Chapter 4. Beyond Default

Table of Contents

We will be examining three environments in this document. First the home user. Then the corporate environment, and finally we'll dig into some ISP type configurations.

4.1 The Home User

The home user environment could be very simple or very complex depending on the number of users on their system, whether they have a networked environment or a stand alone environment, and their type of connection and service provider. The variables make it difficult to cover all the possibilities but most of what we need to handle can be covered if we deal with some essential issues surrounding the stand-alone dial-up users requirements.

4.1.1 The Stand Alone Config

If your machine does not have a network card and is not connected to a Local Area Network (LAN) and you use your MTA to send mail to the net over a PPP connection, a cable modem, or a Digital Subscriber Line (DSL) connection, this section is for you. We will not be dealing with the connectivity issues as that is beyond the scope of this document. We will work through the configuration you need to be able to send mail with sendmail while off-line and online, how to kick the queue once you've connected, and other similar issues.

Here is a sample configuration which should work well for a stand alone dial-up user:

```
divert(-1)
include('/usr/lib/sendmail-cf/m4/cf.m4')
define('confDEF_USER_ID',''8:12'')
define('SMART_HOST','mail.yourisp.net')dnl
OSTYPE('linux')dnl
undefine('UUCP_RELAY')dnl
undefine('BITNET_RELAY') dnl
FEATURE('redirect')dnl
FEATURE('always_add_domain')dnl
FEATURE('use_cw_file')dnl
FEATURE('local_procmail')dnl
FEATURE('nouucp')dnl
FEATURE('access_db')dnl
MAILER(procmail)dnl
MAILER(smtp)dnl
MASQUERADE_AS(yourdomain.net)dnl
FEATURE('allmasquerade')dnl
FEATURE('masquerade_envelope')dnl
```

It is important to note here that the only lines which are actually required here are:

```
OSTYPE()
MAILER()
```

Without these two m4 will fail when run.

We'll get into the details of this in a moment but first we need to digress and discuss masquerading and relaying and why we need them.

4.1.2 Masquerading, relaying and a word about UCE

The *masquerade* lines shown above are critical. *YOU MUST HAVE THESE LINES IN ALMOST ALL CASES* for the stand alone environment using a dial-up connection. Why is this so important? Let's compare an email message to a letter sent through the postal service. When you address the envelope you provide several pieces of information so the postal service can do their job correctly. You would write the person's name and address that you want to send it to and you would also write your name and mailing address so that if the letter can't be delivered, it can be safely returned to you. After you post the letter what does the postal service do? They pick it up and attempt to deliver it based on the information you provided.

Now, if you had a criminal mind and wanted to mail something malicious, you wouldn't put your correct return address on the envelope would you? Probably not. If you put a false return address and the message was ultimately undeliverable the postal service wouldn't be able to return it to you. In this example, everything is very anonymous and you can send out garbage without being held accountable. Just a few short months ago, Internet based electronic mail worked the same way. You could send out garbage with a false return address and it wouldn't be returned to you if it proved undeliverable. Granted, this is not a very nice thing to do but it happens all the time.

In order to eliminate this abuse the people who develop code for Mail Transfer Agents (MTAs -or- the post office in our scenario) started to tighten things up a little bit. These much improved MTAs now check that the sender's envelope address is valid before they will relay, or deliver, a message. What this means is that the post office transmitting the message must exist... it has nothing to do with the from address. This is a good thing but it has caused problems for many people, particularly those who use an MTA rather than client software to send their mail... translation... the dial-up user with a Linux box is likely to get a bunch of bounced mail unless he or she configures their MTA to properly masquerade the envelope and relay through their provider's MTA.

It's not enough to set the From: and Reply-To: addresses in a piece of email client software because, much like a regular letter, all this does is tell the person who receives the message who it's from (maybe). The newer versions of sendmail (particularly since the release of 8.9.1) and other modern MTAs are more concerned with which mail server or MTA it was sent from than they are with which user sent the message so they read the message envelope. They're not looking for the person that sent it by checking the sender's From: or Reply-To: email address, they're checking to see if the post office, or MTA, the message came from is a valid host and that the domain is also valid. This is equivalent to checking the zip code of origin on a letter. Once this information is excerpted from the message it's verified via reverse DNS to see if the sending post office and the

"From: domain exists (in other words, would the sending MTA would receive a bounce if the intended recipient didn't exist) and if that post office (the sending MTA) or the domain does not exist, then the receiving MTA will reject the message. We fix this by configuring our MTA to masquerade the envelope and then we relay our messages through our ISP's mail host which gives our mail the final stamp of legitimacy it needs.

So when using sendmail the primary keys to getting this right are using the correct values for MASQUERADE_AS, define('SMART_HOST', xxx.xxx.xxx), and then we have to make sure we add: FEATURE(allmasquerade)dnl and FEATURE(masquerade_envelope)dnl to our configuration.

In today's Internet environment, no one's MTA should be sending email "direct" from a dial-up connection unless they have a static IP and an MX record which is verifiable via DNS. A fine example of a configuration in which this would be acceptable is a user with a registered domain whose dial-up access is provided by an ISP which gives them a static IP address and handles their primary DNS and secondary MX. In this way any inbound or bounced mail would be stored by the ISP and then forwarded to the primary MX (the users dial-up host) when it was next connected. This is called store and forward but is really a delayed relay. It is critical in today's net environment that relaying be controlled.

What is relaying? Let's look at this from the point of view of a ficticious mailhost for a moment. From the standpoint of this fictional mailhost it thinks of itself as a server and any connection inbound comes from a client irrespective of the actual source. Even though the inbound connection may be from an MTA or a user's machine sending mail using a Mail User Agent (MUAs include Netscape Mail, pine, postilion, etc.) the sending host has still become a client machine the moment it connected to our fictional mail server. So this would be what is called a client-server environment. In other words the client (an MTA or a user's machine) has connected to a service being run on another machine (the server) and the inverse applies when our fictional host sends mail outbound. So, if we use this analogy then what is it that the client is doing? It is sending a message to, or through, the server. If the message is for local delivery then this is the destination host and this mail should always be accepted so long as the sending host and domain are authentic. Any other mail is defined as a relay and the receiving MTA has to be told, in its configuration that it will accept relays from the sending host and or domain or to the intended destination, or all of these. In this scenario an open relay would be a host which does not ever verify the authenticity of its client connections and just stupidly relays everything regardless of origin or destination. This would be an extremely bad thing. Unfortunately the problem is very real and it exists all over the world. Why is it bad?

Well, using our fictional mailhost again, imagine that this mailhost is owned by a small Web hosting company which is trying to build its business. They are paying several thousand dollars a month for the services of a sys-admin, a dedicated connection to the Internet, and they have also invested in the hardware which constitutes our fictional mailhost. This host is handling email accounts for several thousand people so this company has a substantial risk involved if this hardware, their dedicated link, or their sys-admin should fail at their assigned tasks. The company's domain name is company.com and they've been in business for just about a year when suddenly someone in Rhode Island decides to do a spam run using bob@company.com as a Reply-To: address. Now bob is a real user and is actually the CEO of company.com. The spammer has purchased a list of email addresses with about a million addresses on it and has also spent a couple of hundred dollars on a piece of dedicated spamware, which runs on Windows 98. He sets up a throw away dial-up account and proceeds to send a million emails out using bob@company.com as his from address. His total investment is about $500.00 and he's using the family net machine which he bought at a discount house for little Johnny to do his homework. In order to send his

spam, he relays it off of an SMTP host belonging to the telephone company in China which is an open relay. He breaks this down so that each run of 1000 addressees is listed in the blind CC: of his message and no one can see any of the other recipients. He has to do a thousand runs to make this all work but he doesn't care how long it takes because the company which has hired him to do this is paying him a tenth of a cent for each message sent and all he has to do is log his outgoing traffic. He spends about 8 hours and makes a thousand dollars for his trouble. What the heck, right? The computer does all the work. The next morning, company.com is in a real mess. Over 250,000 messages from the spam run have bounced and company.com's mail server is on its knees trying to handle the incoming traffic. Sadly, company.com is a young outfit just starting to get on their feet after their initial cash investment. Their mail server won't respond because it's buried from the load of this spam. Their bandwidth is also getting pounded because of all the inbound complaints and bob@company.com can't get his personal email because there are 250,000 bounces plus about 10,000 complaints inbound to his mailbox. By noon of that day about half of their hard won clients have called to cancel their service and by nightfall bob@company.com is trying to figure out how he's going to make payroll and overhead without enough paying clients. Later that same night, our intrepid spammer repeats his criminal act using the same From: address and by noon of the following day company.com is essentially out of business.

This is a really awful example of what an open relay can do to small companies trying to run a business on the net, but it has happened. The spammer stole resources which did not belong to him or her and destroyed the livelihoods of several people in the process. If we toned down the example and spread it around a little bit this same spammer might send mail with 100 different *From:* addresses in a night. That mail could end up scattered across several hundred thousand mail hosts around the world and though he was paid for his work he actually made that money by stealing the storage space and bandwidth from all of the receiving hosts, companies, and recipients. Now if the Chinese telecom's open-relay had been a properly managed machine that mail would have been rejected at the SMTP port and would have never touched company.com's mail host. If the sending ISP had properly managed their service the bounces generated would have returned to the envelope sender which means that the ISP being used to send this mail would have seen it and killed the idiot spammer's account immediately. Many, many providers today will not allow their users to send anything outside their network on port 25 (the SMTP port) unless they relay through their (the provider's) MTA. Now this may be an extreme solution but it works. Without going to this extreme the simplest and best answer is for everyone who runs an MTA on the net to properly control their relays. If you were bob@company.com and you knew what had happened what would you think?

In spite of the commonality of the gruesome example given here, some people still have difficulty understanding the reasoning for the need to relay through a fully qualified MX host. Those that complain about this have never had the dubious pleasure of handling the thousands of complaints and bounces generated by those who refuse to comply with this simple guidance and insist on sending mail from a domain which does not exist (or in some cases even a domain which *does* exist) and or running an open relay.

An example of a configuration which would still be problematic even for a real domain is a dial-up user who uses a dynamic DNS service (such as dhis.org, dyndns.org, ns1.net, wiw.org/dyn/, dynip.com) to handle their domain, but their IP address is truly dynamic and is issued from a block of IPs which are well known as a dynamic range. If they send mail direct rather than relaying through their ISP and any of the destination domains are subscribers to the DUL (see http://www.mail-abuse.org for information about the DUL) then their mail is likely to be bounced in spite of the fact that their domain is verifiable via DNS. If they would masquerade the envelope properly and use their ISP's MX host as a relay then it would not matter what type of IP address they were operating from. The mail would go through anyway as the relay decision would be made

by their service provider's MTA.

4.1.3 So how do we configure this so it will work?

Now that we've been through some of the concepts and thinking we should be a bit better prepared to deal with this configuration. It often helps if the basic reasoning behind concepts is made clear and that's what we tried to do for you in our discussion on masquerading, relaying and UCE. In other words, now that we understand the *why* it should be easier to execute the task at hand. That said, let's lay out a scenario for our needs. Our user account at our ISP is '*bob*' and so is our user account on our local machine. Our ISP is isp.net so our email account in this scenario is '*bob@isp.net*'. Our ISP's SMTP host is '*mail.isp.net*' and we want to be able to send and receive mail using sendmail on our host which we have named *ns1*. Now let's lay out a sample configuration for this scenario.

```
divert(-1)
include('/usr/lib/sendmail-cf/m4/cf.m4')
OSTYPE('linux')dnl
define('confDEF_USER_ID',''8:12'')
define('SMART_HOST','mail.isp.net')dnl
undefine('UUCP_RELAY')dnl
undefine('BITNET_RELAY') dnl
FEATURE(redirect)dnl
FEATURE(always_add_domain)dnl
FEATURE(use_cw_file)dnl
FEATURE(local_procmail)dnl
FEATURE(nouucp)dnl
FEATURE(allmasquerade)dnl
FEATURE(masquerade_envelope)dnl
FEATURE('access_db','hash -o /etc/mail/access')dnl
MAILER(procmail)dnl
MAILER(smtp)dnl
MASQUERADE_AS(isp.net)dnl
```

We create this file in */usr/lib/sendmail-cf/cf/* and we call it *nuconfig.mc*.

Now we need to run m4 against this file to create a sendmail.cf file for our running configuration and then run a simple test to make sure that it works. Here's how we do it:

```
cd /usr/lib/sendmail-cf/cf/
cp /etc/sendmail.cf /etc/sendmail.cf.bak
m4 nuconfig.mc > /etc/sendmail.cf
/etc/rc.d/init.d/sendmail restart
ifup ppp0
mail -s 'test message' bob@isp.net
This is a test message.
.
Cc: [Enter]

tail /var/log/maillog

Nov 23 00:39:26 ns1 sendmail[20245]: AAA20243: to=bob@isp.net, ctladdr=root
```

If your maillog shows something like the above then you have successfully transmitted your message. Now we have a pretty good idea what your next question is going to be..."How in the world do I get the mail now?" Hey... don't worry about it we're just about to answer that question.

4.1.4 Fetching the mail?

The next part of our discussion details a tool which is currently maintained by Eric S. Raymond. The name of the tool is fetchmail and it is an outstanding tool for people in your situation. You have a net connection and an MTA but you don't have a real domain and you need to get your email.

First let's make sure that fetchmail is installed:

```
[root@boss sbin]# rpm -q fetchmail
fetchmail-5.1.0-1
```

Ah... in our case it is... but if you get a null response don't panic. It comes on your distribution CD-ROM and you can easily install it if it's not there already.

So now that we've verified that it's installed what do we do next?

Well... it's just not a problem. Fetchmail, like most Linux tools, uses a simple, text based configuration file with simple syntax. The file goes in your home directory and it's called .fetchmailrc (note the dot, or period, in the filename). If we continue on with our scenario this is what the text of the file should look like:

poll mail.isp.net proto POP3 user bob is bob here password hidden

What this means is that we want to 'poll' the machine called 'mail.isp.net' with the POP3 protocol using the username bob with the password hidden. Now all this means, quite simply, is that we're going to connect to mail.isp.net using the POP3 protocol, logging in as bob, using the password hidden, and that any mail for bob which is there should be transferred to the mail spool which belongs to the user bob here on the local machine.

Alright now we're ready to run fetchmail. Here's what you do:

```
[bob@boss bob]$ fetchmail -v -a
File /home/bob/.fetchmailrc must have no more than -rwx--x--- (0710) permiss
```

So now we have an error message. But notice how decent an error message it is... it tells you exactly how to fix it. So let's do this:

```
chmod 0710 /home/bob/.fetchmailrc

[bob@boss bob]$ fetchmail -v -a
fetchmail: 5.1.0 querying mail.isp.net (protocol POP3) at Tue, 23 Nov 1999 0
```

```
fetchmail: POP3< +OK POP3 mail.isp.net v7.59 server ready
fetchmail: POP3> USER bob
fetchmail: POP3< +OK User name accepted, password please
fetchmail: POP3> PASS *
fetchmail: POP3< +OK Mailbox open, 4 messages
fetchmail: POP3> STAT
fetchmail: POP3< +OK 4 1517 4 messages for bob at mail.isp.net (1517 octets)
fetchmail: POP3> LIST
fetchmail: POP3< +OK Mailbox scan listing follows
fetchmail: POP3< 1 382
fetchmail: POP3< 2 379
fetchmail: POP3< 3 378
fetchmail: POP3< 4 378
fetchmail: POP3< .
fetchmail: POP3> RETR 1
fetchmail: POP3< +OK 382 octets reading message 1 of 4 (382 octets)
```

...and it continues until...

```
reading message 4 of 4 (378 octets)
fetchmail: SMTP> MAIL FROM:<bob@isp.net> SIZE=378
fetchmail: SMTP< 250 Ok
fetchmail: SMTP> RCPT TO:<bob@localhost>
fetchmail: SMTP< 250 Ok
fetchmail: SMTP> DATA
fetchmail: SMTP< 354 End data with <CR><LF>.<CR><LF> #*
fetchmail: SMTP>. (EOM)
fetchmail: SMTP< 250 Ok: queued as 782A3BD7B flushed
fetchmail: POP3> DELE 4
fetchmail: POP3< +OK Message deleted
fetchmail: POP3> QUIT
fetchmail: POP3< +OK Sayonara
fetchmail: SMTP> QUIT
fetchmail: SMTP< 221 Bye
fetchmail: normal termination, status 0
```

Notice the command string we used with fetchmail. We used a *-v* which means to use verbose mode for its output (so we can tell what it's doing) and we used *-a* which means to download all mail. But where did our mail go? Well... as we said in our .fetchmailrc we wanted the mail to go to the mail spool belonging to local user *bob* and that is exactly where it went. The order of events for this is quite simple. Fetchmail grabs the remote mail using POP3 (port 110) and then hands off the mail to sendmail (using port 25) which, after determining that the mail is for local delivery, passes off the mail to procmail (the default MDA on Red Hat Linux) which delivers the mail to the Inbox you specified with the "*here*" statement in your fetchmail command string. One potential problem with this is that sometimes the mail could be rejected as fetchmail tries to hand off to sendmail. If you see this happening add this line:

```
localhost   RELAY
```

to your */etc/mail/access* file. The reason this might prove necessary is that sometimes sendmail may even reject relay from the localhost if it's not explicitly authorized.

4.1.5 Reading the mail

The mail spool for user *bob* is located in */var/spool/mail/bob* and *bob* can read his mail in a myriad of ways but why don't we try one of the simplest, but most powerful ways to read it, the pine MUA.

The first time we run pine by typing *pine* at the command prompt we will see a message on the screen which says this at the top:

<<<This message will appear only once>>>

Simply press the Enter key or the letter "e" and this will go away and you will see a menu that looks like this:

```
?     HELP               -  Get help using Pine
C     COMPOSE MESSAGE    -  Compose and send a message
I     MESSAGE INDEX      -  View messages in current folder
L     FOLDER LIST        -  Select a folder to view
A     ADDRESS BOOK       -  Update address book
S     SETUP              -  Configure Pine Options
Q     QUIT               -  Leave the Pine program
```

Just press "L" to see your default folders and then press the "Enter" key to see your "Inbox" and there are your four messages. Note that your default Inbox in pine is set to the file */var/spool/mail/bob*.

So there you have it... your MTA is working properly for both outgoing and incoming mail.

4.1.6 A couple of tweaks to make it all nice:

There are really two things which a dial-up user may want to add to this configuration. One thing is to add a *sendmail -q* line to your connect script to force sendmail to dump its queue after the connection is made and you also might want to do the same for fetchmail. Another thing which you could do is run fetchmail in daemon mode or as a cronjob and time it to force a connection hourly. As an alternative you could automate the connection itself and include the sendmail and fetchmail commands in the connection script as mentioned above. Here's a sample of how to run fetchmail as a daemon and as a cronjob:

fetchmail -d 180 <---- this would run fetchmail as a daemon and it tells it to poll the mail server every 180 seconds.

*0,3,6,9,12,15,18,21,24,27,30,33,36,39,42,45,48,54,57 0-23 * * * fetchmail -f /home/bob/.fetchmailrc > /dev/null 2>&1* <---- this cronjob would run fetchmail every 3 minutes for all 24 hours of every day

4.1.7 The Networked Home User Environment

There aren't many changes required between the stand alone and the networked user. Some key elements would be the */etc/mail/access* file where you will have to explicitly allow your local

subnet access to your MTA as a relay. You will also need to ensure that you have the IMAP RPM installed. If we presume that you're using the 192.168.1.0 network for your local network and you have a machine called sally at 192.168.1.10 and ralph at 192.168.1.12, and bob at 192.168.1.14. Here's a brief HOWTO on both the access file and the IMAP RPM elements:

In */etc/mail/access* add:

```
192.168.1.10              RELAY
192.168.1.12              RELAY
192.168.1.14              RELAY
```

Then issue the following command:

makemap hash /etc/mail/access < /etc/mail/access

This builds the .db file from the access text file.

Next we need to check for the IMAP RPM. Here's what we do:

```
[root@ns /root]# rpm -q imap
imap-4.5-3
```

In the example shown above the IMAP RPM is already installed but if you get a null response don't panic. The RPM is on your distribution CD-ROM. Just install it and then check */etc/inetd.conf* and make sure that the ipop3d line is not commented out. Now your client machines on your local network will be able to check their mail with their MUAs using POP3. Don't be thrown by the name of the RPM. It installs both the IMAP and the POP3 daemons.

It's also possible in this environment to have other users on Linux boxes. If this is the case they may want their mail delivered directly to their local machine. A quick, but simple scenario for this is to use the mailertable. Here's how:

User bob is using Red Hat Linux 5.2 on 192.168.1.10. He runs his own MTA and would like mail delivered to him there. In the file /etc/mail/mailertable (just create it if it doesn't exist) add this entry and then run the following command:

```
bob@isp.net              SMTP:bob@[192.168.1.10]
makemap hash /etc/mail/mailertable < /etc/mail/mailertable
```

Then we need to add the following line to the file we created before called */usr/lib/sendmail-cf/cf/nuconfig.mc*:

```
FEATURE('mailertable','hash -o /etc/mail/mailertable')dnl
```

and then create the new sendmail.cf just like we did before.

This will forward all mail inbound for bob directly to bob's machine.

Bob's machine will need to be configured to relay mail through your gateway mail host (we called it *ns1* remember) for forwarding on to the net. Essentially he should use the same configuration for his machine that you've used for *ns1* with the exception that his machine would use *ns1* as its smarthost. Bob could also use a null-client configuration. See the document: */usr/doc/sendmail/README.cf* for more information on this.
