In this chapter:
- *What Is UW IMAP?*
- *UW's Strengths*
- *UW's Limitations*
- *UW IMAP Concepts*
- *Does UW IMAP Match Your Needs?*

10

Introduction to the UW IMAP Server

This chapter provides a high-level picture of the University of Washington IMAP server. The server is part of the IMAP4rev1/C-Client Development Environment written by Mark Crispin of the University of Washington. Crispin is also the author of the IMAP RFC itself, as well as several of the ancillary RFCs related to IMAP.

The primary strength of the UW server is its flexibility. While the Cyrus IMAP server has very specific requirements about the format of the mailstore, UW goes to great lengths to support numerous formats. If Cyrus with all its advanced administration and information-sharing features can be thought of as doing a lot with a little bit, UW is the converse, doing a little bit with a lot. UW lacks the application-layer quotas and access-control support of Cyrus, but its ability to handle many different mail formats makes it an attractive choice for sites that have a chaotic mail infrastructure. It's also the best server for sites that need to bring IMAP up in a hurry with a minimum of time spent on configuration and migration.

What Is UW IMAP?

The University of Washington IMAP server (UW IMAP) is an IMAP server that uses *inetd* or a similar Internet superdaemon to provide users IMAP access to a mailstore.

Usually when people refer to UW IMAP, they're referring specifically to the IMAP daemon component of the IMAP4rev1/C-Client Development Environment. The development environment bundle includes an IMAP test utility called *mtest* and an IMAP API library called C-Client. It also includes a couple of POP servers that offer proxy access to your IMAP server through POP, for an easier transition from legacy POP systems. The UW IMAP daemon itself is bundled with the popular PINE mail client and included with many versions of the Linux operating system.

Available in a separate package are the UW IMAP Utilities, a set of tools for managing an IMAP server. The UW IMAP utilities were developed by the University of Washington and based on the C-Client API. They're covered in Chapter 18, *IMAP Tools*.

The UW feature set and design make it well suited for an existing Unix system that wants to add IMAP. It can be used out of the box on any Unix shell user system, without modifications or special infrastructure.

It can also be used for a dedicated IMAP server; however, you may need to start thinking about modifying it if you plan on scaling it to very large user communities. How many UW IMAP users a particular system will support depends greatly on the hardware and the operating system. UW IMAP does not need much in the way of CPU resources, but it does require adequate per-process memory and disk bandwidth. You can have more UW IMAP users on a system than Unix shell users, but within reason: if a particular machine won't handle 5,000 Unix shell users well, don't expect it to handle 100,000 UW IMAP users well.

In general, scaling works better with a cluster of small systems than with a gigantic monolith. A fast CPU is much less important than lots of disk bandwidth. As simple a trick as putting sendmail's */var/spool/mqueue* directory on a different disk than IMAP mailboxes results in significant performance benefits.

The University of Washington serves its community of 80,000 users with a cluster of small, inexpensive IMAP servers, each of which is assigned a portion of the overall user space. The IMAP servers are in a special DNS domain that is tied to UW's account system. User *fred* may be moved to a different IMAP server, but *fred.deskmail.washington.edu* always points to his assigned IMAP server.

Most Unix variants, particularly the open source varieties, typically come with an unlabeled IMAP daemon (*imapd*). Chances are that the daemon is the UW IMAP server. You will probably find no obvious documentation or clues as to the daemon's origins, but if you'd like to identify it as the UW server, there are a couple of things to try. First, look at the server's capabilities:

```
% echo ". CAPABILITY" | /usr/local/sbin/imapd
* OK localhost IMAP4rev1 v12.250 server ready
* CAPABILITY IMAP4 IMAP4REV1 NAMESPACE IDLE SCAN SORT MAILBOX-REFERRALS
LOGIN-REFERRALS AUTH=LOGIN AUTH=ANONYMOUS THREAD=ORDEREDSUBJECT
. OK Completed
%
```

Most IMAP daemons answer the CAPABILITY command with the name of the company or organization that released the daemon. The UW IMAP server doesn't identify itself as belonging to any particular organization. If the server's version number is of the format "v12.NNN" (versions prior to UW IMAP 2000) or the

format "v2000.NNN" (UW IMAP 2000), it's most likely the UW server. The SCAN, SORT, and THREAD capabilities are UW experimental capabilities; their presence also suggests that the server is the UW server.

You can also look for artifacts of the UW *phile* driver. The *phile* driver is likely to be found only in software based on the UW C-Client library:

```
% strings /usr/local/sbin/imapd | egrep phile
phile
phile
phile recycle stream
%
```

Be careful, though, because there are likely to be other IMAP servers out there based on the C-Client libraries.

Probably the most interesting and significant fact about the UW IMAP server is that it was written by Mark Crispin, the progenitor of IMAP itself. It's fair to say that Crispin is to the IMAP community as Linus Torvalds is to the Linux community. Crispin invented IMAP entirely on his own, when he was asked to build a distributed mail system with no guidance. He wrote the original IMAP server from scratch in DEC-20 assembly language in 1985. IMAP's early design was strongly influenced by the DEC-20 mail system, of which Crispin was also the primary developer and maintainer. The first nine years of IMAP's development can be attributed entirely to Crispin.

History

The UW IMAP server was first written in November 1990. It took Mark Crispin only a few days to write *imapd*, because he based it on the C-Client library. Quoting Crispin, "If you have the right underlying structures and tools, any project can be reduced to triviality."

imapd didn't include support for traditional Unix mailbox format in its original release. The users of legacy Unix mail programs such as */bin/mail* were clamoring for Unix mailbox support in *imapd*, though. Crispin was reluctant to add Unix mailbox support because of its limitations, such as the inability to have a mailbox open by multiple users simultaneously. He added this support to a later release, though, and support for other mailbox formats followed soon after. The current preferred mailbox format, mbx, continues this tradition.

It's interesting that, from its inception, UW *imapd* supported multiple, simultaneous access to a single mailbox using tenex format, something that was considered impossible on Unix systems.

The C-Client Library

C-Client, also written by Crispin, was originally developed while he worked at Stanford beginning in 1988. C-Client is a C port of an LISP IMAP client, hence its name—it's the C client as opposed to the LISP client.

C-Client is a C API that implements IMAP and SMTP as well as numerous mailbox driver interfaces that read and write different mailbox formats, which we'll talk about in later chapters. The UW IMAP daemon is based on C-Client. Many IMAP client programs are also based on C-Client, including the popular PINE mail client.

You'll find excellent documentation on C-Client included in the UW IMAP source distribution in the file *docs/internal.txt*.

UW's Strengths

UW *imapd* is at once both a kit of sorts and a completely self-contained IMAP server. By kit, we mean that there are adequate mechanisms built into UW IMAP to extend it to use virtually any kind of mailbox or authentication scheme, often with only a relinking required. By self-contained, we mean that it already has support for most authentication and mailbox schemes you're likely to want to use.

Flexibility

The UW server takes a very different approach than the Cyrus server to dealing with mailbox formats. Cyrus understands only one mailbox format. To access a Cyrus mailstore, you're limited to access through the Cyrus server. The UW server, on the other hand, understands many different mailbox formats. It goes to great lengths to detect the format of an existing mailbox and work with it.

The flexibility in mailbox formats can be a major selling point for the UW server. Some sites have users who demand that mail be stored in standard Unix format so that it can be directly manipulated (e.g., with Unix tools like *grep* or with Unix mail clients like *elm*) without using IMAP. With the UW server, the site administrator has the option of leaving the mailstore open to direct access or locking it down to IMAP access only.

You may come across claims that the UW server doesn't perform as well as Cyrus. Most performance comparisons of UW to Cyrus are comparing apples to oranges. The default UW IMAP installation uses traditional Unix mailbox format, where the mailbox is a single file of concatenated messages. Cyrus, on the other hand, stores the mailbox in a slightly modified form of the MH format, where each message is stored in a separate file and tracked in an index file. Cyrus does not have to share its data with legacy mail software, so it doesn't have to make the compromises that the UW server does.

In general, Cyrus is faster than the UW server. However, the UW server can do some things (notably, expunge many messages and search text) faster than Cyrus because it doesn't have to deal with multiple files. Replacing the traditional Unix mailbox format with the *mbx* format helps close the gap. A huge benefit is the support for server-based sorting in the UW server. PINE users who use sorted views will notice much better performance from the UW server than from Cyrus.

So the real answer is, "it all depends."

Modularity

UW IMAP also provides system administrators with a modular system. If you want to write code to provide a value-added service or scale not easily available elsewhere, UW IMAP may be a good choice.

UW's Limitations

All this plug-and-play ability must, of course, come at some price. With UW IMAP, that price is lack of support for some important features and a certain degree of Unix-centricity. Note that we say these are limitations, not necessarily shortcomings. Ultimately the result of the limitations is that UW IMAP is a tightly focused server.

No Support for IMAP Quotas

UW IMAP does not support the RFC 2087 quota extension. That means that, instead of application-specific IMAP quotas, UW IMAP servers must rely on the underlying OS for quotas. The effect is that, with the default Unix MDA, a message delivered to an over-quota UW INBOX is bounced back to the sender. It is not deferred and reattempted later—it bounces hard. On an RFC 2087–compliant server, the message would be held in queue for *n* days and delivery would be reattempted periodically, until either the usage drops below the quota limit or time runs out and the mail is bounced.*

Many system administrators would prefer that UW IMAP included support for IMAP quotas (RFC 2087) because such support would allow a finer degree of granularity over allocating space in the mailstore to users. A user, for example, could have numerous unrelated quotas that apply to different parts of her personal mailstore. The IMAP quota specification provides for not just one quota, but entire hierarchies of quotas. User *joebob* might have a quota of 10 MB on his INBOX, a

* Such soft bouncing assumes that the system employs a RFC 2087–cooperative Mail Delivery Agent (MDA), such as Cyrus's *deliver* program. The number of days mail is held in queue is, strictly speaking, a Mail Transport Agent (MTA) thing.

separate overall quota of 30 MB on his entire collection of mailboxes including the INBOX, and a further quota of 10 MB on his sent-mail folder.

Many sites' quota needs may be met just fine by using the standard Unix quota facility. Because incoming mailboxes are frequently on different partitions than users' personal mail folders, it's straightforward to use something like *edquota*(1) to assign users one quota for the */var/spool/* partition, which might hold the users' INBOXes, and another quota for the */export/home* partition, which might hold users' personal mailboxes.

The RFC 2087 quota behavior is kinder to the sender and the recipient at the expense of storage and processing resources used to retain and attempt redelivery of the overflow mail in local queues. The UW behavior is kinder to local system resources at the expense of message sender and recipient angst.

You may be stuck on the horns of a dilemma in which you can't do without the distinctive features of UW IMAP, but you've absolutely got to have quotas that result in Cyrus-style soft bounces. In that case, you might be able to cobble something together by chaining two MTAs together, one that houses your destination mailboxes, and the other a queuing host that initially receives all the incoming mail bound for the mailbox host. It's feasible that the MTA on the queuing host could be modified to retain quota bounces from the destination host, retaining them for later attempts. Such wizardry should only be attempted by the pure of heart and strong of spirit, preferably after massive coffee consumption.

No Support for IMAP ACLs

At the time this book was written, UW included no support for IMAP ACLs (RFC 2086). That lack of support means that, instead of using an IMAP client's built-in mechanism to grant or remove access to mailboxes, a user must rely on the operating system's access control mechanisms (e.g., change permissions on a file using *chmod*). Support for IMAP ACLs in the UW server is in the design phase now and will be included in upcoming releases.

Relies Heavily on Unix

Out of the box, UW *imapd* relies very heavily on the underlying Unix operating system structure. For example, you can't have an IMAP user that doesn't also exist in */etc/passwd*. You can't assign a quota that doesn't already exist on a Unix filesystem. And you can't create a hierarchy of mailboxes that doesn't correspond to files and directories in the underlying filesystem.

Note that everything we say about UW *imapd* applies to UW *imapd* straight out of the box unless otherwise noted. Because the server is so flexible, you can make it fit nearly any requirements it doesn't already fit, with some time and code.

UW IMAP Concepts

Black Box and Clearbox Models

We mention black box* and clearbox modes only because, once you unpack the UW IMAP development environment, you're likely to run across several references to a black box mode in the accompanying documentation. You may also see mention of it in discussion lists and, if you're so inclined, in the server source code itself. Clearbox mode, the default, is the mode in which we strongly urge you run UW IMAP. Black box mode has to be explicitly enabled.

References to black box IMAP servers usually occur in two contexts. One context refers to a machine whose mission is solely that of an IMAP server—it does not provide shell, Web, or other Internet services. In the other context, black box refers to a configuration mode and namespace intended solely for internal use at the University of Washington. It's easy to get the impression from the UW documentation that black box is a valid configuration choice for users outside the University of Washington. It's not. UW black box mode was designed for a single server at UW. It is not a general mechanism, nor is it intended as such. UW's public servers don't even use black box mode, even though they are black boxes. Later, we'll give you chapter and verse of the warning from Mark Crispin himself.

In the general context of Internet messaging, though, the black box distinction is usually loose enough to include any server that:

- Has no interactive logins (with the exception of the administrators' accounts)
- Serves up only IMAP and closely related protocols (SMTP, for example)

UW IMAP Namespace

Having hopefully scared you away from attempting to operate the UW server in black box mode, let's talk briefly about the UW IMAP namespace. In the context of IMAP, a namespace is a convention for describing the location of a mailbox in relation to other mailboxes. Some familiar namespaces in other contexts include the Usenet News newsgroup namespace and the DNS domain namespace.

IMAP4 (RFC 2060) doesn't specify a namespace. There is an IMAP Namespace document (RFC 2342) that presents two alternative approaches to the namespace: the Complete Hierarchy model and the Personal Mailbox model. The Personal Mailbox model is used by UW IMAP as the namespace that describes the location

* Mark Crispin mentioned these briefly in the course of a discussion on the UW IMAP mailing list concerning the IMAP server namespace in 1997. See *http://www.washington.edu/imap/listarch/msg02743.html*.

of each individual user's mailboxes in relation to her home directory. UW IMAP uses the Complete Hierarchy model to name shared, public, and anonymous mailboxes relative to the root of the public namespace.

Issuing the IMAP NAMESPACE command to an out-of-the-box UW IMAP server results in something like the output below:

```
# telnet localhost imap
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
* OK localhost IMAP4rev1 v12.250 server ready
. login kwm xxxxxxxx
. OK Completed
. namespace
* NAMESPACE (("" "/")("#mhinbox" NIL)("#mh/" "/")) (("~" "/"))
(("#shared/" "/") ("#ftp/" "/")("#news." ".")("#public/" "/"))
. OK Completed
. logout
* BYE myhost.unt.edu IMAP4rev1 server terminating connection
. OK Completed
Connection closed by foreign host.
```

The NAMESPACE command returns a list of namespaces (e.g., `("#mh/" "/")`). Each namespace consists of the namespace's name and a hierarchy separator. The hierarchy separator is a character used in a mailbox name to delimit different levels in the mailbox hierarchy. Take the "~" namespace, for example: a mailbox in that namespace would have a name like *~/mail/saved-messages*.

In the previous example, the logged-in user *kwm* has at least eight namespaces available, most of which have "/" as a hierarchy separator, but one of which uses ".". Within the context of this example, then, *#shared/yippie/tie/yie/yea/* is a syntactically valid mailbox name, whereas *#news/comp/dcom/telecom* isn't.

Onward to the individual namespaces of UW IMAP:

INBOX

The INBOX appears as the first token, `("" "/")`, in the output of the NAMESPACE command. As specified in section 5.1 of RFC 2060, this special reserved name is a token that stands for "the primary mailbox for this user on this server." UW IMAP even uses some elementary heuristics to try and figure out what mailbox type (Unix, mmdf, mbx, etc.) the user is given to using so it can apply the appropriate mailbox driver.

#mhinbox

This special name tells UW IMAP to assume that the INBOX is in MH format. Because the INBOX occupies only one level of the hierarchy, there is no hierarchy separator, hence the NIL. UW IMAP includes MH implementation to support legacy mailboxes only. If you really want to use an MH format INBOX, then you have to use *#mhinbox*.

#news

This name permits reading of a local news spool. *#news.alt.sysadmin.recovery*, for example, would present the contents of the *alt.sysadmin.recovery* newsgroup as if it were a read-only IMAP mail mailbox. You can also substitute the stock driver for one that makes UW IMAP use *#news* as a proxy NNTP server.

#ftp

#ftp is a way to download files from the anonymous FTP directory using IMAP. The mailbox *#ftp/pub/messages* actually points to and serves up the file *~ftp/pub/messages*. This capability is typically used to serve up one or more mailboxes to users who aren't necessarily provisioned at your site.

#public

Another anonymous login name. This is different from *#ftp* in that it serves a directory dedicated to IMAP. This would be good for data you want to serve through IMAP only, not through FTP.

#shared

#shared is like *#public*, except that *anonymous* IMAP users cannot access *#shared*, whereas they can access *#ftp* and *#public*.

~/ (tilde expansion)

UW IMAP supports the use of a tilde (~) to indicate files in the home directory of the logged-in user. Ordinarily, users of multiuser systems keep their mailboxes in a directory somewhere south of their top-level home directory. Something along the lines of *~/mail/* or *~/Mail/* is probably more useful than browsing through the top-level files in the user's home directory.

Remote names

Using the remote names convention, UW IMAP can be turned into an IMAP, POP3, or NNTP proxy. A mailbox specification like *{news.myisp.net/nntp}comp.mail.imap* can be used to present a Usenet newsgroup in the same way any mailbox is presented in your MUA. Likewise, you can use the same method to pull up your POP3 mailbox or another IMAP mailbox.

C-Client Drivers

The University of Washington IMAP can be thought of as the *emacs* of IMAP servers. *emacs*'s power lies in the fact that it knows little or nothing about individual terminal types—instead, it interfaces to the *curses* API and the Termcap/Terminfo capability databases, which are easy to use and know about a wide variety of terminals. At its core, UW IMAP doesn't access mailboxes directly. Instead, it makes calls to the C-Client API. C-Client is an API that uses numerous modular drivers to perform mailbox operations (see Figure 10-1).

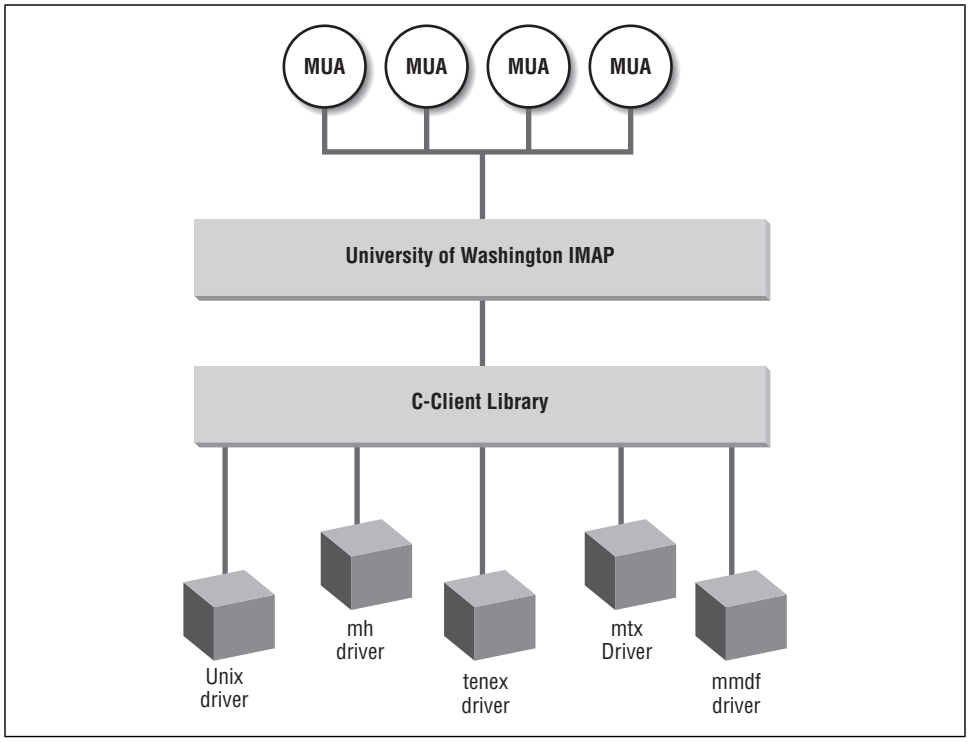

Figure 10-1. C-Client API

Modularity is one of the primary strengths of UW IMAP. In the default clearbox mode, which is the mode all but a handful of people should use, the server goes through some simple heuristics to try and match up the mailbox in question with the correct mailbox driver. In order, it tries to match the mailbox with mbox, mx, mbx, tenex, mtx, mmdf, Unix, and, finally, just plain flat file formats. One of the drivers is selected if certain expected files are present and are in the anticipated format. For example, if mbox, mx, and mbx selection fail, the server would next check to see if file *~/mail.txt* exists and is either empty or in tenex format. If it did exist and it was empty or in tenex format, then UW IMAP would select the tenex driver and proceed.

Authentication and Authenticator Modules

Another strength of UW IMAP is its ability to use additional authenticator modules. By default, the MD5 and Unix standard authentication types are included at build time. Inclusion of Kerberos 4, OTP, S/Key, or other types of authentication schemes can be accomplished by adding a third-party authenticator module.

Your selection of an authentication method directly impacts two features of your mail service: scalability and security. It affects your scalability because different components of different authentication mechanisms begin to break down at different sizes. The flat file, */etc/passwd*, begins to become quite difficult to provision after a certain number of users. Exactly how many depends on the operating system; on some systems it is as low as 10,000 users. The kernel structure defining the number of hard links to any given directory is a fixed number of bits wide (usually 32). That places a fixed limit on the number of users you can have in a non-hashed home directory structure. Component limits like this impact your choice of what authentication mechanisms scales best for you.

The far more obvious impact of your authentication mechanism choice is on security. Depending on the scheme you use, you could be tossing a copy of your front door key into the street every time you enter your home. If you use Unix authentication, you're exposing yourself to the single biggest vulnerability on most IMAP servers: the use of cleartext passwords. More and more people are hopping on the Internet. An increasing number of those people are running mail clients to check for new mail (and sending a cleartext password over the Net in the process) every 5 or 10 minutes. The Net is becoming a gold mine for passwords.

The IETF is working to remedy the situation by refusing to standardize protocols that rely on cleartext authentication. In the meantime, you can do your part to help. If you simply must use a cleartext password, engineer a provisioning system so that it precludes reusing your mail password on any other system. Or employ a virtual private network (VPN) mechanism, such as Point-to-Point Tunneling Protocol (PPTP), Secure Shell (SSH), or Secure Socket Layer (SSL), to encrypt the path between your mail client and the mail server. Webmail is another option. You could install a web-based IMAP client (such as IMP or WING*), run the HTTP server with SSL, and keep the IMAP server and the web server on the same machine. Passwords will then travel from the web server to the IMAP server over the machine's loopback interface and will not travel the Net.

Addition of both drivers and authenticators is accomplished by including them in the `EXTRAAUTHENTICATORS=` or `EXTRADRIVERS=` lines of the top-level makefile. Some drivers or authenticators may require that you add or patch code in your build tree before rebuilding IMAP to accommodate their product.

---

* See Chapter 5, *Web-Based IMAP Clients*, for a discussion of webmail.

Logging

There are various ways to keep track of what's going on with your UW server. It's likely that you'll want to log its messages to *syslog*.

Here are the kind of *syslog* entries you can expect from UW IMAP. Typically, they're logged to facility *mail* and level *debug*.

The following entry is generally the first log entry you'll see. It means the LISTEN on the IMAP TCP port established a connection with a remote machine, in this case, 10.120.220.41:

```
Jun  9 14:55:14 nec imapd[14506]: imap service init from 10.120.220.41
```

The following sequence typifies the trail left behind by IMAP login successes and failures:

```
Jun  5 23:24:40 nec imapd[27712]: Login user=kwm host=raz.unt.edu [10.120.110.4]
Jun  9 11:33:49 nec imapd[11568]: Authenticated user=kwm host=raz.unt.edu
    [10.120.110.4]
Jun  7 22:32:36 nec imapd[5310]:  Login failure user=ANONYMOUS host=localhost
    [127.0.0.1]
Jun  5 23:19:07 nec imapd[27668]: Logout user=kwm host=grove.acad.unt.edu
    [10.120.220.41]
```

As usual, some stuff will always fall on the floor:

```
Jun  9 14:39:53 nec imapd[14429]: Autologout user=??? host=raz.unt.edu
    [10.120.110.4]
Jun  5 23:17:26 nec imapd[27652]: command stream end of file,
    while reading char user=??? host=localhost [127.0.0.1]
Jun  9 16:09:39 nec imapd[19284]: AUTHENTICATE LOGIN failure
    host=grove.acad.unt.edu [10.120.220.41]
Jun  5 23:17:24 nec imapd[27652]: Missing command before authentication
    host=localhost [127.0.0.1]
Jun 10 00:48:44 nec imapd[29142]: Connection reset by peer, while reading
    line user=kwm host=grove.acad.unt.edu [10.120.220.41]
```

Installing TCP Wrappers (*tcpd*) is also a good step. The *tcpd* wrapper daemon has a facility for variable expansion in commands given at connect time. For example, if you put an entry like the following in your */etc/hosts.allow* file:

```
ALL: ALL: (logger -p auth.debug -t "tcp_wrapper[$$]" \
"client(%a), \
client info(%c), \
daemon(%d), \
client address (%h), \
client hostname (%n), \
daemon PID (%p), \
server info (%s), \
client username (%u).\
")&
```

your *syslog* log file will then begin to accumulate log entries like the following:

```
Jun 10 23:26:45 nec tcp_wrapper[1961]: client(10.120.220.41),
    client info(grove.acad.unt.edu), daemon(imapd),
    client address (grove.acad.unt.edu), client hostname (grove.acad.unt.edu),
    daemon PID (1960), server info (imapd@raz.unt.edu),
    client username (29009).
```

Of course, there's the ultimate in logging—the protocol analyzer. One non-open source alternative that is included with every copy of Solaris is *snoop*. The following is a command line that would help watch an IMAP session between two hosts. The *–x 54* tells *snoop* to dump the contents of each packet in classical hexdump format with hex on the left and the ASCII translation on the right, but skipping the first 54 bytes of overhead. The *port imap* means "display all packets using the port assigned to 'imap' in the */etc/services* file." It's assumed that, barring the use of *–d <device>*, the primary network interface is used.

```
# snoop -x 54 port imap

grove.acad.unt.edu -> raz.unt.edu  TCP D=143 S=49462
    Ack=3848578507 Seq=3307990424 Len=17 Win=8760
           0: 3030 3030 3030 3133 204c 4f47 4f55 540d    00000013 LOGOUT.
          16: 0a                                        .

 raz.unt.edu -> grove.acad.unt.edu TCP D=49462 S=143
    Ack=3307990441 Seq=3848578507 Len=0 Win=8760

 raz.unt.edu -> grove.acad.unt.edu TCP D=49462 S=143
    Ack=3307990441 Seq=3848578507 Len=89 Win=8760

           0: 2a20 4259 4520 6e65 632e 756e 742e 6564    * BYE raz.unt.ed
          16: 7520 494d 4150 3472 6576 3120 7365 7276    u IMAP4rev1 serv
          32: 6572 2074 6572 6d69 6e61 7469 6e67 2063    er terminating c
          48: 6f6e 6e65 6374 696f 6e0d 0a30 3030 3030    onnection..00000
          64: 3031 3320 4f4b 204c 4f47 4f55 5420 636f    013 OK Co
          80: 6d70 6c65 7465 640d 0a                 mpleted..
```

A bit of street wisdom here—commercial protocol analyzers are some of your more expensive toys. You might want to exploit the full capability of things like *snoop*, *tcpdump*, *ethereal*, and other open source or easily available packages before you decide that the added bit of functionality is worth the cost.

Does UW IMAP Match Your Needs?

Many sites select UW for its simplicity. For many system administrators, the ability to just slap an entry into your *imapd.conf* file and have a functional IMAP server with literally no time spent on configuration is highly attractive.

If a large proportion of your Unix users access traditional Unix mail directly or with native mail clients, UW will allow you to gradually move the users from traditional Unix mail access methods to IMAP. UW lets you add IMAP to the mix without taking anything away. As we'll see a few chapters later, the Cyrus IMAP server requires "all or nothing." The only way to access Cyrus mailboxes is via IMAP—direct access is not possible because the mailboxes are owned by the Cyrus system and are invisible to normal users.

If you have mailstores in two or more formats or you need to have shell account access to your mailstore, UW is probably the best choice. If you have a modest number of users (under 15,000, say) and can suitably work around the lack of ACL and IMAP quota support, then UW IMAP will probably work for your site.

If you need something that scales to many users more easily, then Cyrus is the best choice. Cyrus fits better because of ACL and IMAP quota support. If you have no need for simultaneous shell account and IMAP access to the same mailboxes, then you would probably be better served by Cyrus.

A typical growth path for an email infrastructure goes something like this:

- The first email system is usually something largely proprietary, either loosely based on Internet standards or not based on Internet standards at all: GroupWise, Notes, or Exchange.
- The day of open standards reckoning arrives, and open source Internet email systems start replacing the old proprietary ones. Until recently, this was very likely to be a POP3 server. Now, that's only somewhat likely.
- The single-mode POP3 server starts to grate on users' nerves when they have to search around for the PC they were using when they last downloaded individual pieces of mail. They will beg for IMAP as they struggle with mail in Netscape on one machine, Pegasus mail on another, and a handful of messages they accumulated during their vacation in Yahoo! Mail.
- The company has been bought out five times in as many years, with a couple of mergers thrown in for fun. UW IMAP isn't supporting the patchwork quilt mail system as well as you'd like, so you design a new system from scratch using Cyrus IMAP that should last through the next dozen mergers.

Everyone has heard the old saw: "The only hard and fast rule is that there are no hard and fast rules." It applies equally well to choosing an IMAP server. We're not just trying to play it safe by saying that. UW IMAP and Cyrus are both strong packages with a great deal of overlap and some definite cultural differences. A good way to think of it might be to see UW IMAP as being the Apache of Unix IMAP servers and Cyrus as being the Netscape. Large and small sites use each, but trying to say which is better is like choosing which of the Mac or the PC is better.

To help you decide whether you prefer UW or Cyrus, Table 10-1 gives a comparison of the features listed with the IMAP CAPABILITY command of Cyrus v1.5.14 and UW IMAP v4.7.

Table 10-1. A Comparison of Some Cyrus and UW Features

Feature	Cyrus IMAP v1.5.19	UW IMAP v4.7
Protocol Revision	IMAP4rev1	IMAP4rev1
IMAP4 ACL Extension (RFC 2086)	✓	Not supported
IMAP4 QUOTA Extension (RFC 2087)	✓	Not supported
IMAP4 IDLE Command (RFC 2177)	Not supported	✓
IMAP4 Mailbox Referrals (RFC 2193)	Not supported	✓
IMAP4 Login Referrals (RFC 2221)	Not supported	✓
IMAP4 Namespace (RFC 2342)	✓	✓
IMAP SORT Extension (Draft)	Not supported	✓
IMAP THREAD Extension (Draft)	Not supported	✓

Here's a more detailed description of the features listed in Table 10-1:

RFC 2086 (ACL Extension)

RFC 2086 provides a system for associating access controls with users and mailboxes that are wholly separate from the rights that may be present in the underlying operating systems. Access rights such as lookup, read, keeping seen/unseen information across sessions, write, insert, post, create, delete, and administer are associated with user/mailbox pairs.

RFC 2087 (Quota)

RFC 2087, known as the Quota Extension, permits servers to place limits on user resource usage. In theory, limits may be placed on any practical resource consumed by IMAP operations. In practice, quotas are usually limited to restrictions on disk usage. Quotas are not necessarily all encompassing, but are applied to one or more "quota roots," which are arbitrary points in a user's namespace, below which her quotas are inherited unless new quota roots with different sets of quotas are encountered.

RFC 2177 (Idle)

RFC 2177 provides a way for the IMAP server to know that it can asynchronously give "EXISTS" responses without being periodically polled to do so (e.g., with a NOOP command). The client can go into idle mode, then after some period of time, return from idle mode by sending a DONE. After leaving idle mode, the client receives all the EXISTS responses that may have queued up during the idle interval because the size of the SELECTed or EXAMINEd mailbox changed.

RFC 2221 (Login Referral)

RFC 2221 gives an IMAP server the ability to authenticate a user, then direct him to a new mailbox, essentially saying, "Yeah, I know you're legit, but you're supposed to get your mail down the hall now." Login Referral capability is useful for sites that split their user base out over numerous mail servers, using the front-door mail processor as a kind of connection multiplexer.

RFC 2193 (Mailbox Referral)

The Mailbox Referral capability, as specified in RFC 2193, gives a kind of symbolic linking capability to individual mailboxes. Thought of another way, it does for mailboxes what RFC 2221 does for entire sessions. With this capability, users may find that their shared folders, or perhaps their personal folders over a given size threshold, have been moved to *BigMailServer@example.com* and will be automatically redirected accordingly.

RFC 2342 (Namespace)

The Namespace capability (RFC 2342) provides a method for clients to discover the symbolic descriptions of the various namespaces used on a given server. Servers that implement this command probably require less user configuration on the client end to get their users up and running.

Draft: IMAP SORT Extension

The IMAP SORT extension is a standard for sorting messages on the server instead of the client.

Draft: IMAP THREAD Extension

The IMAP THREAD extension provides for a threaded view with the server doing the threading instead of the client.

When choosing between Cyrus and UW, remember that, although Cyrus gives you one choice of mailstore format, UW gives you many choices. As you will read in the file *docs/drivers.txt,* in the UW server the various mailbox drivers vary in performance from very good to very poor and have some unique characteristics. It would be an unfair test to pit Cyrus against UW with one of its poorly performing mailbox drivers.

Unfortunately, although you might want to compare Cyrus with UW using its mh driver, the mh driver is listed as being a very poor performer. A better test would be to use the mtx or mbx driver, both of which permit concurrent read-write operations and are listed as very good performers. The University of Washington prefers mbx format.

Late-Breaking Note

When we went to press, the University of Washington was in the early stages of releasing the next major versions of their server, UW IMAP 2000. Here's a preview of the major differences between UW IMAP 2000 and the previous 4.x release:

- **Integrated SSL, TLS, and STARTTLS functionality**. SSL and TLS/STARTTLS functionality are integrated into UW IMAP 2000 by means of the OpenSSL package, available from *http://www.openssl.org/*. TLS is the next generation version of SSL. UW neither supplies nor supports the OpenSSL package, but the UW IMAP does dovetail with it to provide end-to-end encrypted IMAP service, with encryption of both authentication and data. A new document, *docs/SSLBUILD,* conveys the mechanics of how to bring up UW with this capability.

  There are two advantages to doing SSL/TLS this way rather than with *stunnel*. One of the benefits is process economy. Instead of each *imapd* process having a corresponding *stunnel* process, both the SSL/TLS and IMAP functions are carried out by the *imapd* process. Another is that instead of having only SSL functionality, the OpenSSL-enabled *imapd* has SSL and TLS functionality, which includes support for STARTTLS—which, in turn, enables a session to begin in unencrypted mode, then transition cleanly to encrypted mode.
- **Improved C++ Compatibility**. Historically, C++ developers who build their own C-Client applications and servers with the C-Client Toolkit have had a moderate amount of difficulty getting C-Client to build as well with C++ as with a standard C compiler. A new header file, *c-client.h*, greatly reduces or eliminates that problem.
- **Kerberos Version 5**. UW IMAP 2000 will support Kerberos v5 in both Unix and Win32 builds. If you build it for Windows 2000 or Windows ME, Kerberos v5 is enabled automatically. Kerberos v4 is not directly supported by UW in IMAP 2000.
- **The IMAP Administrator**. Starting in UW IMAP 2000, UW IMAP has the feature of an IMAP Administrator. By using this feature, a system administrator can log in by giving her user ID and password, but indicate a different user and actually be logged in as that user, although she is using her own password. If your site supports the "SASL authorization identity," you can use that to emulate *su* functionality in IMAP to let you troubleshoot another person's mailbox without having to know his passwords.

—Continued—

Only users who are in the Unix group *mailadm* can use this facility. If your site doesn't support the SASL authorization identity mechanism, you can separate your authorization identity from your authentication identity with an asterisk. The authorization identity is the identity of the user whose mailbox you are examining, and the authentication identity is your own user ID. The credentials would be the password you normally use to gain access to the server. For example, if you are user *helpdesk* and you are troubleshooting the mailbox for user *enduser*, you would log in as *enduser*helpdesk*. The "*" hack works for all UW IMAP authenticators, such as MD5 and Kerberos.

- **Support for the MULTIAPPEND extension**. In draft status at the time of publication, this extension permits an arbitrary number of messages to be appended to a mailbox in a single atomic action. They either all succeed or all fail uniformly. The draft is available at *http://search.ietf.org/internet-drafts/draft-crispin-imap-multiappend-01.txt*.