ISP-style Email Server with Debian "Woody" and Postfix 1.x

Submitted by Christoph Haas on Sun, 12/27/2009 - 00:32

*Note: Debian Etch is not the stable version of Debian any more. Consider reading the **newest tutorial** (http://workaround.org/ispmail) .*

Abstract

You have probably already seen web hosters who allow you to rent domains and receive email on these domains. Have you ever wondered how they actually handle these thousands of domains? There is surely nobody entering all these domains and aliases into a '*main.cf*' configuration file manually. Postfix offers two nice features that simplify such tasks:

- Virtual domains

  In addition to your local domain (which is probably the domain that is configured in */etc/defaultdomain*) you may receive email for other domains that are called virtual domains. There is no limitation on the number of domains you can receive email for.

- Database lookups

  You do not need to store all the information about your users and valid email addresses in text files. Postfix supports database lookups to common DBMSs like MySQL or PostgreSQL. This approach is especially charming as you may write a web administration GUI to manage the database. You may even allow your users to take care of their email accounts themselves.

This tutorial will introduce you to the basics of this kind of configuration. If you carefully follow all the steps in this document you will end up with a mail server that can handle thousands of domains and user accounts. These are some features you will get:

- POP3/IMAP access for your users
- Webmail access
- Virus scanning
- Spam prevention
- Secure mail relay access for road-warriors
- Easy domain administration

Although I will try to get you going quickly you will need to know a few things already:

- MySQL (creating a database, granting access for users and how SQL queries look)
- SMTP, POP3, IMAP (I assume you have a basic knowledge of these protocols)
- Basic Postfix configuration (you should be familiar with the 'main.cf' configuration file)
- Debian/Linux (you should know basic system administration tasks like installing software or editing text files)

[Important] This tutorial fits for Debian-Woody (3.x). If you are running newer Debian versions read the **Postfix 2.x tutorial** instead.

Table of Contents

What's new

As some of you may have noticed there have been some problems in the first version of this tutorial that may have led to error messages like "user not found" although all information was entered properly into the database. I read a lot of documentation and spent weeks on IRC until I completely understood the quirks of virtual domains in Postfix 1.x. That does not mean that you should not use Postfix 1.x (as shipped with Woody). Just some things behave differently. I decided to completely rewrite my first tutorial. I have added a few paragraphs that describe virtual domains, the general function of mappings and how MySQL can be used in Postfix. As a downside for those who have already set up their server following my old tutorial I changed the database structure fundamentally. Thus my good old Perl script (ispmailadmin) does not work with this database structure any more. I'm planning to rewrite the Postfix 2.x tutorial (for Sarge) too to adopt the structure. After that I'll probably help the **phpmywebhosting** (http://phpmywebhosting.sf.net/) project to make it use this structure. Thanks for your patience and the many feedback emails I received. Have fun with the new version.

The components

The whole setup depends on different software components that play together nicely. Let me clarify what each of them does:

- Postfix: Your MTA (Mail Transfer Agent) that receives emails via the SMTP (simple mail transfer protocol) and delivers them to different places on your hard disk.
- MySQL: The database server that stores the information to control the behaviour of postfix. It knows about users, domains, email forwardings and passwords.
- Courier: Courier is a standalone mail server just like Postfix. I will however just use itsPOP3/IMAP server component to let users access the mailboxes.
- PAM: The PAM (Pluggable Authentication Module) mechanism offers a way to authenticate users. It is used to verify the usernames and passwords of your users with the information in the databse.
- SASL (the Cyrus library): If your users are dialed in at another ISP (Internet Service Provider) while they are on the road they get an IP address outside of your network. Your mail server however only trusts local IP addresses. The SASL (Simple Authentication and Security Layer) adds authentication to SMTP and makes your mail server trust them.
- AMaViS: A mail virus scanner that works as a content filter in Postfix. It scans incoming mail for spam pattern (using the well-known spamassassin) or viruses.
- phpmyadmin: A web interface to manage your local MySQL databases. It's fare more comfortable than using the 'mysql' command from the command-line.

The big picture looks something like this:

What are mappings?

In short a mapping assigns one value to another. You probably know the file */etc/aliases* where you define forwardings for your local domain. A line there looks like this:

```
postmaster: root
```

This makes all mail to postmaster@your-domain be redirected to root@your-domain. The left side (here: "postmaster") is commonly called LHS (left-hand side) and the right side (here: "root") is called RHS (right-hand side) accordingly. You will find that these are common abbreviations when talking about mappings.

Hint: Usually map files do not have colons (':') on the left side. This is special to the aliases table for historical and compatibility reasons. Also the local aliases file special in that it is not compiled with **postmap** but the **newaliases** command. This is just a (bad) example. :)

If you are setting up Postfix the quick and dirty way you will typically start with text files like the one above. You just write the mappings into it and run **postmap filename** on it to convert the text file into a hash file called "filename.db". Then you can access this mapping using *"hash:filename"* in your Postfix configuration. You may notice that the default alias maps configuration looks like *"alias_maps = hash:/etc/aliases"* - just as an example. The *"hash:"* is called the lookup method.

In my setup I replace the text files by MySQL tables. This makes data handling a lot more flexible. But as database tables usually contain more than just two columns you will need to tell Postfix which column is meant to be the LHS and which is the RHS. This definition is stored in a text file like this:

```
user = service
password = DomAKg07
dbname = provider
table = virtual_mailboxes
select_field = email
where_field = mailbox
hosts = localhost
```

If a file like this would be saved as */etc/postfix/mysql_virtual_mailboxes.cf* you could use a mapping of *"virtual_mailbox_maps=mysql:/etc/postfix/mysql_virtual_mailboxes.cf"*. The LHS of the mapping is defined as *'where_field'* and the RHS is defined as *'select_field'*. In this example it would be a mapping of the *`email`* column to the *`mailbox`* column. The other fields in this definition file are *user* (the username that connects to the DBMS), *password* (the password for that user), *dbname* (the name of the database), *table* (the name of the table in that database) and *hosts* (the name of the server where the DBMS runs).

How virtual domains work

Let me begin with a brief introduction to virtual domains in Postfix 1.x (the newer Postfix 2.x handles them differently) because misconfiguration will cost you hair and time. There are two types of domains:

- Local Domains

  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 configure in */etc/passwd*). The mails will be delivered to /var/mail.

- Virtual Domains

  Your mail server can receive emails for additional domains called *virtual domains*. Virtual domains are very flexible. You do not need system accounts for every mail user (that means users do not need to be configured in the */etc/passwd*). So your system can handle thousands of email users easily. A mapping (see above) is used to save the information about the users. In my example I use MySQL for that reason.

  There is a special kind of virtual domains called *virtual mailbox domains*. Such a domain lets you receive email for users of that domain to mailboxes on your hard disk. You can still use the *virtual_maps* mapping to forward email to other mailboxes or external email addresses so not every user on that domain must actually have a mailbox but can also just have the email forwarded somewhere else.

A domain can either be virtual or local - never both! So if you decide you want your default domain be a virtual domain then remove it from the *mydestination* definition. Just leave it blank or set it to *"mydestination=localhost"*.

Four steps are run when you receive an email for a user in a virtual domain:

1. The virtual_maps mapping is checked for the domain name on the LHS. If it finds such an entry Postfix will know

that this domain is a virtual domain.
2. The virtual_maps mapping is checked for the email address on the LHS. If it finds such an entry Postfix will redirect the email to the recipient(s) mentioned on the RHS. (This is done recursively until no more entries exist.)
3. The virtual_mailbox_maps mapping is checked for the domain name on the LHS. If it finds such an entry Postfix will know that this is a virtual mailbox domain. So the email address belongs to a user having a mailbox on this server.
4. The virtual_mailbox_maps mapping is checked for the email address on the LHS. If it finds such an entry Postfix will deliver the email to the mailbox file (or maildir) mentioned on the RHS. (If the RHS ends in *'/'* a maildir structure will be used. Otherwise the destination is just a mailbox file.)

I recommend you also betimes read the original documentation about virtual domains in the VIRTUAL_README that shipped with the postfix-doc package and is found in */usr/share/doc/postfix/VIRTUAL_README.gz*.

Step 1: Install the needed Debian packages

Packages you will absolutely need:

- postfix (initial configuration: "Internet site")
- postfix-mysql
- postfix-doc
- mysql-client

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

- mysql-server

If you want to offer mail access using POP3/IMAP you need:

- courier-authdaemon
- courier-authmysql
- courier-maildrop
- courier-pop (for unencrypted POP3 access)
- courier-pop-ssl (for SSL-encrypted POP3 access)
- courier-imap (for unencrypted IMAP access)
- courier-imap-ssl (for SSL-encrypted IMAP access)

If you want to allow road-warriors to send email through your server using authenticated SMTP you also need:

- postfix-tls (for encrypted authenticated SMTP)
- libsasl-modules-plain
- sasl-bin
- libpam-mysql
- openssl (to create the certificate)

If you want to scan incoming email for viruses and spam:

- [see the section later in this document - you will need backports]

If you want to offer webmail for your users:

- sqwebmail (install as SUID root: yes)

Optional but useful packages:

- phpmyadmin (PHP interface for easy administration of MySQL databases)

Step 2: Create the database

You need to create a database first which holds the tables. If you are experienced in using MySQL you can of course do this work on the command line. However I would rather use *phpmyadmin* for MySQL database management. It is up to you.

Hint: when the mysql-server is first run you can access the database as user *'root'* with no password. You need to create a new database user for Postfix that has read-only access to just the provider database. If you are stuck here please read the appropriate documentation at mysql.com.

First create a database. I call it *'provider'* because I intend to do more than just email (which is not within the scope of

this document at the moment). Either do this in phpmyadmin or run this shell command.:

```
mysqladmin -u [username] -p -h [hostname] create provider
```

Step 3: Create the tables

After you have created the database you will need to create the database tables that will contain the control information for Postfix.

domains

The first table will contain just one boring column containing the virtual domain name. This table will need to have a row for each virtual domain. Just run this SQL statement to create it:

```
CREATE TABLE domains (
domain varchar(50) NOT NULL,
PRIMARY KEY (domain),
UNIQUE KEY domain (domain) )
TYPE=MyISAM;
```

forwardings

The table *'forwardings'* will be used to alias one email address to another. You can use this table for general redirections. (Hint: this even works for your local domain.) This is the SQL statement to create the table:

```
CREATE TABLE forwardings (
source varchar(80) NOT NULL,
destination TEXT NOT NULL,
PRIMARY KEY (source) )
TYPE=MyISAM;
```

users

Finally the *'users'* table contains information about your user accounts. Every user has a username and password for accessing the mailbox by POP3 or IMAP. As users tend to forget things (just look under the keyboard of your boss for his password) I decided to use the *'email'* address also as a login username. The email address is also be used for the directory name where emails for this users will be stored on the hard disk. So just these two fields are sufficient here. Just another SQL query to copy and paste:

```
CREATE TABLE users (
email varchar(80) NOT NULL,
password varchar(20) NOT NULL,
UNIQUE KEY email (email)
) TYPE=MyISAM;
```

Step 4: Create the database mapping definitions

As specified earlier in this document you need to tell Postfix where the control information is stored in the database. You need to create the following three text files in */etc/postfix* for that reason.

Postfix runs in a chroot directory (*/var/spool/postfix*) and cannot access any files outside that directory. Usually you talk to the MySQL database via its socket file */var/run/mysqld/mysqld.sock*. As you can see this file is out of reach for Postfix. So you can either try to move the socket file (which may lead to other problems) or use TCP networking. The latter means you are talking to MySQL through your network stack. Advantage: you do not need to care about the chroot restrictions. Disadvantage: everybody can talk to your MySQL server. So you are strongly recommended to take measures to limit access to your database server (e.g. by using netfilter rules). You have been warned. To enable TCP networking just comment out the line "*skip-networking*" from your */etc/mysql/my.cnf* file and restart MySQL.

mysql-virtual_domains.cf

This is a simple mapping of your virtual domain name (LHS) to the string *'virtual'* (RHS). This mapping is used for the transport table as it maps each virtual domain to the string *'virtual'* (which tells Postfix that this is a virtual domain). It is also used for the *virtual_maps* and *virtual_mailbox_maps* definition where just the LHS matters.

Create the file and fill the '...' with the information of your database user and database server name. Do not use "localhost" as the server name because Postfix would try to use the MySQL socket for communication. Instead use the name or IP address that belongs to your ethernet interface.

```
user = ...
password = ...
dbname = provider
table = domains
select_field = 'virtual'
```

```
where_field = domain
hosts = ...
```

mysql-virtual_forwardings.cf
This mapping reads the 'forwardings' table to provide a way to redirect email addresses. It simply maps the 'source' column to the 'destination' column. We will use it for the virtual_maps mapping.

```
user = ...
password = ...
dbname = provider
table = forwardings
select_field = destination
where_field = source
hosts = ...
```

mysql-virtual_mailboxes.cf
The next definition file deals with user mailboxes. It tells Postfix where to store email for a specific email address. We will use it later in the *virtual_mailbox_maps* mapping. Postfix will search the *'mailbox'* column (RHS) for a given *'email'* address (LHS). I just add a '/' to the mailbox field so Postfix will create a maildir structure instead of a mailbox file. This is needed for the Courier POP3 and IMAP services later.

```
user = ...
password = ...
dbname = provider
table = users
select_field = concat(email,'/')
where_field = email
hosts = ...
```

mysql-virtual_email2email.cf
Another quirk of virtual domains is the precedence of the *virtual_maps* mapping when using catch-all addresses. ("Catch-all" means you are using a forwarding like *"@domain.com"->"service@domain.com"* to catch all email that is sent to any user name in that domain.) If you would map *@domain.com* but not the other specific addresses *specific.user@domain.com* all email would be delivered to *@domain.com* even if the email address is listed in *virtual_mailbox_maps*.

This is a simple mapping for all email addresses in the *users* table to themselves to work around the above problem.

```
user = ...
password = ...
dbname = provider
table = users
select_field = email
where_field = email
hosts = ...
```

Make sure nobody but root can read these files. Otherwise everybody on your system could read your database access password in plain text. Run **"chmod o= /etc/postfix/mysql-virtual_*.cf"** to steal others the privileges. You also need to set the group of these files to postfix by running **"chgrp postfix /etc/postfix/mysql-virtual_*.cf"**.

Step 5: Create a vmail user

Your system can hold mailboxes for thousands of users. You probably do not want to assign a unique UID (user ID) to every user. So I recommend you create a pseudo-user who will become the owner of all mailboxes.

Just enter these lines in a root shell:

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

Step 6: Edit the main.cf

The */etc/postfix/main.cf* is the main configuration file for Postfix. I will describe the basic settings needed for virtual domains. Do not wipe away your main.cf and paste these lines into it. You will probably want to customise the configuration in other ways so I cannot be complete here.

Setting	Meaning
myhostname = ...	Make sure this is set to your fully qualified domain name.
mydestination = ...	List your local domains here seperated by commas. Do not list any virtual domain here.

mynetworks = ...	List the IP ranges here that are allowed to send email through your mail server. This is most likely your local network.
transport_maps = mysql:/etc/postfix/mysql-virtual_domains.cf	The transport mapping is used to determine how to handle a domain. We are just using the 'domains' table from the database that maps virtual domain names to the string 'virtual'. This is needed to tell Postfix this is a virtual domain.
virtual_maps = mysql:/etc/postfix/mysql-virtual_forwardings.cf mysql:/etc/postfix/mysql-virtual_email2email.cf	This is a general purpose redirection table. You can redirect one email address to another or even catch all mail for a domain and redirect it to one specific email address. The information is stored in the *'forwardings'* table. Every LHS email address is rewritten to the addresses on the RHS. If you have multiple destination addresses you can comma-seperate them in one entry. (Hint: the virtual domains must not be listed here or they would become *'virtual alias domains'*. This kind of virtual domains is just used for aliasing - you cannot receive email for such domains in a mailbox.) I also use the *mysql-virtual_email2email.cf* mapping to point the email to itself. That may sound stupid but is badly needed if you use catchall addresses. See the mysql-virtual_email2email mapping.
virtual_mailbox_maps = mysql:/etc/postfix/mysql-virtual_domains.cf mysql:/etc/postfix/mysql-virtual_mailboxes.cf	Another important mapping is the virtual mailbox maps. First it is a list of virtual mailbox domains (that is a domain that has user mailboxes on this server). Second it maps email addresses (LHS) to the location of the mailbox on your hard disk (RHS) relative to the *virtual_mailbox_base*.
virtual_mailbox_base = /home/vmail	This is the base path where the mailboxes of the users will be stored on your hard disk (see above).
virtual_uid_maps = static:5000	You should tell Postfix who shall be the owner of the mailboxes. This is the UID (user ID) of the owner (taken from */etc/passwd*). Set it to the UID of the *"vmail"* user you just created.
virtual_gid_maps = static:5000	The same for the GID (group ID).
smtpd_sasl_auth_enable = yes	Enable the authenticated SMTP feature.
broken_sasl_auth_clients = yes	Some broken mail clients like Microsoft Outlook use a deprecated way to detect if a mail server speaks authenticated SMTP. Make them happy.
smtpd_recipient_restrictions = permit_mynetworks, permit_sasl_authenticated, check_relay_domains	These restrictions are checked whenever a new email arrives at your mail server to check who is allowed to relay email. *permit_mynetworks* will allow everybody you configured in mynetworks. *permit_sasl_authenticated* will allow everybody from everywhere as long as they use authenticated SMTP. And finally relaying is allowed if the user wants to send email to one of the domains listed in *relay_domains* or to local domains.
smtpd_use_tls = yes	Encrypt the authenticated SMTP session using SSL
smtpd_tls_cert_file = /etc/postfix/smtpd.cert	The location of the SSL certificate for TLS (we will create it later)
smtpd_tls_key_file = /etc/postfix/smtpd.key	The location of the SSL private key for TLS

A quick test

Run **postfix reload** and **postfix check**. If you do not get any warnings this part is complete.

Step 7: Make Postfix understand authenticated SMTP (Auth-SMTP)

Imagine your users fetch their email using POP3. Now they need a way to send mails back through your mail server. For security reasons Postfix allows users defined in *mynetworks* to send emails. Usually your mail server will only accept mails for its own domains. If you allowed everybody to send email to every other domain you would provide a so called *open relay* that spammers abuse to send out their digital trash. So the logical way is to make remote users trusted by letting them provide their username and password. If the credentials are correct the user will be trusted like he has an IP in *mynetworks*. Most email cilents have this feature included.

Setting up authenticated SMTP is quite easy. The only pitfall is that Debian runs Postfix in a chroot'ed environment in */var/spool/postfix* by default.

Tell Postfix to use PAM

Postfix cannot directly access MySQL tables in version 1.x. So using MySQL tables for SMTP authentication means using PAM (pluggable authentication modules). PAM is used for about everything that needs authentication - even loggin into your system from the console. The second step is to make SASL use PAM for authentication. This is easy. Create a directory */etc/postfix/sasl* and set the password check method to PAM:

```
mkdir /etc/postfix/sasl
```

```
echo "pwcheck_method: pam" > /etc/postfix/sasl/smtpd.conf
```

We also need to copy the /lib/security/pam_mysql.so to */var/spool/postfix/lib/security* to make it accessible for Postfix (chroot'ed, remember?):

```
mkdir /var/spool/postfix/lib/security
cp /lib/security/pam_mysql.so /var/spool/postfix/lib/security/
```

Tell PAM to use MySQL

All authentication configurations are stored in */etc/pam.d*. As Postfix runs chroot'ed the configuration is searched for in */var/spool/postfix/etc/pam.d*. You need to create that directory first. Then go there and put the following lines into a file called *smtp*:

```
auth required pam_mysql.so user=... passwd=... host=... db=provider table=users usercolumn=email passwdco
lumn=password
account sufficient pam_mysql.so user=... passwd=... host=... db=provider table=users usercolumn=email pas
swdcolumn=password
```

Make sure the PAM file is not readable for the whole wide world: **chmod o= /var/spool/postfix/etc/pam.d/smtp**. At the same time it must be readable for Postfix. So you should run **chown root:postfix /var/spool/postfix/etc/pam.d/smtp**.

Woody's pam_mysql.so can only handle passwords up to 16 characters in length. If you used a longer password the authentication would fail.

Use TLS to encrypt SMTP traffic

An important step is to encrypt the SMTP session. Otherwise the username and password would be transmitted in a very insecure way. So I encourage you to encrypt that communication using TLS. TLS is short for Transport Layer Security (RFC2246) and in short terms uses SSL (Secure Socket Layer) which encrypts the mail connection between the road-warrior and the mail server.

First you will need an SSL certificate. If you don't want to pay for one from your favorite trustcenter you can well use a self-signed one. (Personal note: I wonder how paying for something makes it more trusted.) The only drawback: the mail clients does not know about your CA (certificate authority) and will spit out a warning to the user. Either tell the users to ignore the warning or let them install the certificate on their computers.

For a certificate that is valid for one year for the hostname smtp.domain.tld you would type this:

```
openssl req -new -outform PEM -out smtpd.cert -newkey rsa:2048 -nodes -keyout smtpd.key -keyform PEM -day
s 365 -x509
```

You will then be asked a few question about the fields of the certificate. It does not matter what you enter. Just fill the fields. One exception though - the "Common Name" must be the hostname of your mail server. Example session:

```
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 email services
Organizational Unit Name (eg, section) []:Master of Disaster
Common Name (eg, YOUR name) []:smtp.domain.tld
Email Address []:postmaster@domain.tld
```

After a short moment you will get two files: "smtpd.key" (the private key file) and "smtpd.cert" (the certificate). Move these two files into */etc/postfix*.

Make sure at least the key file is not readable for the whole wide world: **chmod o= /etc/postfix/smtpd.key**

A quick test

Run **postfix reload** to restart Postfix. Run **telnet localhost 25** and enter *EHLO anywhere.org*. You should find a line reading *250-STARTTLS*. Bingo - TLS is running.

Step 8: Configure the POP3 / IMAP service

You already set up a large part of the configuration. However your users will still be unhappy as they cannot reach they mailboxes. So it is time to configure the POP3 or IMAP service. First you need to edit the file */etc/courier/authdaemonrc* and change the directive authmodulelist to *"authmysql"* like this:

```
authmodulelist="authmysql"
```

Then you need to define the fields of the MySQL database table in */etc/courier/authmysqlrc* like this:

```
MYSQL_SERVER ...
MYSQL_USERNAME ...
MYSQL_PASSWORD ...
MYSQL_PORT 0
MYSQL_DATABASE provider
MYSQL_USER_TABLE users
#MYSQL_CRYPT_PWFIELD (comment this out)
MYSQL_CLEAR_PWFIELD password
MYSQL_UID_FIELD 5000
MYSQL_GID_FIELD 5000
MYSQL_LOGIN_FIELD email
MYSQL_HOME_FIELD "/home/vmail"
MYSQL_MAILDIR_FIELD concat(email,'/')
#MYSQL_NAME_FIELD (comment this out)
```

Careful - the authmysqlrc is very picky. Make sure you have not used TABs instead of spaces and that there are no trailing spaces on the lines. Do not forget to restart the authdaemon process using **/etc/init.d/courier-authdaemon restart**

A quick test

Try to reach the POP3 service by running **telnet localhost pop3**. You should get a *"+OK Hello there."*. Voila - your users should be happy now. :)

You cannot fetch emails from a mailbox unless at least one mail has been sent there. Users would get cryptic error messages. So I recommend sending a welcome email to new users.

Step 9: Test your setup

Congratulations. The configuration part is done. Now comes the more practical part. We will now create the database entries for your first domain so you can test to receive an email for the virtual domain. Please create these rows in the appropriate database tables:

domains

Column	Value
domain	virtual.test

users

Column	Value
email	user@virtual.test
password	secret

This means we have one domain called "virtual.test" and one user whose email address (and also the username) is "user@virtual.test". The password for this user is "secret". As you do not have an MX entry (mail exchanger - part of the DNS zone) you need to 'deliver' the email manually. Establish an SMTP connection to your mail server (telnet servername 25) and enter the SMTP commands written on the right side:

Server	You
`220 myserver ESMTP Postfix (Debian/GNU)`	
	`ehlo workaround.org`

`250-mailtest`
`250-PIPELINING`
`250-SIZE 10240000`
`250-VRFY`
`250-ETRN`
`250-STARTTLS`
`250-AUTH LOGIN PLAIN`
`250-AUTH=LOGIN PLAIN`
`250-XVERP`
`250 8BITMIME`	

	`mail from:<test@workaround.org>`

```
250 Ok
                                                    rcpt to:<user@virtual.test>
250 Ok
                                                    data
354 End data with <CR><LF>.<CR><LF></LF></CR></LF></CR>
                                                    This is a test email.
                                                    .
250 Ok: queued as ABC1D1C123
                                                    quit
221 BYE
```

If the server responded like in the example dialog above then the email was at least received. In the log file */var/log/mail.log* you should find a section like this:

```
Jul 16 20:03:06 myserver postfix/smtpd[2692]: EBC1D1C202: client=mailtest[127.0.0.1]
Jul 16 20:04:04 myserver postfix/cleanup[2697]: EBC1D1C202: message-id=<20040716180306.EBC1D1C202@mailtes
t>
Jul 16 20:04:05 myserver postfix/qmgr[2567]: EBC1D1C202: from=<test@workaround.org>, size=347, nrcpt=1 (q
ueue active)
Jul 16 20:04:05 myserver postfix/virtual[2704]: EBC1D1C202: to=<user@virtual.test>, relay=virtual, delay=
59, status=sent (maildir)
Jul 16 20:04:39 myserver postfix/smtpd[2692]: disconnect from mailtest[127.0.0.1]
```

If you read "status=sent (maildir)" then the email has been successfully delivered. Run the command **find /home/vmail** to see all directories and files there. It should look like this:

```
/home/vmail/user@virtual.test
/home/vmail/user@virtual.test/tmp
/home/vmail/user@virtual.test/cur
/home/vmail/user@virtual.test/new
/home/vmail/user@virtual.test/new/1090001045.2704_0.myserver.workaround.org
```

Everyting worked like I described? Great. Then as a last test you may try to fetch the email from your mail client via POP3 or IMAP (depending on what service you installed). The username for fetching email is always the email address ("user@virtual.test") and the password is "secret" in this test case.

Step 10: Populate your database

Now that the first test has succeeded you will want to configure the datase for your own domains and users. Let me explain what you need to insert into the database:

For every new domain...
Insert the domain into the *'domains'* table.

For every new user...
Insert a new row into the *'users'* table containing the email address and the password (in plain text).

For every new forwarding...
Insert a new row into the *'forwardings'* table containing the source (the address you send mail to) and destination email address (the address the mail gets forwarded to). If you have multiple destinations (like a poor-man's mailing list) you may list all email addresses in one row just seperated by commas. Hint: this table is used on every email that passes your system. So you can even redirect local mail addresses.

Examples:

source	destination	Effect
postmaster@my.domain	philip@my.domain	Redirect emails for postmaster to philip.
@my.domain	john@my.domain	Forward all email to email addresses in the domain my.domain to john. This does not apply to users in the `*email*` table though (like tina@my.domain). So more specific users of a domain always have a higher precedence than these so called "catch-all" accounts.
@my.domain	@another.domain	This is a whole domain redirection. Every email address in the my.domain domain will be directed to the same user at another.domain. So julie@my.domain will be redirected to julie@another.domain.
jesper@my.domain	dilbert@my.domain,dilbert@gmail.com	Forward email that is sent to jesper@my.domain to the two email addresses listed on the right side. Both users get a copy.

Scanning incoming email for viruses (optional)

Introduction to AMaViS

The two most annoying things when using email are spam and viruses. Fortunately you can fight both using a software called AMaViS (A Mail Virus Scanner). AMaViS is an interface between Postfix, spamassassin (famous for its bayesian spam filtering capabilities) and any virus scanner (like ClamAV which is freely available). Do not get confused: AMaViS contains the spam filtering part but has no virus scanner built in. I suggest you install ClamAV, too. It is a free virus scanner that gets updated frequently.

AMaViS has a huge advantage over even most of the commercial solutions. It is called while someone is sending you an email - not just after the delivery is done. If a virus is found or the spam probability is high enough then the email will be rejected while you have the sending system still on the hook. You have probably noticed that most spam and virus emails are sent with faked sender addresses. So instead of first receiving the email and then bothering the wrong sender you stop unwanted mails on the door step.

Install the backports for amavis-new and clamav

Woody ships with a very outdated version of AMaViS that I discourage to use. Instead you should get the backport of amavis-new. Using a backport means installing a special software package that is designed for Woody but contains a newer version of the software. If you are running Debian on the i386 platform then you can use the backports from backports.org (http://backports.org/) just first need to add this line to your */etc/apt/sources.list*:

```
deb http://www.backports.org/debian/ woody amavisd-new clamav
```

This enables you to install these two packages from the backports web site. Then run **apt-get update** and **apt-get install amavisd-new clamav clamav-daemon zoo bzip2 unrar spamassassin**. (Hint: "unrar" and "zoo" can only be installed if you have "non-free" in your sources.list.) The software packages should be installed now.

(Info: If you completely dislike backports you may instead use the *"postfix-amavis"* package.)

Configure AMaVIS

Please take a look at the */etc/amavis/amavisd.conf* file. These settings need to be taken care of:

Setting	Meaning
$mydomain = 'yourdomain.org';	Configure this string to your default domain. You will find a reference to $mydomain quite often in the configuration file. So you know where you find it.
@bypass_virus_checks_acl = qw(.);	If this line is commented out (has a '#' at the beginning of the line) then virus checks are enabled.
@bypass_spam_checks_acl = qw(.);	If this line is commented out (has a '#' at the beginning of the line) then spam checks are enabled.
@local_domains_acl = ("$mydomain");	This is a list of all domains that you are hosting. When an email passes your mail server this setting is used to determine if the mail is outgoing or incoming. Outgoing mail will not be scanned.
$final_virus_destiny = D_REJECT;	The default is to just throw away infected mails. I tend to reject them.
$final_banned_destiny = D_REJECT;	This defines how to handle emails with banned attachment types. Many MIME types like PIF, EXE, COM or DOC files are infected. You may choose later which types you do not like.

$final_spam_destiny = D_PASS;	This defines what to do with emails that are classified as spam. You may wonder why I suggest you let these emails pass. My users would get upset if I just threw away emails that a machine has found to be spam. In any case a line *"X-Spam-Status"* is added to the mail header. So users can configure they email clients to handle mails differently depending on the spam status.
$sa_tag_level_deflt = -1000;	AMaViS rates every mail with a "spam score". Usually spam has levels of 5-10. You will get *"X-Spam-Status"* header lines added to the email if the spam score is above this value. I set it to -1000 because I want all mail to get the header lines.
$sa_tag2_level_deflt = 5.0;	If the spam score is higher than this value then the mail will be flagged as spam. This is done by adding a header line *"X-Spam-Status: Yes"*.
$sa_kill_level_deflt = 10;	If the spam score is higher than this value AMaViS start to take action on the mail. The action is defined by $final_spam_destiny.
$sa_spam_subject_tag = '***SPAM*** ';	If you like to have the email subject altered to make spam mails more visible then you can have it rewritten. This string will be added if spam is detected.
@av_scanners = (...	This is a section where you can configure one or more virus scanner. Many common scanners are already preconfigured and just need to have the comment sign ('#') removed from the lines. You are also free to add your own command-line virus scanner here. For the beginning I suggest you just enable ClamAV and comment out all other entries. You may wonder why clamav is mentioned twice. Well, the @av_scanners' entry is about the daemonized version (clamd) and the @av_scanners_backup's entry runs the command-line version (clamscan).

Do not forget to restart the amavis service after you have changed the configuration file.

Tell Postfix to use AMaViS

Now that the amavis daemon is hopefully running in the background you still nedd to tell Postfix that you want to have all your emails scanned. Edit the */etc/postfix/main.cf* and add the following line:

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

Now you have all emails run through the "amavis:" service that is listening on port 10024 on your system. You also need to add the amavis filter to Postfix's configuration. Add these two sections to the */etc/postfix/master.cf*:

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

127.0.0.1:10025 inet n - 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
```

Due to **quirks (http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=255733)** in the file permissions you also need to add the user "clamav" to the group "amavis":

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

Finally run a **postfix reload** and **postfix check** to make sure you have still no errors in your Postfix configuration. All incoming emails should now be tested for viruses and spam. Look at your */var/log/mail.log* file for details. A great service for checking AMaViS is **testvirus.org (http://testvirus.org/)** . They are sending you harmless test mails with the EICAR virus test signature.

How does it work actually?

A last word to explain how Amavis works. Postfix receives your mail and pipes the email as defined in your "amavis" filter. AMaViS will then use the configured virus scanner to scan all attachments. If the attachments are archives it will even use the archive tools to unpack them. After scanning the files it will contact Postfix on localhost's port 10025 and re-inject the email into the delivery process. But this time it sets the "content_filter" option to nothing thus bypasses further content scanning. You will also see this behaviour in your */var/log/mail.log* file. First you will see a "relay=amavis" and then a "relay=virtual". If everything worked you will see a line like this in the */var/log/mail.log:*

```
amavis[677]: (00677-02) Passed, <my@test.address> -> <user@virtual.test>, Message-ID: <20040716231708.871
7C1C21E@myserver>, Hits: -
```

Offering webmail access (optional)

Now that you know how your users can get and send their emails via POP3, IMAP and SMTP you may wonder if there is a comfy way to give your users access to their mailbox using webmail. Luckily this is easy. The package you need is called "sqwebmail" - a web mail system that uses the Courier services and can be told (via PAMs) to use any authentication scheme. Install it using **apt-get install sqwebmail**. You will be asked if you want to run the "webmail" binary chroot. Yes you do! Nothing else needs to be done. The authentication will be done by the courier-authdaemon that you have already set up.

Now access your web server like http://hostname/cgi-bin/sqwebmail and you should get a login dialog. As usual use the email address as the username and the appropriate password from the *`users`* table.

Troubleshooting

Error messages

Log file	Error message	Meaning
/var/log/mail.log	...to=<user2@my.testdomain>, relay=none, delay=0, status=bounced (unknown user: "user2@my.testdomain")	Postfix knows that my.testdomain is a virtual domain. But the user account "user2@my.testdomain" was not found in the 'users' table.
/var/log/mail.log	...to=<foobar@local.test>, relay=none, delay=0, status=bounced (Name service error for domain.com: Host not found)	Postfix could not find an MX entry for the domain "domain.com".
/var/log/mail.log	...warning: connect to mysql server mailtest: Access denied for user: 'service@myserver' (Using password: YES)	Postfix tried to read from the MySQL database using the user service@myserver. However the combination of username and password have been rejected by MySQL.
/var/log/mail.log	amavis[677]: (00677-01) Clam Antivirus-clamd FAILED - unknown status: /var/lib/amavis/amavis-20040717T011641-00677/parts: Can't access the file ERROR\n	You probably did not run **adduser clamav amavis**.
Output from **postfix check**	postfix/postfix-script: warning: /var/spool/postfix/etc/hosts and /etc/hosts differ	A file in the chroot jail in */var/spool/postfix/etc* differs from the files in */etc*. Just restart the postfix service and these files will be copied into the jail.

MySQL debugging

In some cases it may happen that Postfix cannot even read from the MySQL database. If you suspect this you may want to take a look at the */var/log/mysql.log* file. It contains all SQL queries to the database.

Online troubleshooting

A lot of smart users can be found on the IRC channel *#postfix* in the freenode.net network. You are invited to join us. I usually attend there as "ChrisH". Please do not message me privately. Just join the channel and ask your question as precisely as possible. And though I am flattered that so many people use my tutorial I cannot guarantee any reaction times if you send me an email.

Thanks

A lot of people have mailed me for suggestions. I would like to thank the following people for providing valuable additions to this tutorial.

- Alexander Stielau
- Christian Garling
- Christian Kurz
- Christof Gruber
- Daniel Hackenberg
- Eicke Kemm
- Jesper Krogh
- Kay-Michael Voit
- Mario Duve
- Tim Weippert
- Ricardo Arguello
- Milbert Vanilbert

And of course thanks to my wife who showed a lot of patience with me.

About the author

I am a system administrator and programmer. In my nerdy spare time I work on web applications, Python and Ruby programs, write articles or learn new technology. On workaround.org you can find problems, solutions and hints on my findings and get help. Of course your feedback is as welcome as any donation. :) *<Christoph Haas>*