

HOWTO: ISP-STYLE EMAIL SERVER WITH DEBIAN-ETCH AND POSTFIX 2.3

- Title:** Howto: ISP-style Email Server with Debian-Etch and Postfix
- Authors:** Dipl.-Inform. Christoph Haas
- License:** (C) 2007 Christoph Haas. This document is published under the terms of the GNU General Public License (<http://www.gnu.org/licenses/gpl.html>)
- Source:** The most current version of this document can be found at workaround.org. If you feel you need to mirror this document then please keep your copy up to date. Otherwise you are causing the author a lot of headaches because readers stumble upon problems that have likely been dealt with in a newer version of this document.



[German translation](#)

Our sponsor:

Contents

- [Abstract](#)
- [Migrating from the Sarge Tutorial](#)
- [The Components](#)
- [Virtual Domains in a Database](#)
- [Virtual domains versus local domains](#)
 - [local domain](#)
 - [virtual alias domain](#)
 - [virtual mailbox domain](#)
- [Step 1: Installing the needed Debian packages](#)
- [Step 2: Create the database and user](#)
- [Step 3: Create the database tables](#)
- [Step 4: Create the database mapping files](#)
 - [virtual mailbox domains](#)
 - [virtual mailbox maps](#)
 - [virtual alias maps](#)
- [Step 5: Deliver emails through the Dovecot LDA](#)
- [Step 6: Configure Dovecot](#)
 - [/etc/dovecot/dovecot.conf](#)
- [Step 7: Send a test mail through TELNET](#)
 - [Emulating an SMTP session with telnet](#)

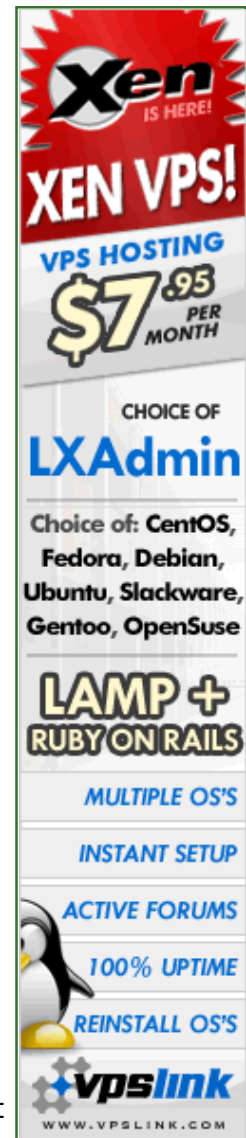
- [Checking the logs](#)
 - [Checking the user's mailbox](#)
- [Step 8: Test fetching emails with IMAP and POP3](#)
 - [Testing POP3](#)
 - [Test IMAP](#)
- [Step 9: Authenticated SMTP](#)
- [Step 10: AMaViS: Filtering spam and viruses](#)
- [Step 11: Learning spam and ham](#)
- [Step 12: Populate and administer the users in the database](#)
 - [Converting your database from the Sarge tutorial](#)
 - [Ronny Tiebel's PHP administration frontend](#)
 - [Peter Gutwein's PHP administration frontend](#)
- [Optional features](#)
 - [Offering webmail access](#)
 - [Sieve: Filtering out spam](#)
 - [Mailing lists with mailman](#)
 - [Using local aliases](#)
 - [Using transport maps \(contributed by Michael Dürchner\)](#)
 - [Using transport maps with a regular expression](#)
 - [Using transport maps with a database mapping](#)
 - [Greylisting with Postgrey](#)
 - [Vacation / Autoresponder](#)
 - [Fetching and filtering using fetchmail and sieve](#)
 - [Using Horde/Imp as a webmail interface](#)
 - [Removing old deleted mails](#)
- [Troubleshooting](#)
- [Thanks](#)

ABSTRACT

You surely know the big web hosters that allow you to rent a domain and use it to receive emails. If you have a Debian computer connected permanently to the internet you can offer the same service. You do not even need to have a fixed IP address thanks to dynamic DNS services like dyndns.org. All you need is this document, a pot of delicious tea and a little time. When you are done with the setup your server will be able to...

- receive and store emails for your users
- let your users retrieve the email through IMAP and POP3 - even with SSL to encrypt to connection
- receive and forward (relay) email for your users if they are authenticated
- offer a webmail interface to read emails in a web browser
- detect most spam emails and filter them out or tag them

This document is not a simple copy-and-paste tutorial. The intention is rather to make you understand the different components that you are using. There is deliberately no script that will do all the setup automatically. But in the end you will be skilled enough to debug problems yourself. If you feel you need help with your setup then read the hints on [Troubleshooting](#). The setup in this tutorial has been tested very thoroughly by several readers. Unlike many other Postfix tutorials on the internet this is already the third edition. Writing this tutorial took a month



Xen
IS HERE!

XEN VPS!

VPS HOSTING
\$7.95
PER MONTH

CHOICE OF
LXAdmin

Choice of: CentOS,
Fedora, Debian,
Ubuntu, Slackware,
Gentoo, OpenSuse

**LAMP +
RUBY ON RAILS**

MULTIPLE OS'S

INSTANT SETUP

ACTIVE FORUMS

100% UPTIME

REINSTALL OS'S

vpslink
WWW.VPSLINK.COM

of work so these are not just quick draft notes thrown together but a consistent document guiding you.

The configuration described here is not very complicated but still needs to be done carefully. You are expected to have at least basic knowledge of:

- MySQL (creating a database, granting access to users, basic SQL queries)
- SMTP (what it is and what a basic SMTP dialog looks like)
- POP3, IMAP (what they do and what the differences are)
- basic Postfix configuration (understand the default settings in your main.cf, have read through the [basic configuration](#) document and know that your mail log file is at /var/log/mail.log)
- Debian/Linux (general system administration, using a text editor, reading log files)

MIGRATING FROM THE SARGE TUTORIAL

The "ispmail" tutorial is maintained since the ancient times of Debian "Woody". You may have followed former tutorials and now want to know how to upgrade to Etch properly. It is hard to provide exact instructions on what steps to go through. And you will definitely have a downtime unless you have multiple mail servers so you can migrate them one-by-one. Some basic information:

- The mailbox format and path stays the same. (Phew.)
- Courier POP/IMAP is replaced by Dovecot.
- SASL is no longer done through *auxprop*. It uses the Dovecot authentication setup thus saving you time.
- Passwords are now encrypted in the database.
- The virtual delivery agent is replaced by Dovecot's LDA.
- The database schema has changed completely. But there is a [conversion script](#) available to help you migrate your database.

Also you should definitely read [Etch's release notes](#) before attempting to upgrade your system from Sarge to Etch.

Please note that when switching from Courier to Dovecot your users will not automatically be subscribed to their IMAP folders any more. But fortunately Courier's 'courierimapsubscribed' file is compatible with Dovecot's 'subscriptions' file, but you need to remove the "INBOX." prefixes from the mailboxes. These files are located in the virtual mailbox directories.

THE COMPONENTS

The configuration described here uses these components:

- **Postfix** for receiving incoming emails from the internet and storing them to the users' mailboxes on the harddisk. And for receiving emails from your users that are sent out to the internet (relaying)
- **Dovecot** to allow your users to get their emails into their email client through POP3 and IMAP
- **Squirrelmail** as a webmail interface (although any IMAP capable webmail interface will do)
- **MySQL** as the database system that stores information about your domains, the user

accounts and email forwardings

- **AMaViS** for scanning incoming emails for viruses, spam and unwanted attachments

VIRTUAL DOMAINS IN A DATABASE

Have you ever wondered how internet service providers (ISPs) maintain their thousands of domains they receive mail for? There is surely nobody entering all these domains and aliases into a *main.cf* text configuration file manually. Postfix offers a better way to handle such domains and accounts by means of *virtual domains* and *virtual users*.

In addition to *local users* (those being listed in your */etc/passwd*) Postfix can handle any number of *virtual users* on *virtual domains*. Virtual users cannot log into your computer and they have neither a user ID nor a home directory. They just exist in the database. But if you connect Postfix to the database those users can suddenly receive emails. Postfix can work on a list of those virtual users and deliver emails to any directory you like. This would look like:

Virtual user	Virtual mailbox location
john@doe.org	/var/mail/doe.org/john
jack@doe.org	/var/mail/doe.org/jack
jeff@foo.org	/var/mail/foo.org/jeff

virtual_mailbox_maps

Postfix looks for the mapping of virtual users to virtual mailboxes in the *virtual_mailbox_maps* setting. The left-hand side (LHS) of the mapping is the email address and the right-hand side (RHS) is the location of the mailbox or maildir on your harddisk.

You also need to tell Postfix which virtual domains you want to use. If a domain is not used on your system then Postfix will reject emails. This would be a list of domains:

Virtual domain	Just some dummy string
doe.org	banana daiquiri
foo.org	rose garden

virtual_mailbox_domains

Postfix checks which virtual domains you want to receive mail for by looking at the *virtual_mailbox_domains* list. Since Postfix just has a notion of mappings (two columns) a list is (ab)using a mapping where the right-hand side (RHS) just contains any string.

So far the information could have been written into a text file. But you can imagine that this will become confusing quickly. Fortunately Postfix can also get this information from other sources like LDAP or SQL databases. So I am using SQL database tables here.

You have now seen that a *mapping* assigns one value to another. If you query a database you need to tell Postfix which two columns you mean. This is done through 'cf' files as documented at http://www.postfix.org/MYSQL_README.html or through "man 5 mysql_table".

Example file:

```
# Information on how to connect to your MySQL server
user = someone
```

```
password = some_password
hosts = 127.0.0.1

# The database name on the servers.
dbname = mailserver

# The SQL query template.
query = SELECT destination FROM virtual_aliases WHERE source='%s'
```

This file defines the way that Postfix can access data from your database. It would be suitable for a *virtual_alias_maps* mapping. Imagine you saved the above lines into a file **/etc/postfix/mysql-virtual-alias-maps.cf**. Then the following line in your main.cf would make Postfix query the database:

```
virtual_alias_maps = mysql:/etc/postfix/mysql-virtual-alias-maps.cf
```

How does this work? Imagine that Postfix is about to send an email to john@doe.net and wants to check the virtual alias map. Postfix then opens up a connection to the MySQL server at the IP address *127.0.0.1* and authenticates to the MySQL server with the username *someone* and the password *some_password*. It selects the database *mailserver* and finally runs a query:

```
SELECT destination FROM virtual_aliases WHERE source='john@doe.net'
```

Let us assume this query returns several results:

- jack@example.com
- jeff@example.com
- kerstin@example.com

That would be equal if you used a text file with aliases like this:

```
john@doe.net jack@example.com, jeff@example.com, kerstin@example.com
```

So much as a quick introduction on how mappings are used with databases.

VIRTUAL DOMAINS VERSUS LOCAL DOMAINS

It is important to understand the three different kinds of domains that Postfix knows. Most of the "it does not work" emails result from people mixing virtual and local domains. A domain is either a...

local domain

All domains listed as *mydestination* in your main.cf are treated as local domains. Your default domain (**/etc/defaultdomain**) is usually configured as a local domain. Emails for local domains are delivered to system users (those you list in **/etc/passwd**). The mails will be delivered to */var/mail* by default. You should consider using at least "localhost" as a local domain so that you can always receive mails for root@localhost. Imagine your database server has problems and sends that to your root account but your root account is on a virtual domain. Think of it as

a safety net.

virtual alias domain

Domains listed as *virtual_alias_domains* can be used for forwarding ("aliasing") email from an email address to another email address (or multiple addresses). Virtual alias domains do not receive email for any users. They only forward mail somewhere else.

The *virtual_alias_maps* mapping contains forwardings (source, destination) of users or domains to other email addresses or whole domains. Incidentally *virtual_alias_maps* also works for local email addresses, too. So you do not really need virtual alias domains as you can declare all domains as virtual mailbox domains and use virtual alias maps for aliases.

virtual mailbox domain

The most interesting domain type in this tutorial is the virtual mailbox domain. Such domains are listed in *virtual_mailbox_domains* and they will receive email for virtual users and store the email to mailboxes on your hard disk.

The *virtual_mailbox_maps* parameter tells Postfix where the mailbox directory is located on the hard disk for a certain user. The path is relative to the *virtual_mailbox_base* directory which is unset by default.

Note

A domain must only be listed in one of these three categories. Getting this wrong will lead to warnings and unpredictable behavior.

If you want to declare all domains as virtual mailbox domains you may wonder what you still need local domains for. You may at least want to set:

```
mydestination = localhost
```

so that you can send email to root@localhost for example.

We recommend you also betimes read the [upstream documentation on virtual domains](#) also known as the VIRTUAL_README.

STEP 1: INSTALLING THE NEEDED DEBIAN PACKAGES

Now that you hopefully understood the basic concepts of virtual domains, mappings and database accesses it is time to get your hands dirty. Become *root* on your server and make sure that your */etc/hostname* contains the host name *without* the domain part. The file */etc/mailname* is supposed to contain the fully-qualified host name *with* the domain part.

Your */etc/hosts* likely also needs to be fixed. Run **hostname --fqdn** and see if you get the fully-qualified hostname. If you just get the hostname without the domain please check that your */etc/hosts* file has the fully-qualified hostname *first* in the list.

Wrong:

```
20.30.40.50 mailserver42 mailserver42.example.com
```

Right:

```
20.30.40.50 mailserver42.example.com mailserver42
```

Then begin by installing Postfix with MySQL map support:

```
$> aptitude install postfix-mysql
```

This will also install the **postfix** package as a dependency. When asked for the general type of configuration choose "*Internet Site*". Answer the question for the "*Mail name*" by entering the fully qualified hostname (that means including the domain part) of your system.

\$> and mysql>

During the tutorial I will use the **\$>** notation to indicate commands you are supposed to enter in a shell. Sections that start with **mysql>** need to be entered as SQL commands.

If you intend to run the MySQL server on the same server:

```
$> aptitude install mysql-server-5.0
```

As you probably want to offer POP3 and IMAP services to your users you need to install:

```
$> aptitude install dovecot-pop3d  
$> aptitude install dovecot-imapd
```

Your users may like it if you filter out spam and (windows) viruses for them. Amavis is doing a good job here:

```
$> aptitude install amavisd-new libclass-dbi-mysql-perl
```

The Perl DBI module is needed so that AMaViS can read domain information from the database later. AMaViS is nearly useless unless you install its suggested packages to scan attachments for viruses and detect spam. You want to install these packages:

```
$> aptitude install spamassassin clamav-daemon cpio arj zoo nomarch lzop cabextract pa
```

Some of the suggested packages are not free enough to be included in Debian's *main* section. If you want to install them you first need to add the *contrib* and *non-free* sections to your Debian mirror in the **/etc/apt/sources.list** file. You usually just need to add the words **contrib** and **non-free** to your existing mirror line:

```
deb http://ftp.debian.org/debian/ etch main contrib non-free
```

Then you are ready to install the non-free packages:

```
$> aptitude update  
$> aptitude install lha unrar
```

Next you will want to install the OpenSSL package so that you can later create a proper SSL certificate letting your users use your mail server securely:

```
$> aptitude install openssl
```

If you intend to offer a webmail service I can recommend the Squirrelmail package. It will automatically install an Apache server if you do not yet have one installed. Type:

```
$> aptitude install squirrelmail
```

As your control information for Postfix will be stored in a MySQL database you may want to install the PhpMyAdmin software that allows you to administer the database and its data in your web browser:

```
$> aptitude install phpmyadmin libapache2-mod-php5 php5-mysql
```

You may want to test the mail server with the simple "telnet" client later:

```
$> aptitude install telnet
```

The console-based *mutt* email client lets you read mail from mailboxes directly from the hard disk. It will be helpful for testing the configuration. And it's even a very powerful IMAP email client that many people use as their main mail program. Maybe you start to like it, too. You should install it:

```
$> aptitude install mutt
```

STEP 2: CREATE THE DATABASE AND USER

Now it is time to create the MySQL database and its tables. If you are experienced in using MySQL you can enter SQL statements on the 'mysql' command line. Alternatively you may use *phpmyadmin* by pointing your browser at <http://yourmailserver/phpmyadmin>

If you just installed your MySQL server you will be able to login as user 'root' with an empty password. Set a new password for that account now. In the shell you need to run:

```
$> mysqladmin password root2007
```

Note

Replace 'root2007' by a more secure password. Install and use 'pwgen' if you do not feel creative. I will use 'root2007' throughout the tutorial though. Replace it by your own choice where appropriate.

Then create the database. Call it 'mailserver' (you will be asked for the above password):

```
$> mysqladmin -p create mailserver
```


For security reasons you will want to create another less privileged MySQL user account that your mail server will use. Connect to your database:

```
$> mysql -p
```

When you see the **mysql>** prompt enter the following SQL statement to grant the appropriate privileges:

```
mysql> GRANT SELECT ON mailserver.*  
TO mailuser@localhost  
IDENTIFIED BY 'mailuser2007';  
exit
```

This will create a user called 'mailuser' that has only the privilege to select/read data from the database but not to alter it. If you want to add or alter data in the database either use the 'root' account or create another account for that purpose. The password 'mailuser2007' is just an example. Please replace it by a more decent password.

STEP 3: CREATE THE DATABASE TABLES

Inside the newly created database you will have to create tables that store information about domains, forwardings and the users' mailboxes. Connect to the MySQL again and choose the 'mailserver' database:

```
$> mysql -p mailserver
```

You will see the **mysql>** prompt again. First create a table for the list of virtual domains that you want to host:

```
mysql> CREATE TABLE `virtual_domains` (  
id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,  
name VARCHAR(50) NOT NULL  
) ENGINE = InnoDB;
```

The next table contains information on the actual user accounts. Every user has a username and password. It is used for accessing the mailbox by POP3 or IMAP, logging into the webmail service or to send mail if they are not in your local network. As users tend to easily forget things the user's email address is also used as the login username. Create the users table:

```
mysql> CREATE TABLE `virtual_users` (  
id int(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,  
domain_id INT(11) NOT NULL,  
user VARCHAR(40) NOT NULL,  
password VARCHAR(32) NOT NULL,  
CONSTRAINT UNIQUE_EMAIL UNIQUE (domain_id,user),
```

```
FOREIGN KEY (domain_id) REFERENCES virtual_domains(id) ON DELETE CASCADE  
) ENGINE = InnoDB;
```

And finally a table is needed for aliases (email forwardings) from one account to another:

```
mysql>  
CREATE TABLE `virtual_aliases` (  
  id int(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  domain_id INT(11) NOT NULL,  
  source VARCHAR(40) NOT NULL,  
  destination VARCHAR(80) NOT NULL,  
  FOREIGN KEY (domain_id) REFERENCES virtual_domains(id) ON DELETE CASCADE  
) ENGINE = InnoDB;
```

You wonder about the *foreign keys*? They express that entries in the `virtual_aliases` and `virtual_users` tables are connected to entries in the `virtual_domains` table. This will keep the data in your database consistent because you cannot create virtual aliases or virtual users that are not connected to a virtual domain. And you avoid redundancy because you just store the domain name once - in the `virtual_domains` table - and nowhere else. The suffix 'ON DELETE CASCADE' means that if you delete a row from the referenced table that the deletion will also be done on the current table automatically. So you do not leave orphaned entries accidentally. Imagine that you do not host a certain domain any longer. You can remove the domain entry from the `virtual_domains` table and all dependent/referenced entries in the other tables will also be removed. This approach is also called a *normalized* database.

An example of the data in the tables:

virtual_domains

<i>id</i>	<i>name</i>
1	example.com
2	foobar.org

virtual_users

<i>id</i>	<i>domain_id</i>	<i>user</i>	<i>password</i>
1	1	john	summersun
2	1	steve	veryloud
3	2	kerstin	dogfood

The email addresses of these three users in the database would be:

- john@example.com
- steve@example.com
- kerstin@foobar.org

Let us add a simple alias:

virtual_aliases

<i>id</i>	<i>domain_id</i>	<i>source</i>	<i>destination</i>
1	1	steve	steve.miller@gmail.com
2	2	kerstin	kerstin42@yahoo.com
3	2	kerstin	kerstin@mycompany.com

This will make the mail for steve@example.com be forwarded to steve.miller@gmail.com. And the mail for kerstin@foobar.org is forwarded to both kerstin42@yahoo.com and kerstin@mycompany.com

Do not be scared if this way looks incredibly complicated. Such a database is not supposed to be maintained by manual SQL commands. But it is still important that you understand the schema of the database tables. There are already web-based administration tools that are supposed to make managing accounts easier.

STEP 4: CREATE THE DATABASE MAPPING FILES**virtual_mailbox_domains**

As described earlier a mapping in Postfix is just a table that contains a left-hand side (LHS) and a right-hand side (RHS). To make Postfix use MySQL to define a mapping we need a 'cf' file (configuration file). Start by creating a file called

/etc/postfix/mysql-virtual-mailbox-domains.cf for the virtual_mailbox_domains mapping:

```
user = mailuser
password = mailuser2007
hosts = 127.0.0.1
dbname = mailserver
query = SELECT 1 FROM virtual_domains WHERE name='%s'
```

Imagine Postfix wants to find out if *example.com* is a virtual mailbox domain. It will run the above SQL query and replace '%s' by 'example.com'. If it finds such an entry in the virtual_domains table it will return a '1'. Actually it does not matter what exactly is returned as long as there is a result.

And you need to make Postfix use this database mapping:

```
$> postconf -e virtual_mailbox_domains=mysql:/etc/postfix/mysql-virtual-mailbox-domains
```

(The **postconf -e** command conveniently adds configuration lines to your **/etc/postfix/main.cf** file. It also activates the new setting instantly so you do not have to reload the Postfix process.)

Postfix will now search your virtual_domains table to find out if a certain domain is a virtual mailbox domain. Let us try if this works. Create a new row in the virtual_domains table with one domain. Connect to your database:

```
$> mysql -p mailserver
```

and run this query:

```
mysql>
INSERT INTO virtual_domains (id, name) VALUES (1, 'example.com');
exit
```

Back on your shell you can now check if the 'example.com' domain is known as a virtual mailbox domain:

```
$> postmap -q example.com mysql:/etc/postfix/mysql-virtual-mailbox-domains.cf
```

You should get '1' as a result.

Note

If you get an error message telling you **postmap: warning: connect to mysql server 127.0.0.1: Access denied...** then you have a problem with the user account "mailuser" that you use to connect to the database. Check the MySQL privileges again.

If you get an error message reading **postmap: warning: connect to mysql server 127.0.0.1: Can't connect to MySQL server on '127.0.0.1'** then your MySQL server is either not running or at least not listening on 127.0.0.1. Check your MySQL server configuration (`/etc/mysql/my.cnf`).

virtual_mailbox_maps

Your first mapping is working. Great. Get straight to the second one. You will now define the *virtual_mailbox_maps* which is usually the mapping of email addresses (left-hand side) to the location of the user's mailbox on your harddisk (right-hand side). If you saved incoming email to the hard disk using Postfix' built-in *virtual delivery agent* then it would be queried to find out the mailbox path. But in this case the actual delivery is done by Dovecot's LDA (*local delivery agent*) so Postfix does not really care about the path. Postfix just needs to see if a certain email address belongs to a virtual user. Just as above you need an SQL query that searches for an email address and returns "1".

But first you need to deal with file system permissions. For security reasons it is suggested you create a new system user that will own all virtual mailboxes. The following shell commands will create a system group "vmail" with GID (group ID) 5000 and a system "user" with UID (user ID) 5000. (Make sure that UID and GID is not yet used or choose another - the number can be anything between 1000 and 65000 that is not yet used):

```
$> groupadd -g 5000 vmail
$> useradd -g vmail -u 5000 vmail -d /home/vmail -m
```

You need to set the *virtual_uid_maps* and *virtual_gid_maps* to these IDs:

```
$> postconf -e virtual_uid_maps=static:5000
$> postconf -e virtual_gid_maps=static:5000
```

Go create an entry in the `virtual_users` table for a test user **john@example.com**:

```
mysql>
INSERT INTO virtual_users (id, domain_id, user, password)
VALUES (1, 1, 'john', MD5('summersun'));
```

Next you will need to create a `cf` file to tell postfix about the SQL query for this table. But apparently you cannot get all the information from just the `virtual_users` table. The domain name has to be fetched from the `virtual_domains` table. "john@example.com" is our virtual user but the user part "john" is stored in the `virtual_users` table while the domain "example.com" is stored in the `virtual_domains` table. Thanks to the `JOIN` SQL statement you can join these two tables together. A query that would tell us the email addresses of all virtual users would look like this:

```
mysql>
SELECT CONCAT(virtual_users.user, '@', virtual_domains.name) AS email
FROM virtual_users
LEFT JOIN virtual_domains ON virtual_users.domain_id=virtual_domains.id;
```

MySQL should print:

```
+-----+
| email          |
+-----+
| john@example.com |
+-----+
```

In addition to the email address it is also important to get the user's password later on. Since the path to the user's mailbox is fixed it is not important to get that information from the database. The directory structure will be `/home/vmail/$DOMAIN/$USER`. So in John's example it would be `/home/vmail/example.com/john`.

The query just has to ask for the `password` field, too:

```
mysql>
SELECT CONCAT(virtual_users.user, '@', virtual_domains.name) AS email,
       virtual_users.password
FROM virtual_users
LEFT JOIN virtual_domains ON virtual_users.domain_id=virtual_domains.id;
```

The result of that query would be:

```
+-----+-----+
| email          | password |
+-----+-----+
```

```
| john@example.com | 14cbfb845af1f030e372b1cb9275e6dd |
+-----+-----+
```

As you see the password is not stored as plain text but the MD5 hash is saved. The query may look a bit complicated if you do not have to deal with SQL queries every day. At least it is not very handy. So instead of writing this query into the cf file there is a feature called *views* that has finally been introduced with MySQL 5.0. A view allows to store queries under a table name. Go create that view:

```
mysql>
CREATE VIEW view_users AS
SELECT CONCAT(virtual_users.user, '@', virtual_domains.name) AS email,
       virtual_users.password
FROM virtual_users
LEFT JOIN virtual_domains ON virtual_users.domain_id=virtual_domains.id;
```

Now we can get the information about virtual users with a simple SELECT query on this *view_users* view:

```
mysql>
SELECT * FROM view_users;
```

You should get the same result as above:

```
+-----+-----+
| email          | password                                     |
+-----+-----+
| john@example.com | 14cbfb845af1f030e372b1cb9275e6dd |
+-----+-----+
```

Now things are a bit simpler and you can finally create a cf file at **/etc/postfix/mysql-virtual-mailbox-maps.cf** that is as simple as:

```
user = mailuser
password = mailuser2007
hosts = 127.0.0.1
dbname = mailserver
query = SELECT 1 FROM view_users WHERE email='%s'
```

Tell Postfix that this mapping file is supposed to be used for the *virtual_mailbox_maps* mapping:

```
$> postconf -e virtual_mailbox_maps=mysql:/etc/postfix/mysql-virtual-mailbox-maps.cf
```

Test if Postfix is happy with this mapping by asking it where the mailbox directory of our john@example.com user would be:

```
$> postmap -q john@example.com mysql:/etc/postfix/mysql-virtual-mailbox-maps.cf
```

You should get **1** back which means that **john@example.com** is an existing virtual mailbox user on your server. Later in the Dovecot configuration part you will also use the *email* and *password* fields but Postfix does not need them here.

virtual_alias_maps

The *virtual_alias_maps* mapping is used for forwarding emails from one email address to another. Examples of how entries in this mapping may look:

source (LHS)	destination (RHS)	Meaning
john@example.com	jmiller@gmail.com	Forward John's mail to gmail
john@example.com	john@example.com , jmiller@gmail.com	Deliver one copy to the original account john@example.com but also send a copy to jmiller@gmail.com
@example.com	john@example.com	Deliver all email for the domain to john@example.com unless there is a specific user account. If kerstin@example.com does not exist then her mail would be sent to john. If kerstin@example.com is a valid user then she would get the mail. This is called a <i>catchall</i> account.

See **man 5 virtual** for a more formal definition.

As you see it is possible to name multiple destinations separated by commas. In the database the same effect can be achieved by using different rows instead. The second entry above should be split into two rows:

source (LHS)	destination (RHS)
john@example.com	john@example.com
john@example.com	jmiller@gmail.com

Let us add this example to the database:

```
mysql>
INSERT INTO virtual_aliases (id, domain_id, source, destination)
VALUES (1, 1, 'john', 'john@example.com'),
      (2, 1, 'john', 'jmiller@gmail.com');
```

Let us also create a view for the virtual aliases:

```
mysql>
CREATE VIEW view_aliases AS
SELECT CONCAT(virtual_aliases.source, '@', virtual_domains.name) AS email,
      destination
FROM virtual_aliases
```

```
LEFT JOIN virtual_domains ON virtual_aliases.domain_id=virtual_domains.id;
```

This query is a bit more complicated than the previous one. It gets all rows from `virtual_aliases` and JOINS the `virtual_domains` table (the `domain_id` field of the `virtual_aliases` table matches the `id` field of the `virtual_domains` table). Too complicated? That will not bother you any longer because you can now use the `view_aliases` view:

```
mysql>
SELECT * FROM view_aliases;
```

and get the appropriate result:

```
+-----+-----+
| email          | destination          |
+-----+-----+
| john@example.com | john@example.com      |
| john@example.com | jmillergmail.com      |
+-----+-----+
```

The SQL view works perfectly so we can use it for Postfix. Create another cf file at `/etc/postfix/mysql-virtual-alias-maps.cf`:

```
user = mailuser
password = mailuser2007
hosts = 127.0.0.1
dbname = mailserver
query = SELECT destination FROM view_aliases WHERE email='%s'
```

Test if the mapping file works as expected:

```
$> postmap -q john@example.com mysql:/etc/postfix/mysql-virtual-alias-maps.cf
```

You should see the two expected destinations:

```
john@example.com,jmillergmail.com
```

Before you define the `virtual_alias_maps` setting there is a small quirk you need to take care of. There is a special kind of forwarding: the "catchall" alias. Catchalls catch all emails for a domain if there is no specific user account. A catchall alias looks like "`@example.com`" and forwards email for the whole domain to one account. We have created the ['john@example.com'](#) user and would like to forward all *other* email on the domain to ['kerstin@gmail.com'](#). So we would add a catchall alias like:

email	destination
-------	-------------

Catch-alls catch spam

Catch-all aliases may look comfortable because they forward all email to one person without the need for creating aliases. They catch an awful lot of spam though. Spammers often try to guess email addresses at a known domain. And with a catch-all

email	destination
@example.com	kerstin@gmail.com

alias you will receive spam for any of those guessed email addresses. Try to avoid them and rather define the existing email addresses.

Now imagine what happens when Postfix receives an email for 'john@example.com'. Postfix will first check if there are any aliases in the *virtual_alias_maps* table. It finds the catchall entry as above and since there is no more specific alias the catchall account matches and the email is redirected to 'kerstin@gmail.com'. This is probably not what you wanted. So you would need to make the table rather look like this:

email	destination
@example.com	kerstin@gmail.com
john@example.com	john@example.com

More specific aliases have precedence over general catchall aliases. Postfix will find an entry for 'john@example.com' first and find that email should be "forwarded" to 'john@example.com' - the same email address. This trickery may sound weird but it is needed if you plan to use catchall accounts. So the *virtual_alias_maps* mapping must obey both the "view_aliases" view and this "john-to-himself" mapping. Create a cf file `/etc/postfix/mysql-email2email.cf` for the latter mapping:

```
user = mailuser
password = mailuser2007
hosts = 127.0.0.1
dbname = mailserver
query = SELECT email FROM view_users WHERE email='%s'
```

Check that you get John's email address back when you ask Postfix if there are any aliases for him:

```
$> postmap -q john@example.com mysql:/etc/postfix/mysql-email2email.cf
```

The result should be the same address:

```
john@example.com
```

Now you need to tell Postfix that these two mappings should be searched by adding this line to your `main.cf`:

```
$> postconf -e virtual_alias_maps=mysql:/etc/postfix/mysql-virtual-alias-maps.cf,mysql:
```

Please note that the order of the two mappings are important here. Postfix will look for all matching entries in the first virtual-aliases mapping and use them as aliases. Only if there are *no* results found then Postfix will consult the second *email2email* mapping.

You did it! All mappings are set up and the database is generally ready to be filled with domains

and users. Make sure that only 'root' and the 'postfix' user can read the cf files - after all your database password is stored there:

```
chgrp postfix /etc/postfix/mysql-*.cf
chmod u=rw,g=r,o= /etc/postfix/mysql-*.cf
```

STEP 5: DELIVER EMAILS THROUGH THE DOVECOT LDA

Postfix comes with a mail delivery agent (MDA) called "virtual" that will usually deliver emails to virtual mailboxes. But Dovecot comes with an own (local) delivery agent also called "Dovecot LDA" that offers more functionality. To make Postfix use that agent you will have to add a service to your **/etc/postfix/master.cf**:

```
dovecot    unix    -        n        n        -        -        pipe
           flags=DRhu user=vmail:vmail argv=/usr/lib/dovecot/deliver -d ${recipient}
```

Restart Postfix:

```
$> postfix reload
```

Also make Postfix use that service for virtual delivery lines to your **/etc/postfix/main.cf**:

```
$> postconf -e virtual_transport=dovecot
$> postconf -e dovecot_destination_recipient_limit=1
```

So far this will make Postfix pass on incoming emails to virtual users to the **/usr/lib/dovecot/deliver** program. Now it is time to configure Dovecot.

STEP 6: CONFIGURE DOVECOT

Let us configure Dovecot which provides both a POP3 and an IMAP service. The configuration files for Dovecot is found under **/etc/dovecot**. Start with the...

/etc/dovecot/dovecot.conf

Set the line **protocols** to:

```
protocols = imap imaps pop3 pop3s
```

so that Dovecot starts the IMAP and POP3 services and also its equivalents that work over an encrypted SSL (*secure socket layer*) connection.

If users start to complain that they cannot fetch their emails consider setting:

```
disable_plaintext_auth = no
```

This will allow plaintext passwords over an unsecured (non-SSL) connection. By default it is set to 'yes' for security reasons. Setting it to 'no' will mean less security but may help the "less

fortunate".

A more important setting is:

```
mail_location = maildir:/home/vmail/%d/%n/Maildir
```

which will tell that the users' mailboxes are always found at **/home/vmail/DOMAIN/USER** and that it should be in *maildir* format.

Note

Previous versions of this tutorial recommended to use **mail_location = maildir:/home/vmail/%d/%n** instead (without the trailing **Maildir** part). It is now recommended you add the extra directory so that additional control files in the virtual mailbox directory do not accidentally get confused with mail folders. Otherwise it may happen that your users see e.g. filter control files as mail folders. So if you have an existing directory structure you have to create a **Maildir** folder right there and move all mail folders (cur, new, tmp and all folders starting with a dot) there.

If you already have virtual mailboxes on your system because you followed the previous tutorials for Sarge or Woody you may want to define the IMAP namespace explicitly so that the users find their folder where they have always been:

```
namespace private {  
    separator = .  
    prefix = INBOX.  
    inbox = yes  
}
```

Next look for a section called "auth default". First define the allowed authentication mechanisms:

```
mechanisms = plain login
```

Then inside that same section you need to change:

```
passdb sql {  
    args = /etc/dovecot/dovecot-sql.conf  
}
```

which tells Dovecot that the passwords are stored in an SQL database and:

```
userdb static {  
    args = uid=5000 gid=5000 home=/home/vmail/%d/%n allow_all_users=yes  
}
```

to tell Dovecot where the mailboxes are located. This is similar to the *mail_location* setting.

You will want to comment out the section called **passdb pam** that deals with system users. Otherwise Dovecot will also look for system users when someone fetches emails which leads to warnings in your log file.

Now look for another section called **socket listen**. Here you define socket files that are used to interact with Dovecot's authentication mechanism. Make the section read:

```
socket listen {
    master {
        path = /var/run/dovecot/auth-master
        mode = 0600
        user = vmail
    }

    client {
        path = /var/spool/postfix/private/auth
        mode = 0660
        user = postfix
        group = postfix
    }
}
```

The *master* section is needed to give Dovecot's delivery agent (the program that saves a new mail to the user's mailbox) access to the userdb information. The *client* section creates a socket inside the "chroot" directory of Postfix. chroot means that parts of Postfix are jailed into **/var/spool/postfix** and can only access files beneath that directory. It is a good security measure so that even if Postfix had bugs and were attacked the attacker would not be able to access `/etc/passwd` for example.

And finally the **protocol lda** section needs to be customized. The LDA (*local delivery agent*) is more capable than Postfix' built-in virtual delivery agent. It allows for quotas and [Sieve](#) (ships with the *dovecot-common* package) filtering. Let the section be:

```
protocol lda {
    log_path = /home/vmail/dovecot-deliver.log
    auth_socket_path = /var/run/dovecot/auth-master
    postmaster_address = postmaster@example.com
    mail_plugins = cmusieve
    global_script_path = /home/vmail/globalsieverc
}
```

Please change the above *postmaster* email address to a valid address where the administrator can be reached.

Edit **/etc/dovecot/dovecot-sql.conf** and change these settings:

```
driver = mysql
connect = host=127.0.0.1 dbname=mailserver user=mailuser password=mailuser2007
default_pass_scheme = PLAIN-MD5
```

```
password_query = SELECT email as user, password FROM view_users WHERE email='%u';
```

Restart Dovecot:

```
$> /etc/init.d/dovecot restart
```

Now look at your **/var/log/mail.log** logfile. You should see:

```
dovecot: Dovecot v1.0.rc15 starting up  
dovecot: auth-worker(default): mysql: Connected to 127.0.0.1 (mymailserver)
```

Upon the first restart of Dovecot it will also generate Diffie-Hellman parameters and fix permissions of **/var/run/dovecot** and **/var/run/dovecot/login**. That is perfectly normal.

Before you send a first test email you will need to fix file system permissions for the **/etc/dovecot/dovecot.conf** file so that the *vmail* user can access the Dovecot configuration. The reason is that Postfix starts the delivery agent with *vmail* permissions:

```
$> chgrp vmail /etc/dovecot/dovecot.conf  
$> chmod g+r /etc/dovecot/dovecot.conf
```

STEP 7: SEND A TEST MAIL THROUGH TELNET

Emulating an SMTP session with telnet

Let us try to send an email to the user now. As the "example.com" is not really existing and your DNS settings will likely not point to the right server we are simulating an SMTP session with the *telnet* command:

```
$> telnet localhost smtp
```

The server should reply:

```
Trying 127.0.0.1...  
Connected to localhost.  
Escape character is '^]'.  
220 mailtest ESMTP Postfix (Debian/GNU)
```

Great. Postfix is listening and wants us to speak SMTP. First we need to be friendly:

```
ehlo example.com
```

Postfix appreciates our friendliness and tells us which features it provides:

```
250-my-new-mailserver  
250-PIPELINING  
250-SIZE 10240000  
250-VRFY
```

```
250-ETRN
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
```

Hey, Postfix, we have a mail from steve@example.com:

```
mail from:<steve@example.com>
```

Looks like Postfix is happy with that because return codes that start with a '2' are good news:

```
250 2.1.0 Ok
```

Tell Postfix who the recipient of the mail is:

```
rcpt to:<john@example.com>
```

Postfix accepts that:

```
250 2.1.5 Ok
```

Then we are ready to send the actual mail:

```
data
```

Postfix agrees and tells us we can send the actual mail now and end our input with a dot on an empty line:

```
354 End data with <CR><LF>.<CR><LF>
```

Okay, type in the mail:

```
Hi John,

just wanted to drop you a note.
.
```

Postfix tells us it has received the mail and queued under a queue ID:

```
250 2.0.0 Ok: queued as A9D64379C4
```

Thanks, Postfix, we are done:

```
quit
```

Checking the logs

Take a look at the `/var/log/mail.log` file now. You should see something similar to:

```
postfix/smtpd[...]: connect from localhost[127.0.0.1]
postfix/smtpd[...]: 5FF712A6: client=localhost[127.0.0.1]
postfix/cleanup[...]: 5FF712A6: message-id=<...>
postfix/qmgr[...]: 5FF712A6: from=<steve@example.com>, size=364, nrcpt=1 (queue active)
postfix/pipe[...]: 5FF712A6: to=<john@example.com>, relay=dovecot, ..., status=sent (de
postfix/qmgr[...]: 5FF712A6: removed
postfix/smtpd[...]: disconnect from localhost[127.0.0.1]
```

The delivery has worked. Postfix has correctly determined that the destination domain is a virtual domain and forwarded the email to the 'dovecot' service.

Checking the user's mailbox

The email should now be somewhere under `/home/vmail/example.com/john`. Let us take a look:

```
$> cd /home/vmail/example.com/john
$> find
.
./cur
./new
./new/1179521979.V801I2bbf7M15352.mailtest
./tmp
```

There sits the email. Try to read the mail with the "mutt" program:

```
$> mutt -f .
```

The new email is shown:

```
q:Quit d:Del u:Undel s:Save m:Mail r:Reply g:Group ?:Help
1 N   May 18 steve@example.c (0.1K)
```

Press ENTER to read the email:

```
From: steve@example.com
To: undisclosed-recipients: ;

Hi John,

just wanted to drop you a note.
```

So the email arrived at John's account. Perfect. Press 'q' twice to exit mutt again.

STEP 8: TEST FETCHING EMAILS WITH IMAP AND POP3

John will surely prefer to read his mail in a comfortable mail program. So he needs a way to get access to his mailbox. Two protocols come to play here:

- POP (Post Office Protocol) is a simple protocol that lets you fetch email from a single mailbox. It is usually used to get all emails and then delete them on the server.
- IMAP (Internet Messaging Application Protocol) is also used to fetch email but you can maintain different mailboxes. The *inbox* is where your incoming emails are stored. But users can move emails to different directories. IMAP is useful when you want to access your email from different locations without losing mail because you fetched it from another location.

Testing POP3

Let us try to create a POP3 connection and retrieve John's email:

```
$> telnet localhost pop3
```

The server replies:

```
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
+OK Dovecot ready.
```

Login as John:

```
user john@example.com
```

The server should accept that:

```
+OK
```

Send the password:

```
pass summersun
```

The server should recognize the correct password:

```
+OK Logged in.
```

Get a list of John's emails:

```
list
```

Dovecot will tell you that one email is in the mailbox:

```
+OK 1 messages:
1 474
.
```

Fetch that email number 1:


```
retr 1
```

Dovecot sends you the email:

```
+OK 474 octets
Return-Path: <steve@example.com>
X-Original-To: john@example.com
Delivered-To: john@example.com
Received: from example.com (localhost [127.0.0.1])
  by ... (Postfix) with ESMTP id 692DF379C7
  for <john@example.com>; Fri, 18 May 2007 22:59:31 +0200 (CEST)
Message-Id: <...>
Date: Fri, 18 May 2007 22:59:31 +0200 (CEST)
From: steve@example.com
To: undisclosed-recipients:;

Hi John,

just wanted to drop you a note.
.
```

Close the connection to the POP3 server:

```
quit
```

The server will disconnect you:

```
+OK Logging out.
Connection closed by foreign host.
```

Test IMAP

Instead of going through the following procedure (IMAP is rather complicated) you may as well just use *mutt* to create an IMAP connection:

```
$> mutt -f imap://john@example.com@localhost
```

Alternatively you can open up a raw IMAP connection to the server and enter the IMAP commands yourself:

```
$> telnet localhost imap2
```

You should get a connection:

```
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^['.
```

```
* OK Dovecot ready.
```

IMAP commands always start with a number and reply to that command with the same number. So the following commands must be entered *with* the number at the beginning of each line. Login with username and password:

```
1 login john@example.com summersun
```

Dovecot logs you in:

```
1 OK Logged in.
```

Ask Dovecot for a list of John's mail folders:

```
2 list "" "*"
```

Here comes the list:

```
* LIST (\HasNoChildren) "." "INBOX"  
2 OK List completed.
```

Select your inbox:

```
3 select "INBOX"
```

Dovecot gives you all kinds of information about that folder:

```
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft)  
* OK [PERMANENTFLAGS (\Answered \Flagged \Deleted \Seen \Draft \*)] Flags permitted.  
* 1 EXISTS  
* 0 RECENT  
* OK [UIDVALIDITY 1180039205] UIDs valid  
* OK [UIDNEXT 3] Predicted next UID  
3 OK [READ-WRITE] Select completed.
```

You see that one email exists. Fetch it:

```
4 fetch 1 all
```

IMAP will just give you basic information on the email:

```
* 1 FETCH (FLAGS (\Seen) INTERNALDATE .....  
4 OK Fetch completed.
```

To read the actual mail body you need to fetch it explicitly:

```
5 fetch 1 body[]
```

Here it comes:

```
* 1 FETCH (BODY[] {474}
Return-Path: <steve@example.com>
X-Original-To: john@example.com
Delivered-To: john@example.com
Received: from example.com (localhost [127.0.0.1])
        by ... (Postfix) with ESMTP id 692DF379C7
        for <john@example.com>; Fri, 18 May 2007 22:59:31 +0200 (CEST)
Message-Id: <...>
Date: Fri, 18 May 2007 22:59:31 +0200 (CEST)
From: steve@example.com
To: undisclosed-recipients:;

Hi John,

just wanted to drop you a note.
)
5 OK Fetch completed.
```

Disconnect from the server:

```
6 logout
```

Dovecot logs you out:

```
* BYE Logging out
6 OK Logout completed.
Connection closed by foreign host.
```

POP3 and IMAP appear to work. You could now use any email program like Kmail, Evolution or Thunderbird/Icedove and set up a POP3 or IMAP email account. The quickest way to check encrypted connections is using mutt again:

```
$> mutt -f imaps://john@example.com@localhost
```

If you use other mail programs note that the username will be the email address 'john@example.com' and the password is 'summersun'. You can try these kinds of connections:

- POP3
- IMAP
- POP3 with TLS/SSL enabled
- IMAP with TLS/SSL enabled

When using TLS/SSL you will get a warning that the certificate of the server cannot be trusted. Dovecot ships with a sample certificate so that you can test your setup and use TLS/SSL to fetch emails securely. Unfortunately the so called "postinst" script (that is called after the package 'dovecot-common' is installed) does not seem to create the certificate correctly. The common name lacks the domain part. (I have reported this issue under bug number [#425917](#))

but this will probably not be fixed in Etch.) So it is advised that you create your own certificate with the proper server name:

```
$> openssl req -new -x509 -days 3650 -nodes -out /etc/ssl/certs/dovecot.pem \
      -keyout /etc/ssl/private/dovecot.pem
```

The certificate and key will be created while you get asked a few questions:

```
Generating a 1024 bit RSA private key
.....+++++
.....+++++
writing new private key to '/etc/ssl/certs/dovecot.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:DE
State or Province Name (full name) [Some-State]:Hamburg
Locality Name (eg, city) []:Hamburg
Organization Name (eg, company) [Internet Widgits Pty Ltd]:workaround.org
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:mailtest.workaround.org
Email Address []:postmaster@workaround.org
```

Of course you should fill in your own information here. The most important setting is the *Common Name* which must contain the fully-qualified name of your mail server. Oh, and this certificate will be valid for 10 years (3650 days) - adjust that period as you want.

Do not forget to set the permissions on the private key so that no unauthorized people can read it:

```
$> chmod o= /etc/ssl/private/dovecot.pem
```

STEP 9: AUTHENTICATED SMTP

It is important that you understand the problem of *open relays* first. For security reasons Postfix allows either all mails from IP addresses listed in *mynetworks* or to local or virtual domains to send emails. All other cases like someone sending mail to another person on the internet is called *relaying* and is limited for security reasons. If you would relay mail for everybody then spammers would abuse your mail server to send out emails on their behalf. You would be a so called *open relay*. You would add to the spam problem and probably get blacklisted quickly. That will make it hard to ever send out emails again. Setting the *smtpd_recipient_restrictions* right (as described below) is very important.

Generally you can define the networks that are allowed to relay through your mail server by

setting the *mynetworks* parameter in your **main.cf**. Usually you set it to your local network so local user do not need to authenticate:

```
$> postconf -e mynetworks=192.168.50.0/24
```

One case of relaying from outside of your network is but important in real-life. Imagine you have a user who is currently not connected to your local network but wants to send out an email to the internet through your mail server. According to the above rules the user would not be able to do that because neither are they on your network nor are they sending an email to one of your domains. You need to find a way to make your remote user be trusted by your mail server. The users will need to show you their username and password so you know relaying should be permitted. This is what *authenticated SMTP* is about. And most email clients have that feature built in.

Authenticated SMTP with Postfix has always been a pain. It was done through the SASL (*Simple Authentication and Security Layer*) library that was part of the Cyrus mail server. It was nearly impossible to debug and threw error messages that were gibberish and misleading. Fortunately starting with Postfix 2.3 we can [make Postfix ask the Dovecot server](#) to verify the username and password. And since you already configured Dovecot this is really easy now. Postfix just needs some extra configuration:

```
$> postconf -e smtpd_sasl_type=dovecot
$> postconf -e smtpd_sasl_path=private/auth
$> postconf -e smtpd_sasl_auth_enable=yes
$> postconf -e smtpd_recipient_restrictions=permit_mynetworks,permit_sasl_authenticated
```

smtpd_sasl_auth_enable enables SMTP authentication altogether. And **smtpd_recipient_restrictions** defines rules that are checked after the remote user sends the **RCPT TO:** line during the SMTP dialog. In this case relaying is allowed if:

- **permit_mynetworks**: the user is in the local network (**mynetworks**) or
- **permit_sasl_authenticated**: if the user is authenticated or
- **reject_unauth_destination**: the mail is destined to a user of a domain that is a local or virtual domain on this system (**mydestination**, **virtual_alias_domains** or **virtual_mailbox_domains**).

There are further restrictions (**smtpd_client_restrictions**, **smtpd_helo_restrictions**, **smtpd_sender_restrictions**) but unless you really want to customize them it is okay to put all restrictions into the **smtpd_recipient_restrictions**.

Try to authenticate during an SMTP session:

```
$> telnet localhost smtp
```

The server will let you in:

```
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^'.
```

```
220 mailtest ESMTP Postfix (Debian/GNU)
```

Say hello:

```
ehlo example.com
```

Postfix will present a list of features that are available during the SMTP dialog:

```
250-mailtest
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-ENHANCEDSTATUSCODES
250-8BITMIME
250 DSN
```

Send the authentication string with a Base64-encoded password:

```
auth plain am9obkBleGFtcGx1LnNvbQBqb2huQGV4YW1wbGUuY29tAHN1bW1lc nN1bg==
```

The server should accept that authentication:

```
235 2.0.0 Authentication successful
```

Disconnect from Postfix:

```
quit
```

Good bye:

```
221 2.0.0 Bye
```

Authentication works. Great.

Note

If you have set John's password to something other than 'summersun' you need Base64-encode it yourself. Use:

```
perl -MMIME::Base64 -e \
    'print encode_base64("john@example.com\0john@example.com\0password")';
```

Now you can test sending email with authentication enabled. To make even your local network untrusted temporarily you can set:

```
$> postconf -e mynetworks=
```

temporarily. Fire up your mail program and watch your **mail.log** while you send an email to a domain on the internet. I recommend you send a test email to **devnull@workaround.org** which is an email address that will just discard your email. If everything worked well your logfile will show:

```
postfix/smtpd[4032]: 1234567890: client=..., sasl_method=PLAIN, sasl_username=john@example.com
postfix/cleanup[4040]: 2EAE8379CB: message-id=<...>
postfix/qmgr[3963]: 1234567890: from=<john@example.com>, size=830, nrcpt=1 (queue active)
postfix/smtpd[4032]: disconnect from ...
postfix/smtp[4041]: 1234567890: to=<devnull@workaround.org>,
    relay=torf.workaround.org[212.12.58.129]:25, delay=6,
    delays=0.09/0.08/5.6/0.23, dsn=2.0.0, status=sent
    (250 OK id=1HsPC3-0008UJ-05)
postfix/qmgr[3963]: 2EAE8379CB: removed
```

Otherwise in case of an error your logfile would look like:

```
postfix/smtpd[4032]: connect from ...[10.20.30.40]
postfix/smtpd[4032]: warning: ...[10.20.30.40]: SASL PLAIN authentication failed:
postfix/smtpd[4032]: lost connection after AUTH from ...[10.20.30.40]
postfix/smtpd[4032]: disconnect from ...[10.20.30.40]
```

Your email program may have warned you that the mail server uses an untrusted SSL certificate. The default certificate that comes with Postfix is made out to the funny nonexistent "Office for Complication of Otherwise Simple Affairs" organisation. This is sufficient for testing but just like you did for Dovecot it is advised you create a proper SSL certificate that at least has the correct host name included. The default certificate is stored at **/etc/ssl/certs/ssl-cert-snakeoil.pem** and the default private key is stored at **/etc/ssl/private/ssl-cert-snakeoil.key**. Consider creating a new certificate/key pair and store it as **postfix.pem**:

```
$> openssl req -new -x509 -days 3650 -nodes -out /etc/ssl/certs/postfix.pem \
    -keyout /etc/ssl/private/postfix.pem
```

Same procedure as above when you created a certificate for Dovecot. Just remember to set the "Common Name" to the fully-qualified hostname. You could as well use the same certificate you created for Dovecot if the server name is the same. In that case just use the files **/etc/ssl/certs/dovecot.pem** and **/etc/ssl/private/dovecot.pem** below.

Do not forget to set the permissions on the private key so that no unauthorized people can read it:

```
$> chmod o= /etc/ssl/private/postfix.pem
```

You will just have to tell Postfix where to find your certificate and private key:

```
$> postconf -e smtpd_tls_cert_file=/etc/ssl/certs/postfix.pem
```

```
$> postconf -e smtpd_tls_key_file=/etc/ssl/private/postfix.pem
```

When you relay through Postfix again you should not get that certificate warning any longer.

By default Postfix will allow that the login data for SMTP authentication is sent in plain text. You better only allow encrypted transmission of the credentials by setting these parameters:

```
$> postconf -e smtpd_use_tls=yes  
$> postconf -e smtpd_tls_auth_only=yes
```

What these parameters do is require encryption for SASL authentication to be used so that if you're using the PLAIN or LOGIN SASL methods your passwords aren't transmitted in the clear. When the client initially connects to the server, the AUTH command isn't offered by the server. If the client issues the STARTTLS command and successfully negotiates the TLS connection, the client sends the EHLO command a second time and this time the server offers the AUTH command. This won't require any kind of encryption from clients who don't need to authenticate (i.e. servers that connect to send mail to the domains our server is a final destination for, as opposed to our end users attempting to relay mail through it to external domains).

If you truly do want to forbid unencrypted SMTP connections (I do this on ports 587 and 465), you'd want to use either "**smtpd_tls_security_level = encrypt**" (for STARTTLS, generally on port 587) or "**smtpd_tls_wrappermode = yes**" (for SSL encryption from the initial connection on, generally on port 465).

STEP 10: AMAVIS: FILTERING SPAM AND VIRUSES

Two annoying issues when using email are spam and viruses. Fortunately you can fight both using AMaViS (A Mail Virus Scanner). AMaViS is an interface between Postfix, spamassassin (famous for its Bayesian spam filtering capabilities) and optionally a virus scanner. AMaViS contains the spam filtering part but has no virus scanner built in. You should install ClamAV for that purpose. It is a free virus scanner that gets updated frequently.

In Debian Etch the AMaViS configuration has been moved from the **/etc/amavis/amavisd.conf** to different files in the **/etc/amavis/conf.d** directory. Fortunately the virus scanner "ClamAV" is already configured by default so you do not need to touch that. However I suggest you take a closer look at **/etc/amavis/conf.d/20-debian_defaults**:

- **\$sa_spam_subject_tag**: if this line stays uncommented then every email that AMaViS suspects to be spam get this string inserted into the email's subject. If you do not want to alter the subject you should comment this line out. Users can still filter out spam by checking the **X-Spam-Status** header.
- **\$sa_tag_level_deflt**: spam with a score that is greater or equal to this level will get spam headers added. Debugging is easier if you always add the headers so you should consider setting it to some very low score (e.g. "undef" - which is the lowest possible level - so that headers are always added).
- **\$sa_tag2_level_deflt**: emails with a spam score greater or equal to this level will be marked as spam. Usually this default is reasonable. Lower it if too much spam gets through. Raise it if you get too many regular mails caught as spam.
- **\$sa_kill_level_deflt**: should be set to the same value as **\$sa_tag2_level_deflt**

- **\$final_spam_destiny**: the default **D_BOUNCE** setting is plain stupid here. The usual approach is to *tag* email as spam. But if you bounced them you would hit some innocent person because spam mails never contain the correct sender address. Just let the user decide what to do with spam. So it is strongly recommended that you set this to **D_PASS**.
- **\$banned_filename_re**: carefully check this list because these patterns tell AMaViS when to bounce an email because it contains data that you do not like to receive in an email

In the file **/etc/amavis/conf.d/15-content_filter_mode** you also need to remove the '#' from the **@bypass_...** lines so that spam and virus filtering gets enabled.

In the **/etc/spamassassin/local.cf** file you should add a line to disable automatic expiration of learned spam emails because that feature conflicts with AMaViS:

```
bayes_auto_expire 0
```

Restart AMaViS if you have made changes to the config files:

```
$> /etc/init.d/amavis restart
```

Make sure that AMaViS is listening on TCP port 10024:

```
$> netstat -nap | grep 10024
```

You should get this output:

```
tcp  0  0 127.0.0.1:10024  0.0.0.0:*    LISTEN  12345/amavisd
```

If you get such a line then AMaViS is running and waiting for incoming SMTP sessions. Otherwise check your **/var/log/mail.log** file - perhaps you have made a mistake in the configuration files.

A few words on how AMaViS will be plugged into Postfix. If some person sends you an email from the internet it will be received by Postfix (the *smtpd* daemon) first on TCP port 25 (SMTP). If Postfix accepts the email it will be forwarded to AMaViS on TCP port 10024 (SMTP). And if AMaViS is happy with the email's contents it will be sent back to Postfix on TCP port 10025 (SMTP). Postfix finally delivers the email to the actual recipient.

Making Postfix forward emails to AMaViS is done by setting the **content_filter** setting. Also set another option that will be explained later:

```
$> postconf -e content_filter=smtp-amavis:[127.0.0.1]:10024
$> postconf -e receive_override_options=no_address_mappings
```

But we need to define the **smtp-amavis** service first in the **/etc/postfix/master.cf**. And also we need Postfix to listen on TCP port 10025 for emails that get sent back from AMaViS:

```
smtp-amavis unix -      -      n      -      2      smtp
-o smtp_data_done_timeout=1200
-o smtp_send_xforward_command=yes
-o disable_dns_lookups=yes
```

```
-o max_use=20

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_delay_reject=no
-o smtpd_client_restrictions=permit_mynetworks,reject
-o smtpd_helo_restrictions=
-o smtpd_sender_restrictions=
-o smtpd_recipient_restrictions=permit_mynetworks,reject
-o smtpd_data_restrictions=reject_unauth_pipelining
-o smtpd_end_of_data_restrictions=
-o mynetworks=127.0.0.0/8
-o smtpd_error_sleep_time=0
-o smtpd_soft_error_limit=1001
-o smtpd_hard_error_limit=1000
-o smtpd_client_connection_count_limit=0
-o smtpd_client_connection_rate_limit=0
-o receive_override_options=no_header_body_checks,no_unknown_recipient_checks
-o local_header_rewrite_clients=
```

Restart Postfix first:

```
$> postfix reload
```

Now two settings here need some explanation. First the **receive_override_options** are set to **no_address_mappings**. This disables all address mappings. Your virtual aliases for examples are not considered at first. Then the email is sent to **smtp-amavis** and in the end returned to the **127.0.0.1:10025** service that sets a lot of options. One of those options is the **receive_override_options** again. But this time the **no_address_mappings** setting is left out. So at this stage Postfix finally checks your virtual aliases. Sounds complicated? Well, it has to be done like this or otherwise your aliases would be evaluated twice which leads to mails sent twice. The other options are used to disable checks that Postfix has already done during the first stage.

Note

Setting **receive_override_options=no_address_mappings** makes Postfix not consider aliases any more. So if you wish to disable AMaViS as a content filter then you must not set this parameter either.

Another tiny caveat is that the user "clamav" must be a member of the system group "amavis" so that the two services are allowed to talk to each other:

```
$> adduser clamav amavis
$> /etc/init.d/clamav-daemon restart
```

And another issue to take care of: AMaViS tries to find out whether a certain email is incoming (sent from the internet to your domains) or outgoing (sent from your system to the internet) by looking at the **@acl_local_domains** setting. You need to tell AMaViS where to check if a certain domain is one of your destination domains. Edit the **/etc/amavis/conf.d/50-user** file and before the "1;" enter these lines:

```
@lookup_sql_dsn = (
    ['DBI:mysql:database=mailserver;host=127.0.0.1;port=3306',
     'mailuser',
     'mailuser2007']);

$sql_select_policy = 'SELECT name FROM virtual_domains WHERE CONCAT("@",name) IN (%k)';
```

The **@lookup_sql_dsn** defines how AMaViS can access your database. And the **\$sql_select_policy** sets the SQL query that is run when AMaViS wants to determine if the destination domain of the currently scanned email is one of your virtual domains. The **%k** is a list of strings that AMaViS expects to find. The actual query will look like this:

```
SELECT name
FROM virtual_domains
WHERE CONCAT("@",name)
IN (
    'john@example.com',
    'john',
    '@example.com',
    '@.example.com',
    '@.com',
    '@.')
```

This may look a bit weird. But in the end the string **'@example.com'** is searched for. Restart AMaViS:

```
$> /etc/init.d/amavis restart
```

Now try sending an email to john@example.com. If you examine the email headers of that mail you should find lines that got added by AMaViS:

```
X-Virus-Scanned: Debian amavisd-new at mymailserver
X-Spam-Score: 0
X-Spam-Level:
X-Spam-Status: No, score=0 tagged_above=-9999 required=6.31 tests=[none]
```

Your users will be able to filter out spam depending on this information. The **X-Spam-Status** line will be set to "Yes" if the spam score is above the required score (**\$sa_tag2_level_deflt**). The **X-Spam-Level** line contains a number of "*" that tell the spam score. If your email program looked for a **X-Spam-Level: ******* header then it would catch spam with at least a score of 5. A real-life example of caught spam:

```
X-Spam-Status: Yes, hits=16.0 tagged_above=-9999.0 required=6.31
  tests=BAYES_99, FORGED_MUA_OUTLOOK, MSGID_FROM_MTA_ID,
  RCVD_IN_BL_SPAMCOP_NET, UNDISC_RECIPS, URIBL_OB_SURBL, WORK_AT_HOME
X-Spam-Level: *****
X-Spam-Flag: YES
```

All incoming emails should now be tested for viruses and spam. Try that. Spamassassin comes with a spam sample containing the *GTUBE* (Generic Test for Unsolicited Bulk Email) that contains a signature that is supposed to trigger spam filters - just like EICAR.COM for virus scanners. Send John that spam email:

```
$> sendmail john@example.com < /usr/share/doc/spamassassin/examples/sample-spam.txt
```

Among other information your **/var/log/mail.log** will now contain a line being written by AMaViS:

```
amavis[13001]: (13001-02) Passed SPAM, <...> -> <john@example.com>, ...
```

So the email was detected as spam and passed through to John. Very well. Finally fix the permissions of the AMaViS configuration files so that no unauthorized user can read the database password:

```
$> chmod o= /etc/amavis/conf.d/50-user
```

STEP 11: LEARNING SPAM AND HAM

SpamAssassin is the software that is supposed to tell spam from regular emails ("ham"). It comes with a large number of tests that are applied to emails. It also queries real-time blacklists on the internet. But in addition to the fixed tests there is the *Bayesian* filter. It uses conditional probabilities to analyse the words used in the email mathematically. So it can derive if an email is likely to be spam. This filter can be trained by telling it what *you* think is ham or spam. The more ham and spam emails you show it the better the spam detection will become. You will need to have trained at least 200 ham mails and 200 spam mails or the Bayes filter will not be taken into account. The configuration described here will not distinguish ham and spam per-user but rather keep a global Bayes database. Results may be better if every user had their own database and that's actually possible but currently not used here.

As the 'amavis' user is the owner of the **/var/lib/amavis/.spamassassin** directory that is used to store the Bayes database and the 'vmail' user is used to store the users' mailboxes we have a permissions problem here. Unfortunately it cannot be easily solved by adding the 'amavis' user to the 'vmail' group. The permissions on **/home/vmail** do not allow group write access. So currently the spam learning is done by the 'root' user which may be a security problem.

A script that will teach SpamAssassin ham and spam can be run once a day by the 'root' user. Example for a script that you can run through crontab:

```
#!/bin/bash -e
```

```

SADIR=/var/lib/amavis/.spamassassin
DBPATH=/var/lib/amavis/.spamassassin/bayes
SPAMFOLDERS="\
    /home/vmail/well-known-customer.com/fred/.spam/cur \
    /home/vmail/spamtrap.org/jeanne/.spam/cur \
    "
HAMFOLDERS="\
    /home/vmail/spamtrap.org/jeanne/.trash/cur \
    "

for spamfolder in $SPAMFOLDERS ; do \
    echo Learning spam from $spamfolder ; \
    nice sa-learn --spam --showdots --dbpath $DBPATH $spamfolder
done

for hamfolder in $HAMFOLDERS ; do \
    echo Learning ham from $hamfolder ; \
    nice sa-learn --ham --showdots --dbpath $DBPATH $hamfolder
done

chown -R amavis:amavis $SADIR

```

You can see what AMaViS has recorded into its database by running:

```
$> sa-learn --dbpath /var/lib/amavis/.spamassassin/bayes --dump magic
```

The result will look like:

```

0.000      0      977      0 non-token data: nspam
0.000      0     27866    0 non-token data: nham
[...]

```

The *nspam* value is the number of spam emails SpamAssassin has seen. And *nham* is the number of ham emails being learned.

Note: this way you are training AMaViS' *global* Bayesian database. So it applies to all emails that enter your system. Make sure you don't learn spam mails from unreliable or untrusted users. They can ruin your spam detection ratio by putting messages in their 'spam' folder that are not really spam. Just because a user never wants to get email from his ex-girlfriend again does not make it spam for everybody. So be careful about the mails you feed to the global bayes database. Also do not make SpamAssassin learn emails that users just 'forwarded' to you. The header information would be gone that contains much of the information that helps SpamAssassin recognize spam. Make sure they use the 'bounce' feature or send the spam email as an attachment. Your best choice is to take the email messages directly from the user's maildir directory.

STEP 12: POPULATE AND ADMINISTER THE USERS IN THE DATABASE

Now that the setup is working with your test entries it is now time to fill the database with real

data.

Converting your database from the Sarge tutorial

If you followed the previous tutorial for *Sarge* then you can simply use the [conversion script](#):

```
$> wget http://workaround.org/articles/ispmail-etch/dbconvert.py
$> chmod +x dbconvert.py
... edit the db_sarge and db_etch variables in the script ...
$> ./dbconvert.py
```

That should convert your data from a Sarge-based database layout to the database layout suggested here.

Ronny Tiebel's PHP administration frontend

Of course you need a convenient user interface to administer the data in the database. Many readers offered to write a web interface to simplify the administration of domains, accounts and aliases. Robert Klikics published a [PHP script](#) written by his apprentice Ronny Tiebel.

Peter Gutwein's PHP administration frontend

Another PHP based web interface to administer user accounts was written by Peter Gutwein and is available at http://www.grs-service.ch/pub/grs_mminstallation.html

OPTIONAL FEATURES

Offering webmail access

This step is optional but most email service providers offer their users a way to read emails in a browser. Luckily this is easy. The package you need is called "squirrelmail" - a web mail system that can use any IMAP server - just like the Dovecot IMAP server we use here. To set it up you first need to add its configuration to your Apache configuration:

```
$> ln -s /etc/squirrelmail/apache.conf /etc/apache2/conf.d/squirrelmail.conf
$> apache2ctl restart
```

You need to change one configuration option at least so please run the configuration utility:

```
$> squirrelmail-configure
```

Select option 3 (Folder Defaults) and set option 1 (Default Folder Prefix) to 'none'. You may also want to browse through the other menu options to e.g. set your organisation's name right.

Point your browser at <http://localhost/squirrelmail> and you should see the webmail interface. Login as '[john@example.com](#)' with password 'summersun' and you should be able to read John's email. It couldn't be simpler.

In case you are unhappy with the <http://my.server/squirrelmail> URL and prefer to use <http://my.server> instead you should read the *users will prefer a simple URL* section in the

/etc/squirrelmail/apache.conf file.

Sieve: Filtering out spam

You already have a fully functioning mail server that even tags spam. But so far you left the actual task of sorting out the tagged spam to the user. We can do better by setting up server-side filters. The task ahead is to move all spam emails to a "Spam" folder of each user. We are using the raw power of the *Sieve* plugin which is a mail filter like *procmail* (which is not useful for virtual mailboxes). Sieve has a simple scripting language that allows us to forward all emails that are tagged by Spamassassin to a "spam" folder automatically. Somewhere above we defined a **protocol lda** and there set **global_script_path = /home/vmail/globalsieverc**. So that file will be the filter that is executed whenever an email is delivered to a virtual user. A basic recipe will do that. Create a **/home/vmail/globalsieverc** file containing these lines:

```
require ["fileinto"];
# Move spam to spam folder
if header :contains "X-Spam-Flag" ["YES"] {
    fileinto "spam";
    stop;
}
```

Send John another spam email:

```
$> sendmail john@example.com < /usr/share/doc/spamassassin/examples/sample-spam.txt
```

Then watch the **/home/vmail/dovecot-deliver.log** file. The last line should look like:

```
deliver(john@example.com): "2007-06-18 22:17:46 "Info: msgid=<GTUBE1.1010101@example.ne
```

This tells that the email was saved into the "spam" folder. If you fetch email you should see the email show up in that folder automatically.

Mailing lists with mailman

A frequently asked question is how to run mailing lists with virtual domains using the famous 'mailman' software. Actually this is not very hard. There are different ways to accomplish it. The simplest but least flexible way uses *local aliases*:

Using local aliases

This first approach will use the aliases that you usually insert into **/etc/aliases**. You just have to understand that virtual users cannot have piped aliases. Piped... what? Well, once you have created a mailing list with the **newlist** command you are requested to add certain aliases to your **/etc/aliases** file like **chitchat-admin**, **chitchat-bounces**, **chitchat-join** and so on that point to destinations that look like **"|/var/lib/mailman/mail/mailman post powerdns-debian"**. This means that the email is piped into the **mailman** program. Mailman is called and gets the email as input.

This works well for local domains but will fail if you put a piped alias like this into a virtual alias. That is because you do not have a system user whose user-id you could use to run this pipe. So

you will have to make a circuit and forward your virtual email address to a local email address where you can use piped aliases.

I recommend that you use 'localhost' as your local domain (**mydestination = localhost**) and forward each of the mailman aliases to the @localhost equivalent. So if you have a [chitchat-subscribe@virtual.domain](#) address you just forward it to [chitchat-subscribe@localhost](#) and use the /etc/aliases as suggested by 'mailman'. Perhaps someone comes up with a nifty solution to put the /etc/aliases into the MySQL database, too. Let me know if you have invented something. :)

Using transport maps (contributed by Michael Dürchner)

If you want to allow your customers/clients to have their own mailing lists then the above solution is not flexible enough. Consider using Postfix's powerful transport maps. They allow you to tell Postfix which transport (service) to use for certain email addresses or even whole domains.

Using transport maps with a regular expression

Using a regular expression lookup table you can forward emails where the recipient email address matches a certain pattern to mailman. The following is a static example of a transport mapping that forwards **mylist@example.com** and anything looking like **mylist-*@example.com** to the mailman service:

```
/^(mylist|mylist-.* )@example\.com$/      mailman:
```

Using transport maps with a database mapping

If you want to maintain your mailing list addresses in a MySQL table then use this **/etc/postfix/mysql-transport-mailman.cf** configuration file:

```
user = mailuser
password = mailuser2007
hosts = 127.0.0.1
dbname = mailserver
query = SELECT 'mailman:' FROM transports WHERE email LIKE '%s'
```

The **transport** table would consist at least of a column **email** that contains mailing list addresses. Use '%' in the address as a placeholder.

Use this config file by setting:

```
$> postconf -e transport_maps=mysql:/etc/postfix/mysql-transport-mailman.cf
```

Greylisting with Postgrey

Some administrators are enthusiastic about a mechanism called *greylisting*. How does it work? Usually MTAs have a queue so that they keep mails queued in case of temporary delivery problems on the recipient's side. SMTP distinguishes temporary errors that have a numeric starting with "4" and permanent errors starting with "5". If a 5xx error is returned then the sender is notified that the delivery failed. In case of a 4xx error though the delivery is tried again for a

few days. Some spammers do not use MTAs with a queue because they want to annoy thousands of innocents and do not want to care about queues. If your system rejected their first attempt to deliver an email they might not try again. This is what greylisting is about. Postfix rejects the first delivery attempt from an unknown system with a 4xx error. After five minutes the server Postfix will accept the next delivery attempt.

Greylisting has pros and cons. You will block off spammers that do not keep a queue. But you will delay legitimate email for at least five minutes what might annoy your users who may impatiently wait for an urgent email. Greylisting seems to become less useful with the time because most of the time spammers use compromised (cracked) systems to relay their emails. Those systems usually have a queue. It is your decision. Some administrators love it - some say it is useless to fight spam.

Fortunately using greylisting with Postfix is rather trivial. You would need to install the **postgrey** package first:

```
$> aptitude install postgrey
```

Right after its installation the **postgrey** daemon will start running in the background and listen for requests on TCP port 60000. To make Postfix use that *policy daemon* you have to add it to the **smtpd_client_restrictions** setting in your **/etc/postfix/main.cf**. Example:

```
smtpd_client_restrictions =  
    permit_mynetworks  
    permit_sasl_authenticated  
    reject_unauth_pipelining  
    reject_rbl_client bl.spamcop.net  
    reject_rbl_client dynablock.njabl.org  
    reject_rbl_client zen.spamhaus.org  
    reject_rbl_client list.dsbl.org  
    check_policy_service inet:127.0.0.1:60000
```

In this example a couple of RBLs (real-time blacklists) are checked to see if the sending system is a known spammer. If none of these criteria match then the **postgrey** policy daemon is asked about the sender's system and will reject the delivery for five minutes. Such delays will show up in your **mail.log** like:

```
postfix/smtpd[1234]: NOQUEUE: reject: RCPT from mail.example.com[10.20.30.40]:  
    450 4.7.1 <mail.example.com[10.20.30.40]>: Client host rejected:  
    Greylisted, see http://isg.ee.ethz.ch/tools/postgrey/help/yourdomain.html.html;  
    from=<spammer@example.com> to=<one.of@your.users> proto=ESMTP helo=<[10.20.30.40]>
```

Vacation / Autoresponder

Remo Fritzsche has kindly published his PHP-based [goldfish](#) autoresponder that can easily be used in conjunction with this setup.

Fetching and filtering using fetchmail and sieve

Mathias Geat wrote an article on [integrating fetchmail and sieve](#) that fits this setup.

Using Horde/Imp as a webmail interface

Oliver Ladner kindly provides instructions on [how to use Horde/Imp with this tutorial's SQL schema](#).

Removing old deleted mails

With IMAP you can mark emails as deleted and some email clients will not even show them any more. But the emails are still there and occupy space. Usually there is an option to purge all marked emails but many users do not care. So Michael Weisgerber suggests to run this command frequently via crontab to remove such emails:

```
find /home/vmail/ -name '*,ST' -ctime +7 | xargs rm -f
```

Dovecot renames all deleted emails so that they get a **,ST** added at the end of the filename. Adjust the parameter to **-ctime** as you like. In this example deleted mails older than 7 days are purged.

TROUBLESHOOTING

If you are having trouble receiving or sending email you may try these general steps:

1. Check your **/var/log/mail.log**. 93.7% of all problems cause a more or less clear error message there. :)
2. Run **postfix check**. No output means everything is well.
3. Ask on the **#postfix** IRC channel on irc.freenode.net. I usually attend there as *Signum* but please just ask and wait for people to help you. Especially read the hints in the channel's topic.
4. Ask on the [workaround chitchat mailing list](#) (english please)
5. This tutorial is meant for Debian "Etch". If you are trying this setup on any other distribution you are completely on your own. No, Ubuntu does not count as Debian because the package versions differ.
6. Sending an email directly to the author asking for help is the worst choice. Sending an email directly to the author giving suggestions or feedback is a good choice. :)

THANKS

This tutorial is maintained since the age of Debian Woody. And a lot of people translated it, contributed suggestions and helped finding bugs. Thank you:

- Alexander Stielau
- Axel Zöllich
- Cameron Jones from [vpslink](#) for sponsoring a virtual root server for testing this setup
- Christian Garling
- Christian Kurz
- Christof Gruber
- Colleen Hatfield
- Daniel Hackenberg
- Daniel Wladow
- Devdas Bhagat
- Eicke Kemm

- Florian Fritsch
- Jasper Slits
- Jesper Krogh
- Jon Cox
- Kay-Michael Voit
- Lucas Baltes
- Mario Duve
- Matthias Quade
- Michael Dürchner
- Michael Weisgerber
- Mikael Tokarev
- Oliver Ladner
- Patrick Blitz
- Patrick Jäger
- Pete Boyd
- Ricardo Arguello
- Simon Kammerer
- Thomas "Balu" Walter
- Tim Weippert
- Tobias Scheeper

You can find additional contributions that did not (yet) make it into the tutorial at the appropriate [contributions wiki page](#). Also thanks to all readers who sent feedback on the tutorial - both technical and emotional. Perhaps you are curious to read what [others think](#). If you like to support the author please consider donating:

