Secure Virtual Mailserver HOWTO

Postfix + OpenLDAP + Dovecot + Jamm + SASL + SquirrelMail

Authored (more or less) by: Peter Lacey; placey at wanderingbarque.com

Revision History:

Revision 2.17: Finally added handling for Postfix catch-all addresses back into this doc. Added quite a few more words regarding Postfix in general and alias processing in general. 2005-10-31

Revision 2.16: Updated to allow signing certificate to last for 10 years. Cyrus-SASL 2.1.19 (maybe 2.1.18 too) corrects the problem introduced in 2.1.17, HOWTO now reflects this. 2005-9-4

Revision 2.15: Update OpenLDAP ACLs to account for new syntax in version 2.2. 2005-8-8

Revision 2.14: Forgot to mention that the change_ldappass plugin for Squirrelmail requires the compatibility plugin. Corrected. 2005-2-2

Revision 2.13: Updated to reflect changes made in Cyrus-SASL 2.1.17 and later regarding expansion of the ldap_filter. Thanks to Lukasz Ostaszewski and Robert Banniza for pointing this out independently. 2005-1-20

Revision 2.12: Corrected some Firefox rendering issues (my fault, not Gecko's). Looks good in Firefox and Safari. I'll check Konqueror soon. Not testing IE. 2005-1-16

Revision 2.11: Corrected conflicting path names in the OpenLDAP section. References /usr/local/etc at one point and /etc at another, when I meant /etc in both cases. Thanks to John Glaze for pointing this out. 2005-1-11

Revision 2.10: Added a simple warning about OpenLDAP and whitespace. Thanks to Bill Adams for mentioning this. 2005-1-8

Revision 2.9: Added a section on creating a "dovecot-auth" user as the previous recommendation of using "nobody" won't work on Debian systems. Thanks to Peter Clark for pointing this out. 2005-1-7

Revision 2.8: This document is proving more popular than expected. Therefore, this revision adds a lot more explanatory text. More detail to follow. 2004-12-31

Revision 2.7: Removed the "accountsmap" section of the Postfix setup. If you think your users may use Postfix "catch-all" ("@domain.tld") aliases, then you'll want to add this back. See this thread in the Jamm documentation. Added a link to ZoneEdit's SMTP test utility. 2004-12-29

Revision 2.6: Changed the user under which the imap-auth process runs to be different than the user that runs the imap-login process. 2004-12-29

Revision 2.5: Added a SquirrelMail plugin to allow user to change their password from inside SquirrelMail. Corrected some minor typos. 2004-12-28

Revision 2.1: Added SSL support to SquirrelMail configuration. Corrected some minor typos. 2004-12-23

Revision 2.0: Added SquirrelMail based Web mail. 2004-12-13

Revision 1.5: The `useradd` command had an erroneous flag (-k) in it – removed. The saslauthd configuration file was misnamed; it's not `/etc/saslauthd` but `/etc/saslauthd.conf`. Detail added on the MECH=LDAP configuration directive for SASL. Thanks to Wayne Walker for catching these. 2004-11-18

Revision 1.4: Minor correction to the LDAP ACL. Thanks to Lee Thomson for catching this. 2004-11-12

Revision 1.3: Fixed *all* LDAP filters, a paren had been accidentally html'd out of existence. Added some detail on password hashing. Added a note about SquirrelMail and Dovecot's disable_plaintext_auth setting. Thanks to Ciro Mattia Gonano for pointing these out. 2004-11-06

Revision 1.2: Fixed link to Cyrus-SASL download. 2004-09-28

Revision 1.1: Dovecot user_global_uid and user_global_gid were shown with wrong values, was 504 is now 101. Updated the "compiling Postfix" section: 2004-09-27

Revision 1.0: Original Posting: 2004-09-19

Introduction

This guide is derived from many HOWTOs, in particular the Virtual Mailserver HOWTO written by Dave Dribin and Keith Garner of the Jamm team (of which this HOWTO is just a heavily edited version), plus not a small amount of hard won experience. It provides instructions on how to set up a secure, integrated mail server using Postfix, OpenLDAP, Dovecot-IMAP, Jamm, Cyrus-SASL and SquirrelMail.

In addition, I built this environement on a White Box Enterprise Linux system (a Red Hat Enterprise Linux clone) provided as a User-mode Linux virtual host by RimuHosting. I have added notes that are particular to this environment, but may also be useful to other White Box/Red Hat/Fedora users and possibly others as well.

This is not going to be a very verbose HOWTO, I don't have time for that. Conversely, it will be somewhat more than a dump of my config files. I began this project with the goal of providing mail services for friends and family (many of whom are self-employed). At the outset I knew little about email, now I know much more that I wanted to. However, I am far from an expert in all the intricacies of Postfix, Dovecot, etc. If there are errors, please let me know at placey-at-wanderingbarque.com. I do not, sadly, have time to help you resolve any issues with your configuration.

Requirements

The Requirements List

- Support for virtual domains and users: a single server with one IP address can host email for users at abc.com, and def.net, etc.
- No support for delivery to local users. It works, of course, but I've made no effort to do this right. No procmail, no IMAP support, etc.
- SMTP relaying for authenticated users.
- SMTP over SSL (TLS).
- Access to mail through IMAP over SSL only. NO POP access though that's trivially added.
- Centralized storage of domain and user information in LDAP.
- Web based interface for managing users and domains allowing limited access (passwords) by users.
- Forwarding (alias) entries for virtual domains. (This is a side effect of Jamm, I don't care)
- Web-based mail interface

To Be Done

- Spam filtering (probably Spam Assassin)
- Virus filtering
- Other meaningful protection

System Architecture

This sections describes which servers were used and how they all fit together at a system level.

Software Selection

Choosing one piece of software over another can often become the place of holy wars on the scale of emacs vs. vi. One could write a paper debating the merits of each of the SMTP servers out there, for example. This section briefly describes our choices for an SMTP server, an IMAP server, an LDAP server, and webmail software.

Postfix

Sendmail was preconfigured by my hosting provider, but everything I've heard and seen about Sendmail cautioned me against it due to its complexity and rickety security. Similarly Qmail looked like a non-starter. That basically left Postfix and Exim. I was leaning towards Exim, but shied away after reading its installation documentation. Thus, Postfix.

Dovecot

There are a number of IMAP servers available to Unix users. However, there seemed to be a lot of praise for the relatively new Dovecot IMAP (and POP) server. So I went that way over the incumbant Courier, etc.

OpenLDAP

Really, what other choice is there.

Jamm

Chosen mainly due to this HOWTO that I'm cribbing from :-) It's also simple and it works. I do want to make some simple changes to it, though.

Cyrus-SASL

SASL is required for SMTP AUTH. Cyrus-SASL seems to be the only open source choice.

SquirrelMail

Freshmeat will turn up any number of Web Mail interfaces (and I even tried one other than SquirrelMail, but it was too ugly), but SquirrelMail's the king. It works, it's proven, it does IMAP, and you can even make it look reasonable.

The Big Picture

In a "typical" Unix-based email system (if there is such a thing) there is a single domain, say myschool.edu, that hosts mail for local users. That is, users who also have a Unix account on the system. In these systems there is already a store of user information, normally the /etc/password file, but perhaps elsewhere. In environments such as this it is normal for the user's mail to be delivered into their home directory. Much of the documentation available on the web is geared towards such an environment.

What I want to accomplish here, is to have an email system that hosts any number of "virtual" domains (mycompany.com, myorganization.org, etc.) each with any number of "virtual" users, i.e. users who do not have an account on the system. In this case, I'll need a user store that is not part of the standard Unix user management system. I have elected to use LDAP, but this goal can be accomplished just as well with flat files, relational databases, or other storage systems. Furthermore, security is high on my list of features.

Figure 1 shows how the various packages I've chosen interact with one another. Postfix accepts incoming mail from SMTP and delivers it to the maildir mailboxes on the file system. It needs to have a list of valid users so it can bounce email for unknown users. It also needs to know where each user's mailbox is located on the file system so it can properly deliver messages. Postfix also accepts relay requests (sending mail) from authenticated users, and sends it on its way. Users are authenticated by passing login information to the SASL daemon which also needs a list of valid users. Dovecot provides remote access to the maildir (and mbox, but I'm not using mbox) mailboxes via the IMAP and POP protocols (though I'm not using POP either). It needs to have a list of valid users and some means to authenticate them so users can log into their account. It, too, needs to know where each user's mailbox is located on the file system so it can read their messages. Virtual user information is stored in an LDAP directory since LDAP provides a mechanism for searching for valid users, getting user information, and authenticating users. And Web access to the IMAP message store is given via Squirrelmail.

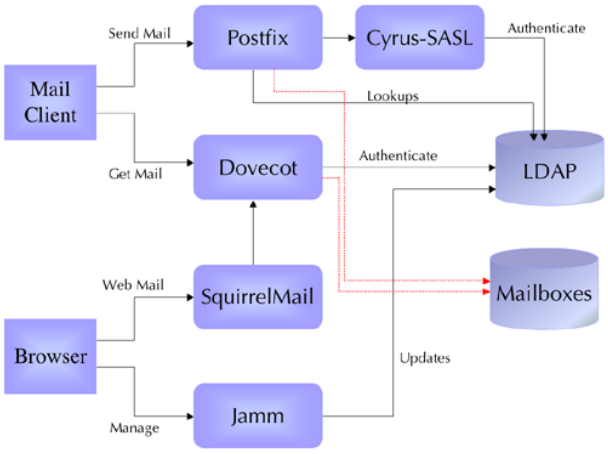

Figure 1. Overall Design

Mailbox Location

Unlike local users, there is no standard location for a virtual user's email. The Jamm guys chose to create a single Unix account, called `vmail`, whose home directory holds all the virtual mailboxes in a subdirectory called `domains`. This global mailbox location is stored in LDAP as is each user's personal mailbox. Postfix will retrieve the global and personal mailbox directories at runtime. Dovecot can too, but its assumptions are slightly different than normal, so, in actuality, Dovecot retrieves the user's mailbox location from its configuration files.

Each virtual domain has a subdirectory within the `~vmail/domains/` directory. For example, if you were hosting email for myschool.edu, then the mail for all users in this domain would be stored in `~vmail/domains/myschool.edu`. And the mail for "john@myschool.edu" would be stored in `~vmail/domains/myschool.edu/john/` in maildir format.

Implementation

This section describes how to implement a virtual mail solution. Not every little detail is covered, just what is needed above and beyond the "standard" installations.

Prerequisites

Here is the list of software that I used. It is likely that other, older and newer, versions will work, but I didn't test them. However, It's essential that both Postfix and Cyrus-SASL be at versions greater than 2.

The Software List

- White Box (Red Hat) Enterprise Linux 3
- Postfix 2.0.16
- OpenLDAP 2.0.27
- Dovecot 0.99.10.9

- Jamm 0.9.6
- Cyrus-SASL 2.1.15
- SquirrelMail 1.2.1.1

Preparing Your System

To prepare a Unix (like) system there are a few tasks you'll need to accomplish:

Pre-installation Preparation

- Create the `vmail` user and decide where you're going to store the virtual users email.
- Optionally, remove sendmail from the system.
- Determine your mail server's domain name.
- Determine your LDAP base.
- Create certificates for Postfix, Dovecot, and Apache (SquirrelMail).

Create the vmail User

Hint: It is not strictly necessary to create an actual user. It is only necessary to create a mailbox directory and change the owner and group to some ID that's is not going to be used by any real user, like 5000:5000.

Creating the `vmail` user is just like creating any other system account. You'll want to have a UID and a GID that is used for `vmail` alone. You may also want to set its home directory to the location you've selected for the storage area of the virtual users' email. In my system I used `vmail` as the user and group name. I also decided to store our virtual users email in `/home/vmail/domains`.

The following example works on a RedHat Linux distribution and results in a `vmail` user being created and an empty mail storage directory being created. I'm told that CentOS 4 (and therefore RHEL 4 and WBEL 4) requires the -g (group) flag.

```
# groupadd -r vmail
# useradd -m -r -d /home/vmail vmail
# mkdir ~vmail/domains
# chown vmail.vmail ~vmail/domains
```

Hint: If you elected not to create a real user, then skip the `groupadd` and `useradd` commands, and change the rest to something like `mkdir /home/vmail/domains; chown 5000.5000 /home/vmail/domains`.

Remove Sendmail

On the advice of somebody out there I completely removed (the pre-installed) Sendmail, just in case it got in the way of Postfix.

```
# rpm -e sendmail
```

Determine your mail server's domain name

If you have a static IP address, then you most likely already have a registered domain name. If, like me, you have a single host on the net, you may have given it the same name. However, if you want to use that name as a virtual host, you'll have some difficulties. For instance, if you already own the domain "whitehouse.gov," and your host is named "whitehouse.gov," and you want to have virtual users at "whitehouse.gov," then you're out of luck as Postfix will treat all users at "whitehouse.gov" as local. You can probably correct this by setting the appropriate Postfix variables ($myhostname, $mydomain), but you may consider renaming your host instead.

Furthermore, the domain name you use in your certificate should match the SMTP/IMAP server name used in your mail clients, otherwise the mail clients will complain. Finally, you'll probably want to use your domain name as the base name in your LDAP tree.

To neatly resolve all these issues, I elected to buy a new domain name, "whitehouse.net" (continuing the example), and rename my server accordingly. Here's how I renamed my machine:

- Modified `/etc/hosts`
- Modified `/etc/sysconfig/network`
- Modified `/etc/hostname`

I rebooted after this, but if nothing's yet running that cares about the hostname, you can probably just run `hostname --file /etc/hostname`.

Hint: You will need an MX record set up in the public DNS that points to your server. The MX record should not be the IP address of your machine. Instead it should be the name of an A record. That is, set up an A record, e.g. mail.mydomain.com to point to your IP, then set the MX record to be mail.mydomain.com.

Determine your LDAP base (root, suffix, whatever)

Do whatever you want here, but the current convention, and the one I used, is to break your domain name into components and reference them with the "dc" (domain component) attribute. That is, your base should be something like: `dc=whitehouse,dc=net or dc=mail,dc=whitehouse,dc=net`.

Summary

- Your server's name should *not* also be the name of any virtual host
- The domain name used in your cert should be the same as your server's DNS name
- You should probably use your domain name as the root of your LDAP tree.

Creating certificates for Postfix, Dovecot and Apache

If you want you can skip this step for now and return to it once you've got the unencrypted versions of Postfix and Dovecot running.

What we want to do here is create a cert and a private key that can be used for Postfix, Dovecot, and Apache (SquirrelMail over SSL). Technically, it's not necessary to sign this cert, but we will. This allows our users to install the signing (root) certificate in their user agents/operating systems. There are a number of HOWTO's on this subject, but you probably want to put a little thought into this first. What I wanted was to create a signing certificate (root CA certificate), a signed cert and a private key that were appropriately named. On Red Hat like systems certs are kept in `/usr/share/ssl`. I didn't want to use the existing directory structure below that, instead I create a directory called `hosting.example` (remember that's a pseudonym for what I really used), and created all my certs in there.

There are a handful of shell scripts in `/usr/share/ssl/misc` that wrap the OpenSSL utilities for manipulating certs, and we'll use these. (You can call OpenSSL directly for more fine grained control, if you want. It will avoid some post-creation manipulation of the certs.) But first we have to modify the script we want to use, `CA`.

By default the CA script will encrypt the certs it creates. Generally this is a good thing, but on a server it's not. This is because a process that uses the cert needs to supply a keyphrase to unlock it. If the server reboots on its own, then no one will be there to type in the key, and the server will never fully boot up. So make a copy of CA (call it CA_nodes) and edit it. Search for "# create a certificate" and add -nodes to the line below, the one that begins with $REQ. When your done with this search for "# create a certificate request" (just below) and do the same again.

Another change we want to make is to make sure the signing cert lasts for longer than the default year. Do this by searching for the line that reads 'DAYS="-days 365"' (the first non-comment line in my instance) and change 365 to some larger value - I used 3650, ten years.

When you're done it should look like this:

```
DAYS="-days 3650"

...

-newcert)
    # create a certificate
    $REQ -new -nodes -x509 -keyout newreq.pem -out newreq.pem $DAYS
```

```
    RET=$?
    echo "Certificate (and private key) is in newreq.pem"
    ;;
-newreq)
    # create a certificate request
    $REQ -new -nodes -keyout newreq.pem -out newreq.pem $DAYS
    RET=$?
    echo "Request (and private key) is in newreq.pem"
    ;;
```

Now, these scripts will ask for a lot of input. To make life easier, and to avoid errors in typing, this input can be defaulted to the contents of a particular file; `/usr/share/ssl/openssl.cnf`. It should already be there, lets edit it.

You'll need to change `countryName_default`, `0.organizationName_default`, `organizationalUnitName_default`, `commonName_default`, and `emailAddress_default`. In addition, I also changed the `default_days` of the `CA_default` setting from 365 to 3650 (1 year to 10 years). For clarity's sake, here's the relevant bits of my `openssl.conf` file:

```
...
[ CA_default ]

dir             = ./demoCA         # Where everything is kept
...
default_days    = 3650             # How long to certify for
...

[ req_distinguished_name ]
countryName                     = Country Name (code)
countryName_default             = US
countryName_min                 = 2
countryName_max                 = 2

stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = Massachusetts

localityName                    = Locality Name (eg, city)
localityName_default            = Anytown

0.organizationName              = Organization Name (eg, company)
0.organizationName_default      = My Hosting Company Name

# we can do this but it is not needed normally :-)
#1.organizationName             = Second Organization Name (eg, company)
#1.organizationName_default     = World Wide Web Pty Ltd

organizationalUnitName          = Organizational Unit Name (eg, section)
organizationalUnitName_default  = ISP

commonName                      = Common Name (eg, your name or your server\'s hostname)
# (Very Important, in order to keep mail clients and other user agents from complaining, this name must
# match exactly the name that the user will be entering into their client settings.  Whether that be
# domain.extension or mail.domain.extension or what.  It must be a valid DNS name pointing at your
# server.
commonName_default              = myhosting.example
commonName_max                  = 64

emailAddress                    = Email Address
emailAddress_default            = postmaster@myhosting.example
emailAddress_max                = 64
...
```

With this done we can create a signing (root CA) certificate. Go to the directory you created earlier; `/usr/share/ssl/hosting.example` , and run the `CA_nodes` script:

```
# ../misc/CA_nodes -newca
CA certificate filename (or enter to create) [hit enter]

Making CA certificate ...
Generating a 1024 bit RSA private key
..............................++++++
.............++++++
writing new private key to './demoCA/private/./cakey.pem'
Enter PEM pass phrase: [enter a password and remember it]
Verifying - Enter PEM pass phrase:
-----
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) [US]: [hit enter]
State or Province Name (full name) [Massachusetts]: [hit enter]
Locality Name (eg, city) [Anytown]: [hit enter]
Organization Name (eg, company) [My Hosting Company Name]: [hit enter]
Organizational Unit Name (eg, section) [ISP]: [hit enter]
Common Name (eg, your name or your server's hostname) [myhosting.example]: [hit enter]
Email Address [postmaster@myhostng.example]: [hit enter]
```

You now have a directory called `demoCA` in which is your signing cert, `cacert.pem`, and a number of other files and directories that makeup the (currently empty) database of certificates you've signed and revoked. Now we'll create a new certificate "request" (we'll have a proper cert once we sign it).

```
# ../misc/CA_nodes -newreq


Generating a 1024 bit RSA private key
..................................................................................++++++
.............++++++
writing new private key to 'newreq.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) [US]:[hit enter]
State or Province Name (full name) [Massachusetts]:[hit enter]
Locality Name (eg, city) [Anytown]:[hit enter]
Organization Name (eg, company) [My Hosting Company Name]:[hit enter]
Organizational Unit Name (eg, section) [ISP]:[hit enter]
Common Name (eg, your name or your server's hostname) [myhosting.example]:[hit enter]
Email Address [postmaster@myhosting.example]:[hit enter]

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:[anything will do, I used "certpass"]
An optional company name []:[hit enter]
Request (and private key) is in newreq.pem
```

The output of this is your certificate request, `newreq.pem` inside of which is your certifcate and private key (take a look, if you want). Now we'll sign this to generate a real certifcate.

```
# ../misc/CA_nodes -sign

Using configuration from /usr/share/ssl/openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem: [enter the passphrase used when creating the signing (CA) cert above]
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 1 (0x1)
        Validity
            Not Before: Sep 4 19:04:43 2004  GMT
            Not After : Sep 4 19:04:43 2014  GMT
        Subject:
            countryName               = US
            stateOrProvinceName       = Massachusetts

            [output elided]

Certificate is to be certified until Sep 2 19:04:43 2014 GMT (3650 days)
Sign the certificate? [y/n]:[hit "y"]

 1 out of 1 certificate requests certified, commit? [y/n]:[hit "y"]
Write out database with 1 new entries
Data Base Updated
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)

        [output elided]

-----BEGIN CERTIFICATE-----
MIIEGTCCA4KgAwIBAgIBATANBgkqhkiG9w0BAQQFADCBsTEL...

       [output elided]

-----END CERTIFICATE-----
Signed certificate is in newcert.pem
```

Your certificate is now in `newcert.pem`. There's just one thing left to do to make all this nice and clean, we want to extract the private key from the certificate request and into its own file. So edit newreq.pem, delete the certificate (all the lines between "Begin Certificate Request" and "End Certificate Request" inclusive, and save with a meaningful name, e.g. `ExamplePrivateKey.pem` (where "Example" is your domain name, like whitehouse). I also renamed `newcert.pem` to `ExampleCert.pem`.

In summary we now have three files we care about (we don't care about `newreq.pem` anymore):

- `demoCA/cacert.pem`: Our root CA certificate
- `ExampleCert.pem`: Our certificate for use
- `ExamplePrivateKey.pem`: Our private key

Because various processes, running as various users will need to access these certs, make sure they are readable by world (should be already). This is probably bad practice in the event that a local user (or a black hat who has local user privleges) steals them, but I only have one user on my machine, root. And if root gets owned, well that's it.

OpenLDAP

Installation

Not all of OpenLDAP was preinstalled on my system. White Box supports apt which I used to get OpenLDAP. You'll need all three packages. I highly recommend installing from your distro's package management system rather than compiling yourself.

```
# apt-get install openldap
# apt-get install openldap-servers
# apt-get install openldap-devel
```

Understanding the Jamm Schema

Configuring OpenLDAP for our needs requires Jamm's schema files so you should download the Jamm binary now. Put it anywhere and explode it.

```
# tar -zxvf jamm-0.9.6-bin.tar.gz
```

The Jamm schema introduces four new object classes and a handful of attributes. These are:

Object Class	
JammMailAccount	A user's mail account
Interesting Attributes	
mail	User's full email address and, consequentially, their login name. Ex: joe@myschool.edu
homeDirectory	User's home directory. Here it will always be /home/vmail/domains

mailbox	User's mail directory. Ex: myschool.edu/joe. The concatenation of homeDirectory and mailbox give the absolute path to a user's mail directory
cn	User's common name. Ex: Joe Blow
accountActive	Boolean telling whether account is active
delete	Boolean telling whether account has been deleted. Note Jamm never actually deletes anything, it just sets this flag
userPassword	User's password, preferably encrypted

Object Class	
JammVirtualDomain	A domain that's hosted on this system
Interesting Attributes	
jvd	A hosted domain name. Ex: myschool.edu
accountActive	Boolean telling whether this domain is active
delete	Boolean telling whether this domain has been deleted. Note Jamm never actually deletes anything, it just sets this flag

Object Class	
JammMailAlias	Aliases (other email addresses) that users may set up to redirect their mail
Interesting Attributes	
mail	The *receiving* email address. Ex: joe@myschool.edu
maildrop	Email address to redirect to. Ex: joseph@myschool.edu. Ex: joe@yahoo.com
delete	Boolean telling whether this domain has been deleted. Note Jamm never actually deletes anything, it just sets this flag
accountActive	Boolean telling whether this alias is active

Object Class	
JammPostmaster	Signifies that this account is a "Postmaster," kind of a domain level super user. Multiple people can be Postmasters in a domain.
Interesting Attributes	
roleOccupant	The distinguished name (dn) of the user who acts as postmaster for a domain. Can be more than one

Once you have built the base LDAP tree and added a few domains and users the structure will look like what's shown in figure 2.

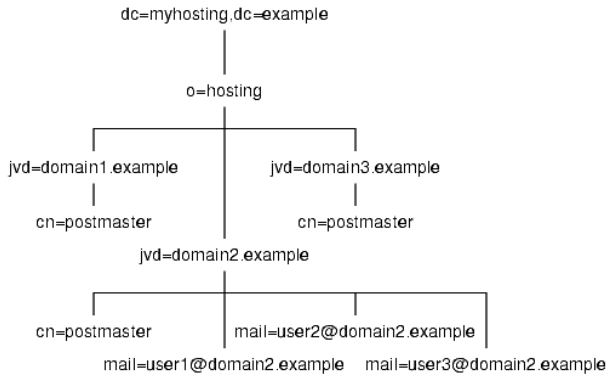

Figure 2. Jamm LDAP tree

Configuring slapd

All slapd configuration is in `slapd.conf`. On my box that's in `/etc/openldap`. On yours it might be in `/usr/local/etc/openldap`.

Adding Schemas

You need to make Jamm's schema file available, so copy the `jamm.schema` file in the Jamm distribution to the OpenLDAP schema directory, `/etc/openldap/schema/` in my case. `jamm.schema` depends on `cosine.schema` and `nis.schema`. Add these lines to `slapd.conf`. The first two may already be there.

```
include /etc/openldap/schema/cosine.schema
include /etc/openldap/schema/nis.schema
include /etc/openldap/schema/jamm.schema
```

Remember, these schemas might be in `/usr/local/etc/opennldap/schema` (or anywhere else) on your machine.

Setting the Password Hash Type

Passwords are (should be) encrypted when stored in LDAP. The default encryption mechansim is SSHA, but Dovecot doesn't support that. So set OpenLDAP's password hashing mechanism to CRYPT. I added the following line near the top of slapd.conf, right after all the includes.

```
password-hash {CRYPT}
```

Adding a Database Definition

Next, you need to set up a database definition. You can do this with the following lines:

```
database ldbm
directory /var/lib/ldap
suffix "dc=myhosting,dc=example"
```

The database directive specifies the back-end type to use. You should use LDBM as the back-end database. The directory directive specifies the path to the LDBM database. The suffix directive specifies the root suffix for this database.

Creating the Root User

The next few lines set up the "super user" or "root" account:

```
rootdn "cn=Manager,dc=myhosting,dc=example"
rootpw {SSHA}ea0sD475P32ASAlaAhR8kgi+8Aflbgr7
```

The `rootdn` entry has complete access to the database, which is why the password is stored outside the actual database. The password in `rootpw` should always be stored in hashed format. Do not store the password in clear text. To convert the clear text password `secret` to a hashed format, use the `slappasswd` command:

```
# slappasswd
New password: [enter some password and remember it]
Re-enter new password: [enter it again]
{SSHA}ea0sD475P32ASAlaAhR8kgi+8Aflbgr7
```

Take the output from `slappasswd`, and copy that into `slapd.conf`, as we did above.

Setting up Access Control

NOTE: The instructions that follow are for OpenLDAP 2.0.x. Most distributions now ship with 2.2. In OpenLDAP 2.2 the syntax for setting up ACLs changed slightly. Please read the comments associated with 2.0, but use the 2.2 syntax that's given immediately after.

The last part in `slapd.conf` is the access control. You can define your own policy, be here's the one Jamm follows that I've modified for Dovecot:

- The user can change any of their own attributes.
- Anyone in the postmaster group of the domain may change any user's attributes in their domain, including the password. This allows the postmaster to reset a users password if they forget it.
- The "dovecot" user can read passwords.
- Anonymous (non-authenticated) users may read all information, except the password attribute.

Access control statements are evaluated in order, so they should be defined from most specific to most general. Access to the password attribute, `userPassword`, is the most specific in our case, and hence it's specified first:

```
access to dn=".*,jvd=([^,]+),o=hosting,dc=myhosting,dc=example"
        attr=userPassword
    by self write
    by group/jammPostmaster/roleOccupant="cn=postmaster,jvd=$1,o=hosting,dc=myhosting,dc=example"  write
    by dn="cn=dovecot,dc=myhosting,dc=example" read
    by anonymous auth
    by * none
```

Please note, the line in red referencing dovecot is not in the original Jamm HOWTO, but is needed by Dovecot so it can read the `userPassword`. Typically an authenticating application tries to bind to LDAP as the user in question, a successful login thus validating the password. Dovecot does not yet support "authentication binds," so we must allow the Dovecot user read access to the user's password.

The `access to` line specifies what entries and attributes to which the following rules apply. The `dn` regular expression matches any entry in a domain of our hosting tree, and `attr` limits these rules to the `userPassword` attribute. Write access is granted to the user itself and anyone in the postmaster group. The dovecot user can read it. Anonymous users may only access this field when trying to authenticate. For all other cases, access is denied.

Next, all other attributes to entries in a domain's tree are specified:

```
access to dn=".*,jvd=([^,]+),o=hosting,dc=myhosting,dc=example"
    by self write
    by group/jammPostmaster/roleOccupant="cn=postmaster,jvd=$1,o=hosting,dc=myhosting,dc=example" write
    by * read
```

This access to line is very similar the previous one, except that there is no `attr` specification and no reference to dovecot. Hence, this matches all other attributes other than `userPassword`. Again, write access is granted to the user and anyone in the postmaster group. Everyone is granted read access.

Finally, we provide read access to all other elements in the database:

```
access to *
   by * read
```

Use these ACL statements if using OpenLDAP 2.2. Caution: Untested.

```
access to dn.regex=".*,jvd=([^,]+),o=hosting,dc=myhosting,dc=example"
        attr=userPassword
        by self write
        by group/jammPostmaster/roleOccupant.expand=\
            "cn=postmaster,jvd=$1,o=hosting,dc=myhosting,dc=example" write
        by dn="cn=dovecot,dc=myhosting,dc=example" read
        by anonymous auth
        by * none

access to dn.regex=".*,jvd=([^,]+),o=hosting,dc=myhosting,dc=example"
        by self write
        by group/jammPostmaster/roleOccupant.expand=\
            "cn=postmaster,jvd=$1,o=hosting,dc=myhosting,dc=example" write
        by * read

access to *
        by * read
```

Creating the Directory Tree

Now that slapd is configured, it's time to start adding data to the LDAP directory. We will use the command line tools that come with OpenLDAP and create LDIF files to modify the directory.

The first step is to create a base tree structure with our root node, the hosting organization, and an entry for the `rootdn`. Create a file called `base.ldif` (I put mine in `/etc/openldap` for safekeeping) with the following contents:

Caution: OpenLDAP is *very* sensitive to whitespace in LDIF files. Please make sure that there's no trailing spaces on any of these lines.

```
dn: dc=myhosting, dc=example
objectClass: top
objectClass: domain
domainComponent: myhosting

dn: cn=Manager, dc=myhosting, dc=example
```

```
objectClass: top
objectClass: organizationalRole
cn: Manager

dn: o=hosting, dc=myhosting, dc=example
objectClass: top
objectClass: organization
o: hosting


dn: cn=dovecot, dc=myhosting, dc=example
objectClass: top
objectClass: organizationalPerson
cn: dovecot
sn: dovecot
```

Note, the bit in red is not part of the original Jamm HOWTO, but is needed for Dovecot. This is the user Dovecot will bind to LDAP as.

Start up OpenLDAP. On RH/WB Linux you can use: `service ldap start`, or `/etc/init.d/ldap start`. It's probably similar on your system. Alternately you can start it directly with `slapd -u ldap -h ldap://127.0.0.1`.

Now use `ldapadd`, binding as the root user, to add this LDIF:

```
# ldapadd -x -D "cn=Manager,dc=myhosting,dc=example" -W -f base.ldif
Enter LDAP Password: [enter the LDAP password created earlier]


adding new entry "dc=myhosting, dc=example"
adding new entry "cn=Manager, dc=myhosting, dc=example"
adding new entry "o=hosting, dc=myhosting, dc=example"
adding new entry "cn=dovecot, dc=myhosting, dc=example"
```

Note, the Dovecot user requires a password. Add one like this:

```
# ldappasswd -x -W -S -D "cn=Manager,dc=myhosting,dc=example" "cn=dovecot,dc=myhosting,dc=example"
New Password: [enter a password for the Dovecot user and remember it]
Re-enter new password: [enter it again]
Enter bind password: [enter the LDAP password created earlier]
```

Hint: If you ever need to blast this database and start again from scratch, simply stop openldap, delete all the files in the LDAP directory (`/var/lib/ldap`), start openldap again, and repeat the above process.

Postfix

We'll only cover the sections of Postfix that pertain to the mail hosting. To deal with other parts of Postfix setup, please visit the Postfix web page.

Compiling Postfix with LDAP

Postfix was pre-installed on my system and linked with the appropriate libraries (LDAP, SASL, etc.). The following instructions are not guaranteed to work, but may be helpful. If at all possible install Postfix from a properly configured package, it's just easier. Detailed instructions on installing from source can be found here: http://www.postfix.org/INSTALL.html.

Download the Postfix source and untar it. Postfix veers slightly away from the ordinary `configure; make; make install` pattern of autoconf. In lieu of `configure`, with Postfix you make the makefiles. The default makefiles don't include LDAP or SASL, so you'll need to rebuild the makefiles to include them. To do this, execute the following command.

```
# make makefiles CCARGS="-DUSE_SASL_AUTH -DHAS_LDAP -I/usr/include" AUXLIBS="-lldap -llber -lsasl"
```

Note, this is how it would be done on *my* system. On yours the LDAP and SASL libraries are probably in `/usr/local/lib` and the header files in `/usr/local/include`. In which case the following will work for you.

```
# make makefiles CCARGS="-DUSE_SASL_AUTH -DHAS_LDAP -I/usr/local/include" AUXLIBS="-L/usr/local/lib -lldap -llber -lsasl"
```

Also note that the above commands are for SASL 1. If you want SASL 2 support, just change `-lsasl` to `-lsasl2`. Details are here: http://www.postfix.org/SASL_README.html.

Finally, Postfix does not include TLS support in the main code base. In order to use TLS, you need to patch the postfix source as documented here: http://www.aet.tu-cottbus.de/personen/jaenicke/postfix_tls/.

After you have rebuilt the makefiles and patched the source you can follow the normal Postfix compiling and installing instructions as documented in its INSTALL file. Which mostly amounts to `make; make install`.

At one point I tried to upgrade to Postfix 2.1.5 from source, but never succeeded. If I gave myself more time, I could have, but by the time I tried to do this my existing Postfix install was my primary mail server, and the longer I futzed with it, the more mail I was dropping. Anyway, I had all sorts of issues with OpenLDAP containing SASL 1 code and the Postfix I just built having SASL 2 code, and all sorts of library issues like that. These problems tend to show up as strange, unrelated errors in the log files. Let the compiler beware.

Understanding Postfix

Read this, it'll probably help. Postfix is composed of a number of components that run in synchrony. First, there's a mail transfer agent (MTA) called *smtpd*. The MTA accepts mail over the network using the simple mail transfer protocol, SMTP. The MTA is essentially a router, it determines whether incoming mail is ultimately destined for this server or not. If not, it relays it on (or, more commonly, refuses to accept it). If the message should be delivered to someone on this server, however, it hands it over to another process called *cleanup* that rewrites and sanitizes the message and drops it in the incoming queue. The MTA's job is now done.

Once a message is put in the queue, the queue manager passes it to a mail delivery agent (MDA) for ultimate delivery to a user's inbox or to another program for further processing. These MDAs and other programs are called "transports" in Postfix. The different transports are defined in the file `/etc/postfix/master.cf` (on my system). For our purposes there are two MDAs we want to know about: *local* and *virtual*. Both of these agents put email in the user's mailbox.

The MDA takes the verified mail that the MTA has put in the queue and delivers it. The *local* transport knows how to deliver mail for users that have accounts on the system. For virtual users there is a different transport named "virtual." The virtual agent, the one we use, is used when users do not exist on the system. The primary difference between the two is that *virtual* can get user information from remote data stores like LDAP, while local assumes the user store is system based. In fact, virtual is simply a hacked version of local. Much more information can be found on the Postfix architecure page.

Configuring Postfix

While configuring Postfix for this task, we'll be mostly concerned with `/etc/postfix/main.cf` (possibly `/usr/local/etc/postfix/main.cf` on your machine). For most of the Postfix configuration, you will configure things in a way that make the most sense for your site and you can follow the documentation contained in the Postfix source or on the Postfix web page. In this document, we'll talk about the settings that are unique to and/or affected by this setup. If any of the configuration examples shown below aren't explicitly attributed to a specific file, assume they would be found in `main.cf`.

Configuring LDAP Sources

Postfix user and domain information can be stored in a variety of places, i.e. sources. When using LDAP, you can create a source name out of thin air, then use that name as a prefix for the required LDAP variables. Later that same name will be used to tell Postfix that a certain piece of information can be found in LDAP by using these variables. For instance, if you are going to have Postfix search LDAP for domain information, the variable prefix might be "domains." Then variables will be defined as such: domains_server_host, domains_search_base, and so on.

You can easily define multiple LDAP sources. LDAP source parameters are documented in README_FILES/LDAP_README The parameter names follow the pattern of ldapsource_parameter. The LDAP source name is defined when it is first used. In main.cf, you'll need one LDAP source definition per each lookup.

Configuring the Source for Virtual Domain Information

```
domains_server_host = localhost
domains_search_base = o=hosting,dc=myhosting,dc=example
domains_query_filter = (&(objectClass=JammVirtualDomain)(jvd=%s)(accountActive=TRUE)(delete=FALSE))
domains_result_attribute = jvd
domains_bind = no
domains_scope = one
```

The first LDAP source definition is for looking up the virtual domains being hosted. By having this as an LDAP lookup, we'll be able to dynamically add new domains by adding new `JammVirtualDomain` entries to LDAP. `jvd` is "Jamm Virtual Domain," the attribute where Jamm stores domain names like whitehouse.gov.

We've named this LDAP source "domains". In our configuration, as specificed by the server_host line, our LDAP server is running on localhost. Our search base is the top of the hosting subtree we defined in our LDAP server, and according to scope we only want to search the directory level right under the base. We're querying for items where the jvd element matches the domain of the e-mail recipient as well as items that are of the `jammVirtualDomain` object class. We also check to make sure the `accountActive` attribute is set to true and that the `delete` attribute is set to false. As specified by bind, we do not want to bind/login to the LDAP server, we just want to do an anonymous search. Since we're only interested in whether there's a match, and not any particular value of the match, we just return `jvd` as the result_attribute.

Configuring the Source for User Aliases

```
aliases_server_host = localhost
aliases_search_base = o=hosting,dc=myhosting,dc=example
aliases_query_filter = (&(objectClass=JammMailAlias)(mail=%s)(accountActive=TRUE))
aliases_result_attribute = maildrop
aliases_bind = no
```

This LDAP source definition is for virtual *aliases*. We've named this LDAP source "aliases." We're querying for items where the `mail` element matches the email recipient as well as items that are of the `jammMailAlias` object class. We also check to make sure the alias is active by checking if the `accountActive` attribute is set to true. The destination of the alias is the `maildrop` attribute. Because we have not specified a scope in our ldap definition, it will perform the default search of the entire subtree under the base.

Aliases are a good way of having generic mail addresses delivered to one or more specific people. For instance, you can create an alias (easy when using Jamm) called sales@example.com, and have all the mail sent to that address actually delivered to bill@example.com and sue@example.com. Of course, the actual recipients may be in another domain; for instance, if Bill has left the company, you can delete his email account and create an alias of the same name, such that all mail sent to bill@example.com is forwarded to bill@someplacelese.com.

But possibly the least intuitive use for this feature is as a replacement for the user oriented .forward file. It turns out that it's the *local* mail delivery agent that knows how to process .forward files, *virtual* doesn't. Even though *virtual* is just a hacked version of *local*, during the hackery, apparently for security reasons, the ability to process a .forward file was removed. The upshot of this is that there's no easy way to allow for a user to specify that they want mail deliverd to their normal inbox *and* one or more external mailboxes. One possible approach is to use a different delivery agent that supports both LDAP and .forward functionality. Procmail won't do because, like local, it can't get user information from LDAP. Maildrop might work except the latest incarnation of Maildrop requires yet another daemon process to run in order to get to LDAP (and MySQL, etc.), and I simply don't want that. There are no other suitable delivery agents that I'm aware of.

However, the proper use of aliases can solve this problem. The trick is to create an alias of the name that the user will be known as to the outside world, say jane@example.com, then give that aliased user two or more *destinations*. One destination would be the email adress of the actual user on this server (that you also create), say jane.doe@example.com and the rest are the remote addresses to which mail should also be forwarded, such as jane@gmail.com. The user would have to set up her IMAP clients (including Squirrelmail) to have a from: or replyTo: set to the alias name (jane@example.com) and not the actual account name. Any mail sent directly to the actual user (jane.doe@example.com) won't get forwarded.

Configuring the Source for User Accounts

```
accounts_server_host = localhost
accounts_search_base = o=hosting,dc=myhosting,dc=example
accounts_query_filter = (&(objectClass=JammMailAccount)(mail=%s)(accountActive=TRUE)(delete=FALSE))
accounts_result_attribute = mailbox
accounts_bind = no
```

The accounts source is very similar to our aliases source. It's used by Postfix to look up actual users. The big difference here is that we're looking for entries that have an object class of `jammMailAccount` and we're interested in the `mailbox` attribute of the resulting match. We also check to make sure the account is still active by looking at the `accountActive` attribute and make sure the account is not marked for deletion by checking the `delete` attribute.

It's possible to use virtual aliases to define "catch-all" addresses, such as "@example.com -> mike@example.com." A catch-all address receives mail for every address in this domain that is not *also listed in the virtual alias list*. What this means is that if we have a catch-all address, it will indeed catch *all* email, even email destined for actual users on the system, unless those actual users are also listed in the alias list. If you use catch-all aliases, you can guard against this behavior by creating another (seemingly redundant) LDAP source that returns the email address (contained in a user's mail attribute) of all users, and force Postfix to use both this LDAP source and the aliases LDAP source when looking up virtual aliases. Here is that LDAP source:

```
accountsmap_server_host = localhost
accountsmap_search_base = o=hosting,dc=myhosting,dc=example
accountsmap_query_filter = (&(objectClass=JammMailAccount)(mail=%s)(accountActive=TRUE)(delete=FALSE))
accountsmap_result_attribute = mail
accountsmap_bind = no
```

This is identical to the accounts LDAP source except we are returning the mail attribute (email address) of a user rather than her mailbox location.

The Virtual Alias Maps

Now that the aliases LDAP source(s) have been defined, we need to let Postfix know to use it. This is taken care of using the virtual_alias_maps parameter in `main.cf`

```
virtual_alias_maps = ldap:aliases
```

If you are using catch-all addresses, and need to correct for Postfix's quirkly handling as just described, then the virtual alias maps should look like this instead:

```
virtual_alias_maps = ldap:accountsmap, ldap:aliases
```

When Postfix builds this mapping table it will include all actual users plus all aliases, keeping catch-all aliases from catching mail meant for legitimate users.

The Virtual Accounts

Telling Postfix about the virtual accounts is a bit trickier than the aliases. This is due to the fact that we need to define a lot of extra information about the virtual mail storage.

For this example, we assume that there is a `vmail` Unix account created that has a UID of 101, a GID of 101, and its home directory is `/home/vmail`. We will use the home directory of the `vmail` user as the place where we store our virtual mail repository. As before, add this to main.cf

```
virtual_transport = virtual
virtual_mailbox_base = /home/vmail/domains
virtual_mailbox_maps = ldap:accounts
virtual_mailbox_domains = ldap:domains
virtual_minimum_uid = 101
virtual_uid_maps = static:101
virtual_gid_maps = static:101
```

Most of the above is pretty straight forward, except for `virtual_transport`, `virtual_minimum_uid`, `virtual_uid_maps`, and `virtual_gid_maps`.

For virtual accounts, we want to use the `virtual` transport and set `virtual_transport` to specify this.

With the domains LDAP source defined, Postfix needs to be configured to use it. This is done by setting the `virtual_mailbox_domains` in `main.cf` to `ldap:domains`.

The Postfix documentation states *"[virtual_minimum_uid] specifies a minimum UID that will be accepted as a return from a virtual_uid_maps lookup. Returned values less than this will be rejected, and the message will be deferred."* Since we have decided that all mail for virtual accounts will be stored using the `vmail` Unix account, we set the `virtual_minimum_uid` to be the UID of `vmail`. Also, we set the `virtual_uid_maps` and `virtual_gid_maps` to a special static map and hard code it to the UID and GID of the `vmail` user. All of the parameters shown here are fully documented in README_FILES/VIRTUAL_README that comes with the Postfix source.

Other Postfix Settings

Many defaults are fine in this setup (myhostname, mydomain, etc.), but change them if you need to. In my case I also set (in main.cf):

```
inet_interfaces = $myhostname, localhost
```

This tells postfix to listen for connections from the outside world and from localhost. localhost is needed by SquirrelMail if nothing else.

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

Even though we are depending solely on the virtual transport, the local transport is apparently still active. This transport really wants to have an alias database of its own, and that's what these are. It seems safe to comment these out, *if, and only if,* you also comment out the local transport in the `master.cf` file (but I'm not sure how advisable that is). I elected to leave these intact and have postfix create the local alias database from the empty local alias maps file by runing the command: `newaliases` or `postalias /etc/postfix/aliases` (same thing). You'll probably need to do the same thing.

```
home_mailbox = Maildir/
```

Make Postfix use Maildir (one file per email) format instead of mbox (one big file)

.

Postfix setup is complete. You can start Postfix with the following command:`service postfix start`. If you don't have another email account to test this one with (like whoever@yahoo.com), then this service might be useful: http://www.zoneedit.com/smtp.html.

SMTP AUTH with SASL

The setup so far will allow a virtual user to receive mail and that's it. No virtual user can send (relay) mail (though local ones can), nor can any other server. We don't want servers to be able to relay, but you definitely want your users to. There are a number of inelegant ways to get this to happen, but the cleanest is to use SMTP authentication; making your users authenticate to Postfix, and allowing authenticated users to send mail.

Building SASL

To use SMTP AUTH you must also use SASL, an authentication protocol invented by Netscape. The most common FOSS implementation of SASL is Cyrus-SASL from Carnegie Mellon University. On my machine Cyrus-SASL was preinstalled, but it lacked LDAP support, so I downloaded the source and compiled that. You can get the source tarball here: http://ftp.andrew.cmu.edu/pub/cyrus-mail

Some of the defaults were not as they should be for a Red Hat like system, so I ran configure like this:

```
# ./configure CPPFLAGS=-I/usr/kerberos/include LDFLAGS=-L/usr/kerberos/lib --prefix=/usr --sysconfdir=/etc --mandir=/usr/share/man --with-ldap
# make
# make install
```

The important part is the "--with-ldap" flag (make sure you have the OpenLDAP development libraries installed as above). The `CPPFLAGS` and `LDFLAGS` may or may not be important. Dovecot needed them (more later), and I figured they couldn't hurt, so I used them here too. They basically point to the Kerberos development files which on my system were not in `/usr/lib` and `/usr/include`.

Configuring SASL

Cyrus-SASL requires a particular directory to keep it's runtime information. This directory will (probably) not be created for you. Run `saslauthd` from the command line and let it yell at you, then you'll know. You can create the asked for directory manually without problems. I used `/var/run/saslauthd`. Or rather, the pre-existing init script did by passing in the -m flag, but I concurred.

Cyrus-SASL also uses a config file that's not automatically created. In my case it's called `/etc/saslauthd.conf`. Create this file with the following self-explanatory contents:

```
ldap_servers: ldap://127.0.0.1
ldap_search_base: o=hosting,dc=myhosting,dc=example
ldap_filter: (&(objectClass=JammMailAccount)(mail=%u)(accountActive=TRUE)(delete=FALSE))

Important: If you are using Cyrus-SASL 2.1.17 (possibly 2.1.18, as well), then you must change the ldap_filter directive above to be as follows:
ldap_filter: (&(objectClass=JammMailAccount)(mail=%u@%r)(accountActive=TRUE)(delete=FALSE))
```

Finally, you must tell Cyrus-SASL that it is to use LDAP by passing `-a LDAP` to it at startup. There are two ways to do this (and you might find that it's already been done for you); you can add it to the init script or you can add it to a file read in by the init script. I chose the former, but it's up to you. Here's the relevant part of my init script (located at `/etc/init.d/saslauthd`:

```
```

```
# Source function library.
. /etc/init.d/functions

# Source our configuration file for these variables.
SOCKETDIR=/var/run/saslauthd
MECH=ldap
FLAGS=
if [ -f /etc/sysconfig/saslauthd ] ; then
        . /etc/sysconfig/saslauthd
fi

RETVAL=0

# Set up some common variables before we launch into what might be
# considered boilerplate by now.
prog=saslauthd
path=/usr/sbin/saslauthd

start() {
        echo -n $"Starting $prog: "
        daemon $path -m $SOCKETDIR -a $MECH $FLAGS
        RETVAL=$?
        echo
        [ $RETVAL -eq 0 ] && touch /var/lock/subsys/$prog
        return $RETVAL
}
```

Notice how the `MECH` variable is set to `ldap` and how it is later used with the `-a` flag when kicking off the daemon. (Also notice how the `SOCKETDIR` variable is set to the path of SASL's runtime directory.) Alternately, as you can see, you coul have added the `MECH` (and `SOCKETDIR`) variables to the `/etc/sysconfig/saslauthd` file which is sourced by this script.

Later, when you've actually added a user to LDAP, you can test your SASL configuration like this:

```
# testsaslauthd -u users_login_name -p users_password
```

For instance:

```
# testsaslauthd -u george@whitehouse.gov -p thisisasecret
0: OK "Success."
```

Configuring Posftix / SASL Environment

You may or may not need the following. My setup works both ways, however I'm leaving it in for safety. The premise is that every process that users SASL can have a SASL specific configuration file. In other words Postfix (not SASL) will look in `/usr/lib/sasl2` (note the "2"), for a file called `smtpd.conf`. On some systems (Debian? chrooted?) this file path may be `/etc/postfix/sasl`. Postfix will then learn a few things about SASL. What we're interested in telling Postfix is what mechanism SASL will use to look something up and what formats it will accept user information in. In short, create the file `/usr/lib/sasl2/smtpd.conf` (or `/etc/postfix/sasl/smtpd.conf`) and make it look like this:

```
pwcheck_method: saslauthd
mech_list: login plain
```

This will tell Postfix to contact the saslauthd daemon for authentication purposes, and keep Postfix from telling user agents that is supports, say Kerberos (which it may, but SASL/LDAP doesn't) when SASL only accepts "plain." (Or something like that.)

Configuring Posftix

Add the following Postfix directives to the end of `/etc/postfix/main.cf`:

```
# SASL support
smtpd_sasl_auth_enable = yes
smtpd_sasl_local_domain =
smtpd_recipient_restrictions = permit_mynetworks, permit_sasl_authenticated, check_relay_domains
smtpd_sasl_security_options = noanonymous
smtp_sasl_auth_enable = no
```

The first line is obvious. The second is very important -- `smtpd_sasl_local_domain` must be there (not missing or commented out) and it must be blank! The value of this variable is appended to the login name Postfix sends to SASL. Since our login names already have the domain component, using this would cause Postfix to send something like "george@whitehouse.gov@whitehouse.gov" or worse "george@whitehorse.gov@myisp.net." And if it's not there at all, bad things happen.

The `smtpd_recipient_restrictions` allow local users and users authenticated via SASL to send mail -- and nobody else (unless you have set up allowed relays, which, presumably, you haven't.)

The `smtpd_sasl_security_options` bit is obvious but important. The final variable, `smtp_sasl_auth_enable` refers to having this server authenticate to other servers, and we don't care about that.

SMTP over SSL (TLS)

Since we are using plain text logins we need to be able to encrypt them. Besides, there's no reason to let others sniff our mail either. Turning on SSL is pretty easy. You just have to create a few certs and then set a few variables.

How to create certs was detailed above. If you haven't done that part, you'll need to do it now. To enable Postfix to support TLS modify `/etc/postfix/main.cf` as follows (these settings won't be there by default, so just add them to the bottom):

```
# TLS Support

smtpd_use_tls = yes
smtpd_tls_auth_only = yes
smtpd_tls_key_file = /usr/share/ssl/hosting.example/ExamplePrivateKey.pem
smtpd_tls_cert_file = /usr/share/ssl/hosting.example/ExampleCert.pem
smtpd_tls_CAfile = /usr/share/ssl/hosting.example/demoCA/cacert.pem
```

```
smtpd_tls_loglevel = 1
smtpd_tls_received_header = yes
smtpd_tls_session_cache_timeout = 3600s
tls_random_source = dev:/dev/urandom
```

These settings should be more or less self-explanatory, although I don't know why Postfix needs the CA cert. You can play with the log level, but I found setting it to 3 generated a lot of LDAP/SASL noise in my log files.

Dovecot IMAP

Building Dovecot

Dovecot was not pre-installed on my system. It was available via apt-get, but not with LDAP support. This meant compiling from source, here's how:

```
# ./configure CPPFLAGS=-I/usr/kerberos/include LDFLAGS=-L/usr/kerberos/lib --prefix=/usr --bindir=/usr/bin --sbindir=/usr/sbin --libexecdir=/usr/libexec --
datadir=/usr/share --sysconfdir=/etc --mandir=/usr/share/man --with-ldap -with-ssldir=/usr/share/ssl # make # make install
```

In the above run of configure, the `--with-ldap` flag is the most important. But you must pay special attention to the output of `configure`. Even if the LDAP libraries are not found, Dovecot will still build and install! This may be legitimate, but Dovecot will fail to communicate with LDAP, and it may lead you (as it did me) to believe that your Dovecot build is good, and something else is keeping the communication from happening. In a similar vein, Dovecot wants to build against Kerberos, and silently continued even though it couldn't find the Kerberos libraries, which on my machine are in `/usr/kerberos/lib` (the header files are in `/usr/kereberos/include`) instead of `/usr/lib`. The `--with-ssldir` is used to tell Dovecot the base directory for certificates. It's not really important in our configuration, as we'll be setting the full path to our certs, but it might as well be accurate anyway. As for all the other directory flags, well, I would have liked to keep everything in the default `/usr/local` (and you probably do too), but previous installs of the non-LDAP, apt-get binary made me chose to imitate that and place things as you see above – your choice.

Creating the Dovecot Auth User

Dovecot's IMAP implementation is made up of several processes. One of these, imap-login, accepts incoming connections and should run as the "dovecot" user, which should have been created for you during package installation or during the `make install` step. The Dovecot authentication process, dovecot-auth, which authenticates users against some user store, should run, for security reasons, as some other user. It defaults to root, which would be necessary for `/etc/shadow` or PAM based authentication. But since our users are kept in LDAP, we should run this process as a less privileged user. On my RedHat-like system, this user can be the "nobody" user. I'm informed, however, that this will not work on Debian-based systems. In this case, and even on RedHat, you should create a dovecot-auth user and group.

```
# groupadd -r dovecot-auth
# useradd -m -r -d /usr/libexec/dovecot dovecot-auth
```

Note the use of `/usr/libexec/dovecot` as a home directory. This is where I've installed the Dovecot binaries. You can use whatever you want.

Configuring Dovecot

Dovecot uses the `dovecot.conf` file for most of its configuration settings. Using the above `configure` command the `dovecot.conf` file will be found in the `/etc` directory (in your case it might be `/usr/local/etc` or wherever you set `sysconfdir` to point to). LDAP is configured elsewhere and discussed in the next section. In general, if you leave a Dovecot setting commented out it defaults to something reasonable. Below, I will show only those settings that are meaningful in the context of this HOWTO.

```
protocols = imap imaps
```

Enable only IMAP and IMAP over SSL. Do not enable POP or secure POP. Though you can if you want to.

```
imap_listen = 127.0.0.1
```

Non-secure IMAP will only accept connections from local processes. This will be needed for SquirrelMail.

```
imaps_listen = *
```

Secure IMAP will accept connections from anywhere.

```
ssl_disable = no
```

It's not enough to simply set `imaps` in the protocols setting, you have to explicitly enable SSL.

```
ssl_cert_file = /usr/share/ssl/hosting.example/ExampleCert.pem
ssl_key_file = /usr/share/ssl/hosting.example/ExamplePrivateKey.pem
```

The absolute path to the certificate and private key created earlier. You do not need to specify the CA cert.

```
disable_plaintext_auth = no
```

Setting this to true would keep people from connecting unless they came in over SSL. However, that would keep SquirrelMail from working, so this has to be set to no. It's okay though, as the `imaps_listen` directive above keeps non-encrypted IMAP ports from being open to the outside world..

```
login_user = dovecot
```

The user that the login process runs as. The dovecot user should have been created for you during `make install` or during the package installation. Should not be root.

```
first_valid_uid = 101
last_valid_uid = 101
```

When we get around to configuring Dovecot for LDAP we will set up a single virtual user, `vmail`, just as we did for Postfix. Since `vmail` will be our only user, we can set the first and last valid user IDs to `vmail`'s uid; 101 in this exampl, almost certainly different on your system.

```
first_valid_gid = 101
last_valid_gid = 101
```

Same as above, but for groups.

```
valid_chroot_dirs = /home/vmail/domains
```

This is a list of directories where chrooting can take place. In our case, we need only one. It should be set to the root directory of our user's mailboxes, i.e. `/home/vmail/domains`.

Note: Immediately below this is a setting called `mail_chroot`. Do not set this! This value is implied by the fact that we are using an absolute path in the `default_mail_env` setting.

```
default_mail_env = maildir:/home/vmail/domains/%d/%n
```

The all important setting! Okay, if I got this right, Dovecot has this notion of a "mail environment." It consists of a mailbox format (mbox or maildir), a colon, the relative (?) or absolute path to the user's mailbox, and a few other things that are inadequately explained. It is possible to store the mail environment in LDAP, but since this is not a standard LDAP attribute, nor part of the Jamm schema, we will forego this. When the mail environment can't be retrieved from LDAP, Dovecot uses the `default_mail_env` instead. (If both of these are unavailable, I think Dovecot makes a best guess.)

The value of this setting is constructed at runtime from the text given here and some simple substitution (explained in the conf file comments). In my case it is set to use the maildir mailbox format. It also specifies that mailboxes can be found in `/home/vmail/domains/[the domain name of the user logging in]/[the user name of the user logging in]`. Expanded, this might be, `/home/vmail/domains/whitehouse.gov/george`. Note, I did not use "%u" (you) for user name, I used %n (en). This is because "%u" will expand to "user@domain.extension," and we just want the first part.

```
auth = default
```

Set up our first (and only) authentication process.

```
auth_mechanisms = plain
```

The user will send authentication information as clear text. The session, of course, is SSL encrypted.

```
auth_userdb = ldap /etc/dovecot-ldap.conf
```

Where the user database is. In our case, this is LDAP. The LDAP settings are found in the file /etc/dovecot-ldap.conf (created in the next step).

```
auth_passdb = ldap /etc/dovecot-ldap.conf
```

Where the password database is. Same as above.

```
auth_executable = /usr/libexec/dovecot/dovecot-auth
```

This is Dovecot's authentication executable. I didn't have to uncomment it as it's in the default place, but you may have to if you installed in `/usr/local`, for instance.

```
auth_user = dovecot-auth
```

The user to run the above authentication executable as. This is the user we created earlier.

That's it. There's a number of other Dovecot settings you might want to use, e.g., `auth_verbose`, `maildir_copy_with_hardlinks`, and so on. The conf file explains each of these well enough for you to decide.

Configuring Dovecot for LDAP usage

Dovecot keeps its LDAP settings in a separate file. This file is referenced by the `auth_userdb` and `auth_passdb` settings in `dovecot.conf`. It's name defaults to `dovecot-ldap.conf` and it should be in the `/etc` directory. You do not have to create this from scratch, a sample file can be found in the Dovecot docs (`/usr/share/doc/dovecot-0.99.10.9/dovecot-ldap.conf` on my system). Copy this file to `/etc` and edit it as follows.

```
hosts = localhost
```

The server name/IP address where LDAP is running.

```
dn = cn=dovecot,dc=myhosting,dc=example
```

The DN of the user that Dovecot will bind to LDAP as.

```
dnpass = secret
```

The Dovecot user's password. You do remember it, don't you?

```
ldap_version = 3
```

What version of LDAP to use.

```
base = o=hosting,dc=myhosting, dc=example
```

The LDAP base under which our users can be found.

```
deref = never
```

I have no idea. If you think you care, maybe this will help: http://www.holbaeksem.dk/help/readme.nsf/0/ffc017ce09e9fd2585256cc600651017?OpenDocument

```
scope = subtree
```

How far under the base should a search look. Subtree is all the way down.

```
user_attrs = mail,homeDirectory,,,,
```

Pay attention to this one. The `user_attrs` setting lists the names of the LDAP attributes for those parts of a user's entry that Dovecot cares about. They are, in order:

1. The virtual user's user name (user@domain).
2. The user's Home directory.
3. The user's mail environment. See the `default_mail_env` setting above.
4. The local user's user name.
5. The local user's user ID.
6. The local user's group ID.

Now, I may not have gotten this perfectly correct, but of these we're only interested in the first one. In the Jamm schema the virtual user's user name is stored in the `mail` attribute.

I have also set the attribute for the user's home directory (`homeDirectory` in the Jamm schema). This is not strictly necessary, and can be safely left out. However, Dovecot claims to have some additional logging that's dependent on this setting (among other things). This is also where core files will be dumped if Dovecot crashes. I was never able to get this logging to work, however, even after following the FAQ

on this subject.

As was discussed earlier regarding the `default_mail_env` setting, it is possible to put the user's mail environment (e.g., `maildir:/home/username/Maildir`) in LDAP, but since the standard LDAP schemas and the Jamm schema have no such attribute, we leave that blank.

As for the remaining three attributes, none of our users are local, therefore we don't need to set these. When Dovecot needs a uid and gid it will get them from the `user_global_uid` and `user_global_gid` settings below. It won't need a system user name, as, apparently, that's only needed for accessing `/etc/groups`.

```
user_filter = (&(objectClass=JammMailAccount)(mail=%u)(accountActive=TRUE)(delete=FALSE))
```

The LDAP filter Dovecot will use when looking up users. Should be familiar by now.

```
pass_attrs = mail,userPassword
```

The LDAP attributes that contain the user's virtual user name and password.

```
pass_filter = (&(objectClass=JammMailAccount)(mail=%u)(accountActive=TRUE)(delete=FALSE))
```

The LDAP filter Dovecot will use when looking up a user's password.

```
default_pass_scheme = CRYPT
```

The format that passwords are stored in LDAP. We use CRYPT here to match our setting in slapd.conf. Other values are PLAIN, PLAIN-MD5, and DIGEST-MD5.

```
user_global_uid = 101
user_global_gid = 101
```

Though we set the first and last valid uid and gid in `dovecot.conf`, we never did set the uid of the `vmail` user -- we do that here. This is the uid and gid Dovecot will use in lieu of the empty LDAP settings above when reading and writing from the user's mailbox and chrooting too, I guess.

And that's it for Dovecot.

Jamm

What follows are the [slightly edited] instructions from the original Jamm HOWTO.

Installing and Configuring Jamm

Installing and configuring a web servlet container like Tomcat or Resin is outside the scope of this document. (On my hosted system Tomcat was already installed in /usr/local/tomcat). However, once you have a working servlet container, installing and configuring Jamm is a snap. Change into the webapps deployment directory, make a new directory called jamm, cd into that directory, drop the jamm.war file (that we downloaded way back at the beginning of this HOWTO) into the jamm directory, and unjar the war file. Then cd to the WEB-INF directory. Copy jamm.properties.dist to jamm.properties, and edit jamm.properties as apppropriate.

```
# cd /usr/local/tomcat/webapps
# mkdir jamm
# cd jamm
# cp [wherever]/jamm-0.9.6.war .
# jar -xvf jamm-0.9.1.war
# cd WEB-INF
# cp jamm.properties.dist jamm.properties
```

Now you need to edit jamm.properties. To continue to follow our examples for dc=myhosting,dc=example, we've edited the following lines in jamm.properties.

```
jamm.ldap.search_base = o=hosting,dc=myhosting,dc=example
jamm.ldap.root_dn = cn=Manager,dc=myhosting,dc=example
```

None of the values in jamm.properties should have quotes around them. This will cause problems at run time as Jamm is not expecting them. This has bitten people in the past when they copied their rootdn from slapd.conf.

Administration

To access Jamm, startup your servlet container (on my system this is Tomcat; **service tomcat start**) if it's not already started. From a browser goto: http://servername.tld:8080/jamm.

To login as the site administrator, the username is "root" (as specified in the jamm.properties file). The password is whatever password you gave to the LDAP superuser or root user way back when we were configuring LDAP.

Jamm allows for three levels of access: the site admin, the domain admin, and the user. The site admin controls the entire site and has access to every option all the time, very much like root on a unix system. The domain admin can add, remove, and modify accounts and aliases for his domain as well as assign other people to be a domain admin. The user can only effect his settings.

Site Admin

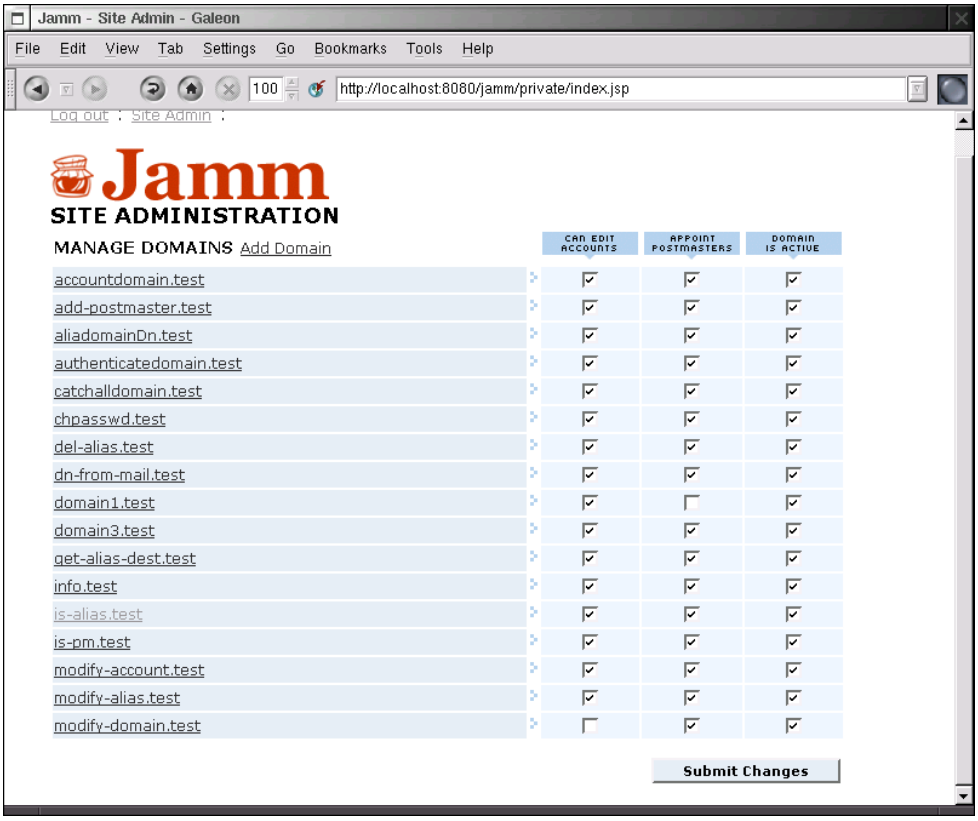

Figure 3. Site Admin Screen

When a site admin logs in, they are presented with a list of domains. They can click on the domain to drill down to that domain admin page or manipulate the capabilities of the domain admin.

Can Edit Accounts controls the ability for a domain admin to add and remove virtual accounts. When this is switched off, the domain admin can still modify the attributes of existing accounts such as the password.

Appoint Postmasters controls the ability for a domain admin to grant the powers of domain admin to other accounts in the domain. With this turned off, only the site admin can give users domain admin access.

Domain Is Active turns on or off the "active" flag on the domain in ldap. If your mail server or imap server are configured to pay attention to this flag, one can turn on or off domains temporarily without removing them from ldap.

Domain Admin

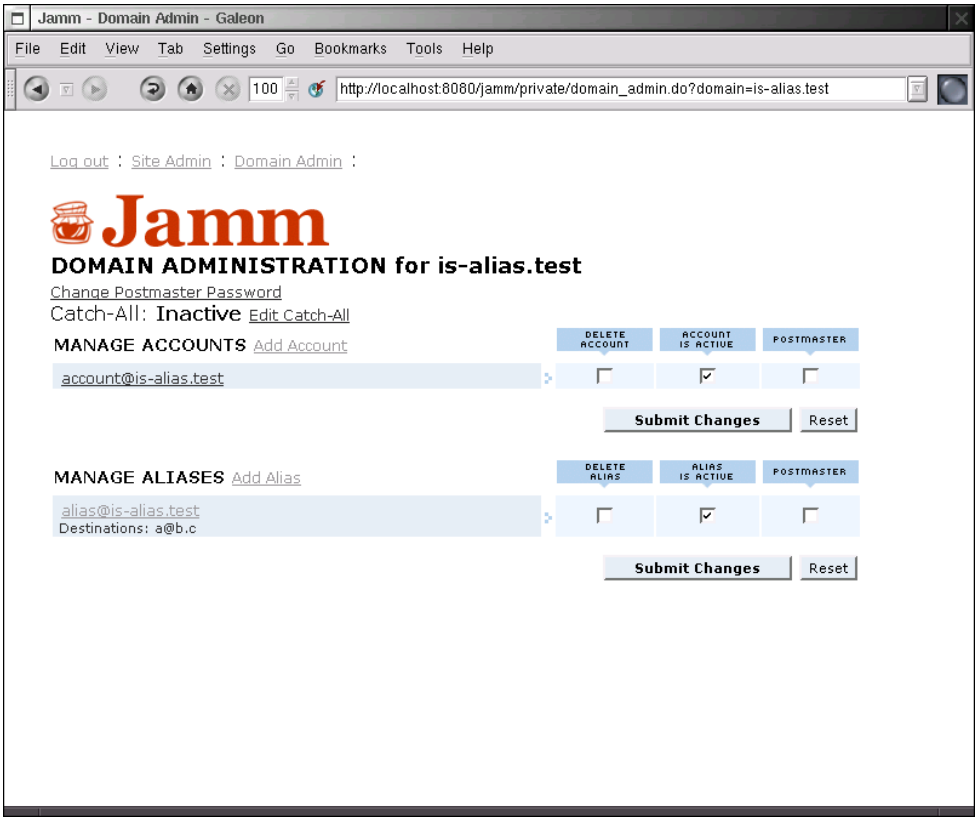

Figure 4. Domain Admin Screen

When a domain admin logs in, they are presented with a list of accounts and aliases for their domain. They can click on a user to drill down to that user admin page, add or delete accounts or aliases, appoint other admins/postmasters, and activate and deactivate accounts. Some of this options may not be present depending on how the site admin has configured the domain's capabilities.

Delete Account does pretty much what it says it will.

Account Is Active activatees or deactivates an account without deleting it. Much like *Domain Is Active*, your mail server and imap servers must be configured to pay attention to this flag inside ldap.

Postmaster gives or removes the ability for that user to act as a domain admin.

User Admin

Figure 5. User Admin Screen

When a user logs in, they are presented with a user screen appropriate to whether they have an account or an alias. Currently, all that a user with an account can do is change their password. An alias user is a bit more intereting, they can edit their destination(s).

To add destinations to an alias, the user only needs to add them in the text area in either a comma seperated list or one per line. To delete destinations, just check the box next to the destination to be deleted.

Account Creation Notes

When you create an account or an alias inside the LDAP database it will instantly become active as far as the mail system is concerned. For virtual accounts, it should be noted that the Unix directory in ~vmail is not created at this time. However, we can work around this because Postfix's virtual delivery agent will create the necessary directories the first time it has to deliver mail. Due to this fact, we recommend sending a welcome e-mail as soon as you create the account. Important! I did not find this to be true! Postfix did not create any directories for me. Therefore, for me anyway, account creation is a two step process; create the appropriate directorty tree (/home/vmail/domains/somedomain/someuser) and then create that domain and/or user in LDAP via Jamm.

Account Deletion Notes

When you delete an account or an alias in the LDAP database, it will instantly become inactive. For virtual accounts, it should be noted that the Unix file system isn't cleaned up, i.e. the data remains on disk until a sysadmin can remove it. This will allow you to keep the data from dead accounts around for a grace period in case the account was deleted in error. However, if another account is created with the same name with the same mail path, the data will be available to the new user. This could be considered a privacy violation for the previous user.

SquirrelMail

Installing and Configuring SquirrelMail

SquirrelMail is simplicity itself to install. You have to make sure your system matches the prerequisites stated here: http://www.squirrelmail.org/wiki/SquirrelMailRequirements – essentially Apache and PHP. After that you should install the binary package (or grab the tarball, if necessary – http://www.squirrelmail.org/download.php)

```
# apt-get install squirrelmail
```

The installation process will put the appropriate Apache 2.0 configuration file in Apache's `conf.d` directory. The config file simply adds a "webmail" alias that points at the Squirrelmail index page. It put Squirrelmail itself in `/usr/share/squirrelmail`. Now you just need to make a couple of quick changes to the Squirrelmail configuration. You can do this via a Perl script located at `/usr/share/squirrelmail/config/conf.pl` or by directly editing the config file `/usr/share/squirrelmail/config/config.php` (which is aliased to `/etc/squirrelmail/config.php`). I chose the latter. The edits are few and are summarized here:

```
$use_authenticated_smtp = true;
$imap_server_type       = 'courier';
$optional_delimiter     = '.';
$default_folder_prefix  = '';
```

Yes, I know it says "courier" for IMAP server type, but Squirrelmail doesn't have a quirks mode for Dovecot, and the Courier settings work. There are any number of other changes you may want to make, but they're all optional.

Enabling Apache 2.0 SSL

Important: If you are supporting (name-based) virtual hosts, then read this: http://httpd.apache.org/docs-2.0/ssl/ssl_faq.html#vhosts2.

It's likely that your Apache install can already speak SSL. If you're happy with this, then great, skip to the next section. However, the default SSL configuration won't be using those shiny new certs we made earlier. To enable SSL to begin with or to modify which certs it uses goto `/etc/httpd/conf.d` (on my system, yours may be different) and edit the file `ssl.conf`.

First, make sure that SSL is enabled by searching for the string "Listen" (with a capital "L"). It should be uncommented and set to an IP address of all zeros or the IP address of the server and followed by a port number (443), like this:

```
Listen 0.0.0.0:443
```

It's likely that the rest of the global settings are acceptable, so skip down to the VirtualHost settings and search for the string `SSLCertificateFile` and change the file path to `/usr/share/ssl/hosting.example/ExampleCert.pem` (again, hosting.example/ExampleCert.pem is a place holder - substitute in the actual name you gave the cert). A few lines below this is the private key information. So change `SSLCertificateKeyFile` to point to `/usr/share/ssl/hosting.example/ExamplePrivateKey.pem`. That's all the changes you *need* to make, but you may want to fiddle with some other settings. Save the file and restart Apache: `service httpd restart`.

Enabling SquirrelMail SSL

Now that Apache can handle SSL, you only need to make one small change for SquirrelMail. In the same directory, `/etc/httpd/conf.d`, you'll find the SquirrelMail configuration file `squirrelmail.conf`. It's just one alias command. Add the following command to it:

```
<Location /webmail>
  SSLRequireSSL
</Location>
```

You can now access Squirrelmail via the URL: https://myhosting.example/webmail.

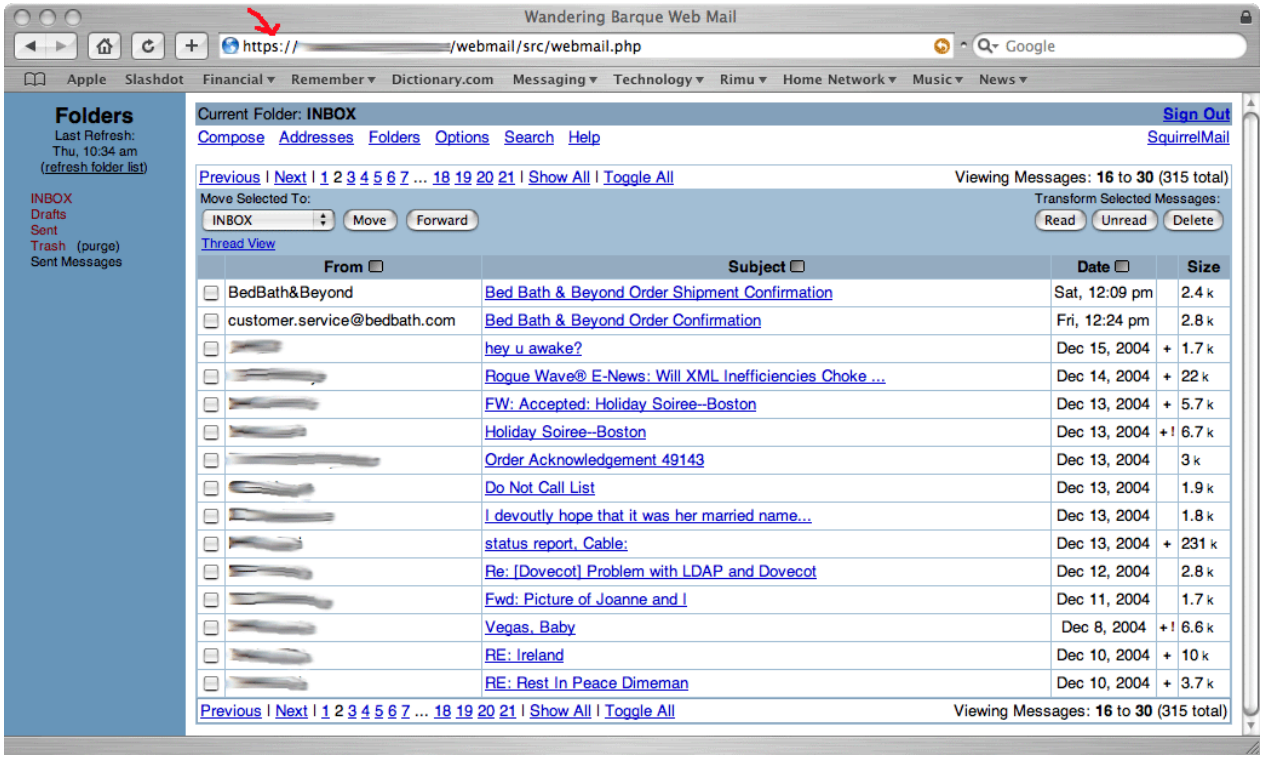

Figure 6. My Squirrelmail interface after fiddling with fonts and themes.

Allow Users to Change Their LDAP Password from SquirrelMail (OPTIONAL)

The functionality presented here is entirely optional as it reproduces some of the functionality of JAMM. However, now that SquirrelMail is up and running, it makes sense to me to have the user stay in that interface as much as possible. Over time I will install and create more plugins to SquirrelMail so that the everyday user can perform all personal administration from their.

To allow a user to change their LDAP managed password, first download the "change_ldappass" SquirrelMail plugin: http://squirrelmail.org/plugin_view.php?id=26. Change_ldappass is dependent on the Squirrelmail "compatibility" plugin, so download that too: http://www.squirrelmail.org/plugin_view.php?id=152.

Installing it is a breeze. First, untar the compatibility plugin into the Squirrelmail plugins directory. Then untar the change_ldappass package into the SquirrelMail plugins directory, cd into the resulting `change_ldappass` directory, copy the `config.php.sample` file to `config.php` and edit it.

The changes that have to be made to the config file are minimal and limited to the very top of the file. Simply change the `$ldap_user_field` to `mail`, the LDAP attribute where our usernames (e.g. joe@example.com) are stored; and change the `$ldap_base_dn` to your version of `o=hosting,dc=myhosting,dc=example`. Do *not* change the `$ldap_password_field` from `userpassword` to `userPassword` (note the capital "P") as I did. It will work with the default, but not elsewise. The top few lines of the `config.php` file should look like this:

```
<?php

        $ldap_server = "localhost";
        $ldap_password_field = "userpassword";
        $ldap_user_field = "mail";

        //put the ldap base dn of your server here
        $ldap_base_dn = "o=hosting,dc=myhosting,dc=example";
```

There's no need to restart Apache. Users can now access the change password screen from the "options" page of SquirrelMail.

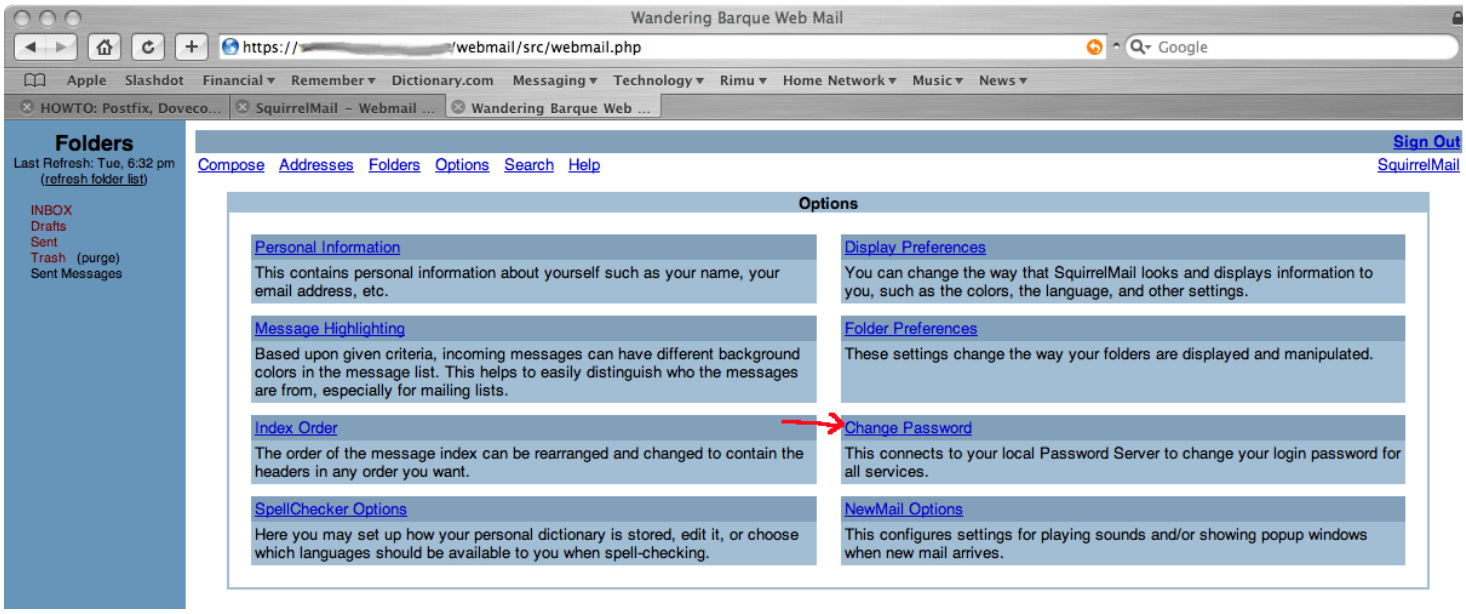

Figure 7. The new "Change Password" option in Squirrelmail's option page.

Post Install Configuration

Now that all the software has been installed and configured, there are a few other things you probably want to do.

Make LDAP inaccesible to the Internet

Currently the OpenLDAP process will answer requests from anywhere. What you probably want to do is limit connectivity only to the processes running locally (on this server). This is done with the -h flag to slapd. I modified the init script on my system, /etc/init.d/ldap, to accept normal and SSL connections only from processes on the same host. Here's the relevant part of the init script, in the start function.

```
prog=`basename ${slapd}`
echo -n $"Starting $prog: "
if grep -q ^TLS /etc/openldap/slapd.conf ; then
    daemon ${slapd} -u ldap -h '"ldap://127.0.0.1 ldaps://127.0.0.1"' $OPTIONS $SLAPD_OPTIONS
    RETVAL=$?
else
    daemon ${slapd} -u ldap -h '"ldap://127.0.0.1"' $OPTIONS $SLAPD_OPTIONS
    RETVAL=$?
fi
```

Start up on reboot

You probably want to set up your system to start up all processes at reboot. This is especially true if your server is hosted, and you may not even be aware that your server has been restarted. If your machine is local to you, you can probably use the appropriate GUI application. However, if your machine is remote (and RedHat based), you can use the chkconfig utility to do this. Or, if necessary, do it manually. I set all processes to start at runlevels 3 and 5. You may want to use 2, 3, 4, and 5. You can get details on how chkconfig elsewhere, but here's how I did it:

```
# chkconfig --level 35 ldap on
# chkconfig --level 35 saslauthd on
# chkconfig --level 35 postfix on
# chkconfig --level 35 dovecot on
# chkconfig --level 35 tomcat on
```

Make sure you're not a relay

If you followed the instructions above, and given the default configuration of Postfix, you should not be acting as an open (spam) relay. But, better safe than sorry. Go here and test your system: http://www.abuse.net/relay.html

Adjust Postmaster Account

When you create a virtual domain with Jamm it creates the postmaster@domain.name and abuse@domain.name accounts automatically. Both of these accounts are set as aliases to the "postmaster" user who is presumed to be be a local user. This is fine, but on my system I don't want *any* local users. Besides, if I did, I'd have to add another authentication mechanism to Dovecot, and I don't want to do that either. In any case, if you would like to change the abuse and postmaster aliases to point at a virtual user, you can do so as follows:

Create an LDIF file that looks something like this:

```
dn: cn=postmaster,jvd=domain.name,o=hosting,dc=myhosting,dc=example
changetype: modify
replace: maildrop
maildrop: user@domain.name

dn: mail=abuse@domain.name,jvd=domain.name,o=hosting,dc=myhosting,dc=example
```

```
changetype: modify
replace: maildrop
maildrop: user@domain.name
```

Where domain.name is the virtual domain you've created and maildrop is the virtual user in that domain who is to receive mail for postmaster and abuse.

You can use this file to update the LDAP directory like this:

```
# ldapmodify -x -D "cn=Manager,dc=myhosting,dc=example" -W -f ldif_file_name
```

Email Client Settings

I won't detail how to set up mail.app, Thunderbird, Outlook, etc., but here are the client settings that should be generally applicable.

Assumptions: Your mail server is accessible via a public DNS MX record at example.net. You want to host a virtual domain of anydomain.org and its MX record points to the same host, Julie is a user at anydomain.org.

User Name: julie@anydomain.org
Password: [Julie's password]

Incoming Mail Account Type: IMAP
Incoming Mail Server: example.net
Incoming Mail Server uses SSL: Yes, on the default port of 993
Incoming Mail Server Authentication: Password

Outgoing Mail Server: example.net
Outgoing Mail Server uses SSL: Yes
Outgoing Mail Server Authentication: Password

Also, remember that you'll want your users to import the signing (CA) cert into their client or OS as applicable.

Credits

In the overly long time it took to make all this work I regret to say that I failed to bookmark every web page, HOWTO, and newsgroup posting from which I gleaned some information. Here, though, are the most prominent.

- Jamm Virtual Mailserver Howto
- Dovecot Documentation
- Dovecot Wiki
- Dovecot Wiki page on OpenLDAP
- Postfix Documentation
- Basic Postfix Configuration Docs
- Quick Start to Authenticate with SASL and PAM
- Someone named Jeremy's notes on Postfix, Dovecot, and SquirrelMail
- Someone else's Postfix, Dovecot, OpenLDAP notes (with spam and virus filtering)
- OpenLDAP Quick-Start Guide
- LDAP for Rocket Scientists
- L'Xtreme: Dovecot, Postfix, PAM
- Postfix SMTP AUTH (and TLS) HOWTO
- Eclectica: Creating and Using SSL Certificates