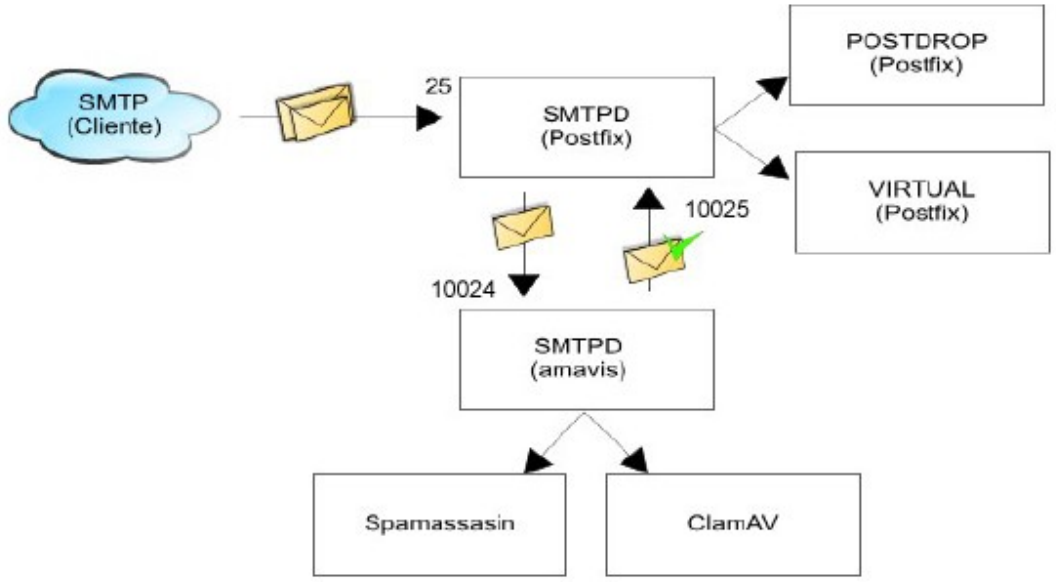
13.AMAVISD-NEW

Amavisd-new es una aplicación basada en un script de Perl flexible y de alto rendimiento que se ejecuta como un servicio, con un proceso maestro y otro hijo.

Sirve de interfaz de filtrado entre un MTA y otras aplicaciones (por ejemplo, un antivirus o un antispam) actuando como un servidor SMTP, recibiendo el mensaje de correo del servidor Postfix, procesándolo y enviándolo o devolviéndolo al servidor SMTP.

Acompañaremos a AMaVIS de dos aplicaciones auxiliares: ClamAV para el filtro de correo con virus y Spamassassin para el filtrado de correo no deseado.

El funcionamiento de filtrado es el siguiente: un correo ha de llegar a nuestro puerto 25 (Postfix). Este será enviado al puerto 10024 (por ejemplo) de AMaVIS, y AMaVIS una vez analizado, nos lo enviaría de nuevo a otro puerto (el 10025 por ejemplo) de Postfix sin restricción alguna entre ellos. El porqué de que se envíe a otro puerto es que si AMaVIS devuelve el correo analizado a Postfix por el 25, Postfix volverá a entregárselo y entraría en un bucle infinito. Si hacemos que Postfix reciba correo de AMaVIS en otro puerto, el problema se soluciona.



Procederemos a instalar AMaVIS de los repositorios con el siguiente comando:

```
# apt-get install amavisd-new
```

Lo primero es añadir al **master.cf** un nuevo servicio de smtpd secundario escuchando en el puerto 10025.

```
smtp-amavis unix -      -       -     -       8  smtp
            -o smtp_data_done_timeout=1200
            -o disable_dns_lookups=yes
127.0.0.1:10025 inet n  -       -     -       -  smtpd
            -o content_filter=
            -o local_recipient_maps=
            -o relay_recipient_maps=
            -o smtpd_restriction_classes=
            -o smtpd_client_restrictions=
            -o smtpd_helo_restrictions=
            -o smtpd_sender_restrictions=
            -o smtpd_recipient_restrictions=permit_mynetworks,reject
            -o mynetworks=127.0.0.0/8
            -o strict_rfc821_envelopes=yes
```

Lo que estamos consiguiendo con esto es poner a la escucha en localhost (sólo podremos acceder por bucle local), en el puerto 10025.

Editamos el fichero main.cf para añadir la opción que hace que los correos que lleguen se reenvíen al AMaVIS.

```
content_filter = smtp:[127.0.0.1]:10024
```

Ahora los correos llegarán a Postfix y serán, una vez aceptados, enviados al puerto 10024. Después, cuando AMaVIS analice el correo, lo enviará al puerto 10025. Para esto tenemos que editar el fichero de configuración de AMaVIS (/etc/amavis/amavisd.conf) y añadir la siguiente línea:

```
$forward method = 'smtp:127.0.0.1:10025';
```

Así le decimos a AMaVIS que reinyecte los mensajes a Postfix usando smtp en localhost por el puerto 10025.

Otras opciones que nos interesan dentro del fichero amavisd.conf son:

```
$sa_tag2_level_deflt = 6.3;
$sa_kill_level_deflt = $sa_tag2_level_deflt;
```

Esto nos indica el nivel en el cual un correo es considerado como SPAM. Esto lo analizaremos con más atención en el apartado de SpamAssassin.

AMaVIS, usando las aplicaciones como ClamAV o Spammassassin, puede detectar si el contenido del mensaje es deseado o no. Con las siguientes opciones le indicaremos qué hacer con los mensajes no deseados:

```
$final_virus_destiny = D_DISCARD;
$final_spam_destiny = D_PASS;
```

Con *$final_spam_destiny* se indica qué hacer en caso de que el correo sea detectado como SPAM y con *$final_virus_destiny*, la acción a tomar con un mensaje contenedor de virus. Si es un virus, lo más recomendable es descartarlo y si es SPAM, dejarlo pasar puesto que el correo ya tendría la cabecera `X-SPAM-Flag: Yes.` De esta manera dejamos que el usuario tome la decisión de qué hacer con el mensaje y evitamos pérdidas de mensajes por falsos positivos (los clientes de correo detectan que el mensaje tiene el flag de SPAM activado y lo marca como tal).

Reiniciamos entonces AMaVIS y Postfix para que carguen la nueva configuración:

```
# /etc/initd.d/amavis restart
# /etc/initd.d/postfix restart
```

Con esto ya está todo preparado. Postfix escuchando en el puerto 25, Amavis en el 10024 y una segunda instancia de Postfix en el 10025.

14.CLAMAV

ClamAV es otra herramienta utilizada en la inspección de mensajes electrónicos que permite identificar si el contenido del correo es un virus. AMaVIS le pasará los mensajes para que los inspeccione y luego ClamAV los devolverá.

Para instalarlo desde los repositorios:

```
#apt-get install clamav clamav-daemon
```

Tras instalar nos preguntará como queremos actualizar la base de datos antivirus de ClamAV. Seleccionamos el método cron.

A continuación nos pregunta cual será el servidor con el que sincronizar la base de datos. Es conveniente usar uno próximo a nosotros, por lo que escogeremos el de España `db.es.clamav.net` (España).

Si usamos proxy, debemos meter su configuración.

Por último nos pregunta si queremos recargar la base de datos cada vez que se actualizan las reglas. Contestaremos que no pues de eso se ocupará Amavis.

En este paso el ClamAV actualizará su base de datos automáticamente.

Para que todo se integre correctamente, ClamAV tiene que ejecutarse con un usuario del grupo amavis por lo que lo añadiremos:

```
# adduser clamav amavis
```

En el fichero de configuración de ClamAV (/etc/clamav/clamd.conf) se puede especificar si queremos escanear el correo entero o solamente los adjuntos con las opciones *ScanMail* y *ScanArchive*. Las comentamos o descomentamos a nuestro gusto.

También instalaremos descompresores para que ClamAV pueda analizar los archivos comprimidos:

```
# apt-get install lha arj unrar zoo nomarch lzop arc unzoo
```

Reiniciamos entonces AMaVIS y ClamAV:

```
#/etc/init.d/clamav-daemon restart
#/etc/init.d/amavis restart
```

Dentro del paquete de ClamAV también se incluye la herramienta clamscan que permite escanear ficheros desde la línea de comandos en busca de virus.

```
#clamscan [opciones] [archivo/directorio/-]
```

Escanea directorios de forma recursiva:

```
#clamscan –r
```

Muestra sólo los archivos infectados:

```
#clamscan –i
```

Guarda el resultado del escaneo en el archivo indicado:

```
#clamscan –l ARCHIVO
```

15.SPAMASSASSIN

El spamassassin es un filtro anti-SPAM que funciona en la parte servidora y realiza un análisis de cada mensaje que le pasa AMaVIS, aplicando una serie de reglas que van puntuando el nivel de SPAM. Consiste en un mecanismo heurístico basado en reglas y ponderaciones predefinidas incorporados en un algoritmo bayesiano. Además los usuarios pueden entrenarle para que aprenda.

Para proceder a instalarlo:

```
#apt-get install spamassassin
```

Para que AMaVIS haga uso de Spamassassin, en el archivo de configuración de AMaVIS se a de comentar la línea:

```
@bypass_spam_checks_acl = qw( . );
```

Ahora procederemos a configurar el filtro de correo Spamassassin. Como lo usaremos con AMaVIS, editaremos los siguientes ficheros de configuración:

- El de Spamassassin: etc/spamassassin/local.cf
- El de AMaVIS: /etc/amavis/amavisd.conf.

Debemos tener claro que las opciones de *amavisd.conf* prevalecerán sobre las de *local.cf*.

Agregamos estas opciones en **amavisd.conf**:

$sa_local_tests_only = 0;
Lo desactivaremos para que haga todo tipo de restricciones, requieran conexión a Internet o no. Es útil si usásemos listas negras en SpamAssassin.

$sa_timeout = 30;
Tiempo para llamar a SpamAssassin. Cuando se realiza algún tipo de consulta a Internet puede que el servidor no responda, así que podemos establecer un tiempo antes de dar el error de “Tiempo de espera agotado”.

$sa_mail_body_size_limit = 150*1024;
Establece el tamaño máximo en bytes del cuerpo de un mensaje para que no sea analizado. Como dice el fichero de configuración, solamente uno de cada 100 mensajes de SPAM sobrepasa los 64 KB. Le hemos asignado 150*1024 bytes, 150 KB. La verdadera utilidad es no consumir recursos si el mensaje es muy grande pues podemos estar casi seguros de que no será SPAM.

$sa_tag_level_deflt = -100;
Spamassassin añade unas cabeceras a los mensajes cuando el nivel de probabilidad de SPAM es superior a este valor. Es interesante tener siempre esas cabeceras presentes así que lo configuraremos a -100. Esto hará que siempre se agreguen cabeceras puesto que es casi imposible que el valor sea menor de –100.

$sa_tag2_level_deflt = 6.3;
Este es el nivel que determina si un mensaje se considera como SPAM o no. Para comprobar que el nivel que hemos establecido es el correcto, debemos probar enviando correos y recibiendo para observar si con cierto nivel tenemos falsos positivos.

Cuando Postfix recibe un mensaje se lo pasa a AMaVIS que a su vez se lo envía al demonio de SpamAssassin, quien procede a ejecutar una serie de comprobaciones sobre él.

Cada una de estas comprobaciones añade o quita unas pocas milésimas de puntuación a dicho correo. Si la puntuación que asigna es positiva, es que tiene cierto parecido con el spam; por contra, si esa puntuación es negativa, es que tiene parecido con correo válido.

Así, una vez pasados todos los análisis, se suman las puntuaciones para hallar la puntuación total de dicho correo. Si esa puntuación está por encima de *$sa_tag2_level_deflt*, el mensaje podrá ser marcado como spam. Si pasa por encima de *$sa_kill_level_deflt*, será directamente descartado.

$sa_kill_level_deflt = $sa_tag2_level_deflt;
Con esta opción configuraremos cual será el valor con el cual se rechazará el correo. Lo más común es que se rechace con el valor establecido en el parámetro anterior (pues con ese ya creemos que es SPAM).

$sa_dsn_cutoff_level = 10;
Cuando un correo es detectado como SPAM, un mensaje se envía a postmaster. Si sobrepasa este nivel, 10, no se enviará ese aviso y se rechazará sin más.

$sa_spam_subject_tag = '*SPAM***';**
Con esta opción añadiremos al asunto el texto que pongamos.

$sa_spam_modifies_subj = 1;
Sin esta opción activada, la anterior no funcionará, es decir, no se ponen cabeceras.

15.1. APRENDIZAJE EN LA CLASIFICACIÓN DE SPAM

SpamAssassin incluye utilidades para la gestión de su base de conocimiento sobre lo que es o no spam: principalmente sa-learn (SpamAssassin learn), que toma como entrada un archivo o una carpeta de archivos procedentes de correos electrónicos y los marca como spam. Es el entrenador de SpamAssassin.

```
#sa-learn [opciones] archivo/s de correo
```

Esas opciones pueden ser:

- --ham: aprende los correos del archivo como no spam.
- --spam: aprende los mensajes como spam.
- --forget: olvida el mensaje que le hicimos aprender.

- --dump [all | data | magic]: muestra los contenidos de la base de datos bayesana.
- --backup: hace un backup a la salida STDOUT de la base de datos.
- --restore <fichero>: recupera una base de datos del fichero.

ANEXO A: CERTIFICADOS

Si se pretende ofrecer servicios de correo electrónico, es más que recomendable usar un certificado X.509 que esté firmado por una autoridad certificadora (CA, del inglés *Certificate Authority*) reconocida, como Verisign, que garantice que un certificado pertenece a su legítimo propietario. Sin embargo, esto supone una importante suma de dinero.

Si sólo se va a usar en la red local de una empresa pequeña o mediana, o simplemente se quiere disfrutar de las comunicaciones cifradas a título personal, posiblemente sea suficiente con crear una autoridad certificadora nosotros mismos y firmar con ella los certificados. En cualquier caso, ambas soluciones son igual de seguras; es una cuestión de formalidad ante el cliente o usuario.

Un certificado X.509 es típicamente un archivo que contiene la siguiente información:

- Nombre Distintivo de la entidad. Incluye la información de identificación (el nombre distintivo) y la llave pública.
- Nombre Distintivo de la Autoridad Certificadora. Identificación y firma de la Autoridad Certificadora (CA) que firmó el certificado.
- Período de Validez. El período de tiempo durante el cual el certificado es válido.
- Información adicional. Puede contener información administrativa de la CA como un número de serie o versión.

Por tanto, los certificados están firmados electrónicamente por la Autoridad de Certificación utilizando su clave privada. Esto requiere de:

- Un par de claves, una pública y una privada (claves RSA o DSA, encriptación asimétrica). La clave pública es expuesta a todo el mundo y la privada es sólo conocida por el emisor. Con nuestra CA generaremos tanto la clave pública como la privada.
- Un certificado de seguridad, que es una versión "firmada" o verificada de la clave pública RSA o DSA.

A continuación, montaremos nuestra propia CA para firmar certificados.

Primero debemos instalar el paquete de herramientas OpenSSL de los repositorios:

```
#apt-get install openssl
```

OpenSSL se configura mediante el archivo */etc/ssl/openssl.cnf*, donde podemos especificar las carpetas donde queremos que se guarden los certificados y las claves e información que incluiremos en los certificados.

A. 1. CREACIÓN DE UNA AUTORIDAD CERTIFICADORA

OpenSSL proporciona un script en perl para crear la autoridad certificadora y los certificados: CA.pl. Con el parámetro *–newca* se generará:

```
#/usr/lib/ssl/misc/CA.pl -newca
```

Con este comando creamos una CA para certificados X.509. Una vez lanzado el comando, se nos pedirán una serie de datos:

Introducimos la PEM pass phrase dos veces.

El código de país con dos letras:

```
Country Name (2 letter code) [AU]: SP
```

Nombre de la provincia o estado:

```
State or Province Name (full name) [Some-State]: Bizkaia
```

Nombre de la ciudad:

```
Locality Name (eg, city) [ ]: Bilbao
```

Nombre de la organización:

```
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Empresa Loretahur, S.L.
```

La unidad organizativa:

```
Organizational Unit Name (eg, section) [ ]: Servicio Informático
```

Aquí va el nombre del host para el que creamos el certificado (en nuestro caso mail.prueba.com). Si no, el cliente de correo del usuario le indicará que puede que el certificado no sea válido.

```
Common Name (eg, YOUR name) [ ]: mail.prueba.com
```

Dirección del administrador del servidor:

```
Email Address [ ]: postmaster@prueba.com
```

Tras introducir todos los datos de la autoridad certificadora, se genera la clave privada (cakey.pem) y el certificado de la CA, al cual se remitirá al cliente cuando quiera comprobar la autenticidad del certificado que le ha enviado nuestro servidor Postfix (cacert.pem).

Una vez generada la clave privada, deberemos cambiarle los permisos para que sea de solo lectura únicamente para el propietario del fichero, puesto que la clave privada no debe ser obtenida por nadie.

```
#chown root cakey.pem
#chmod 400 cakey.pem
```

A. 2. CREACIÓN DE UN CERTIFICADO

Antes de hacer un certificado, hay que hacer una petición de firma donde se defina el propietario del certificado:

```
#/usr/lib/ssl/misc/CA.pl -newreq
```

Como en la creación de la CA, se introducen los datos del certificado (país, provincia, organización,... y la contraseña para el certificado).

Esto genera un fichero denominado newreq.pem que contiene la solicitud de firma del certificado y la clave privada cifrada.

A. 3. FIRMAR EL CERTIFICADO CON NUESTRA CA

Una vez que la petición se crea, podemos emplear la CA para firmarla.

```
#/usr/lib/ssl/misc/CA.pl -sign
```

Una vez lanzado el comando nos pedirá el password de la CA que lo emite y el fichero newcert.pem se generará. Newcert.pem es el certificado público que enviaremos al cliente para establecer la comunicación segura.

Ahora podemos renombrar los ficheros y moverlos a la carpeta de trabajo del servicio que necesita ese certificado.

```
#mv newcert.pem smtpd_cert.pem

#mv newreq.pem smtpd_key.pem
```

RECURSOS

- Página Oficial de Postfix: http://www.postfix.org
- Página Oficial de Courier: http://courier-mta.org
- Página Oficial de MySQL: http://mysql.com
- Página Oficial de AMaVIS: http://amavis.org
- Página Oficial de ClamAV: http://clamav.net
- Página Oficial de Spamassassin: http://spamassassin.apache.org/
- Página Oficial de Openssl http://www.openssl.org
- Página de RBL's: http://rbls.org
- Manual de creación de certificados y CA:

  http://www.openssl.org/docs/apps/CA.pl.html
- Postfix, Courier y MySQL. Álvaro Marín Illera:
  http://www.e-ghost.deusto.es/docs/articulo.postfixmysql.html
- Sistemas de Correo en GNU/Linux. Iker Sagasti Markina:
  http://documentacion.irontec.com/sistema_correo.pdf
- Postfix: The Definitive Guide, *Kyle D. Dent*, O'Reilly Diciembre 2003

LICENCIA

Reconocimiento-NoComercial-CompartirIgual 2.5 España

Usted es libre de:

- copiar, distribuir y comunicar públicamente la obra
- hacer obras derivadas

Bajo las condiciones siguientes:

Reconocimiento. Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador.

Compartir bajo la misma licencia. Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

No comercial. No puede utilizar esta obra para fines comerciales.

Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.

Alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor

Los derechos derivados de usos legítimos u otras limitaciones reconocidas por ley no se ven afectados por lo anterior.

Creative Commons