*In this chapter:*
- *IMAP Session Concepts*
- *IMAP Components*
- *An IMAP Session Play-by-Play*

# 3

# *Anatomy of an IMAP Session*

This chapter covers the conceptual middle ground between a layman's understanding of the IMAP protocol and complete coverage, as contained in RFC 2060 (IMAP4rev1). This chapter will provide enough information to arm you to troubleshoot most IMAP problems and evaluate most clients, but not enough to write your own client or troubleshoot some of the stickiest dilemmas. For those situations, you would be much better off using the RFC 2060 documentation and the RFCs for the extensions your server professes to use.

## *IMAP Session Concepts*

This section covers how a client talks to a server, the details of what an IMAP session looks like, and how we captured that information on our network.

### *IMAP Is Line-Oriented*

You may occasionally see IMAP referred to as a "line-oriented" protocol. All this means is that the conversation between the IMAP client and server is transmitted in the form of character strings that end with CRLF. Line-oriented protocol sessions are easy to follow: a command is sent as a line of text to the server, and the server returns its response as a line of text. Line-oriented protocols are easy to learn and understand for the very same reason.

In fact, commands that make up a line-oriented protocol are frequently so understandable that a user can use the commands to masquerade as an IMAP client. As an example, consider just about anyone with access to a version of *telnet* that lets you specify the target port. The Telnet protocol, by definition, uses TCP port 23, but most Telnet software can usually be directed at alternative ports. This means that a user can *telnet* to TCP port 25 and trick an SMTP server into interacting with

him as if she were an SMTP client. It's just as easy to for a user to *telnet* to TCP port 143 (IMAP) and interact with an IMAP server as if he were an IMAP client.

Why do we bother telling you this? Line-oriented protocol spoofing, apart from being a common hacking tool, is a helpful troubleshooting tool. If you're having difficulty getting a given client and server to talk, one approach is to take the client out of the equation. If you spoof IMAP from a *telnet* session, you can observe the commands being sent to and from the server and classify problems as either client or server problems.

The primary difference between IMAP and other line-oriented TCP protocols like SMTP, NNTP, and POP3 is that each command from the client is preceded by a short alphanumeric string, called a *tag*. The purpose of the tag is to help the IMAP client keep track of which response goes with which command. Once a connection is opened, all tags generated by the client must be unique until the connection is closed. When the server responds to a command, it attaches the tag that the client sent to its response, as illustrated in Figure 3-1.

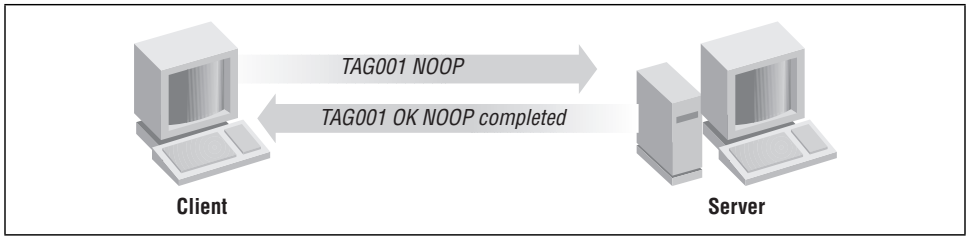


*Figure 3-1. Tagged commands and responses*

We mentioned that only the server's completion result is tagged. There are actually two classes of server response: *tagged response* and *untagged response*. Untagged responses convey data. They may or may not be in reaction to a command from the client. An untagged response can be thought of as a type of command from the server to the client to update state on the client. An untagged response, like its name implies, is not preceded by a tag. Instead, it is preceded by the "*" character.

## Story of an IMAP Session

*inetd* listens for IMAP requests on TCP port 143.* Strictly speaking, until the first time a client connects to your server on the IMAP port, the IMAP server isn't running at all. An IMAP server daemon (*imapd*) is spawned by *inetd* to respond to each new connection from an IMAP client. *inetd* accepts the client's request for a

* The upcoming release, Version 2.0, of the Cyrus IMAP server does not use *inetd*, but instead, runs as a daemon.

connection, then passes the I/O stream off to the IMAP server. The client continues to send commands though port 143 to the IMAP server.

The IMAP server, it should be noted, knows nothing about IP, TCP, or networking. All it knows is how to communicate through STDIN and STDOUT and how to access local mailboxes. From the beginning to the end of each connection, the IMAP server will respond to an arbitrary set of commands. While it may be perfectly legitimate for the client to connect, perform one command, disconnect, connect, perform one command, etc., it's terribly inefficient. We know of no clients that wasteful. Typically (at least during online modes) clients will connect, perform whatever business they need to perform, then disconnect, either when the user shuts down the client or when the server's timeout limit is reached.

Disconnected mode is slightly different. In disconnected mode, the client connects to the server, retrieves new mail that has arrived, moves mail around to various mailboxes if the user so desires, then disconnects as soon as possible, minimizing the amount of time spent online.

# *IMAP Components*

Here we discuss the components of IMAP to prepare us for our blow-by-blow protocol trace later in the chapter.

## *Modes*

As we saw in Chapter 2, *What Is IMAP?,* IMAP clients can operate in one of three modes: *offline* mode, *online* mode, or *disconnected* mode. While you're not likely to see any IMAP operations take place between the client and server that say, "I'm an online session" or "I'm on offline session," these modes are quite an important part of IMAP.

## *States*

First of all, the session can be in one of four states. Commands and responses are, in most cases, valid only in certain states (e.g., you can't issue the command to select a mailbox unless you're in the authenticated state). Although two or more sessions with a given mailbox can each be in different states, no single session can be in more than one state simultaneously. For example, when your client initially connects to the server and you're in the non-authenticated state, you authenticate successfully, leave the non-authenticated state, and enter the authenticated state. Here's a description of the four states. A diagram of the relationship between the states is shown in Figure 3-2.

*Non-authenticated state*
: The non-authenticated state begins immediately when a connection starts.

*Authenticated state*
: The authenticated state begins when the client authenticates successfully. It may also begin after an error in selecting a mailbox.

*Selected state*
: The selected state begins once a mailbox has been selected successfully.

*Logout state*
: The logout state begins when the client sends the LOGOUT command or the server unilaterally decides to close the connection (e.g., when the session reaches the inactivity time-out limit). The logout state lasts only long enough for the server to close the TCP connection.

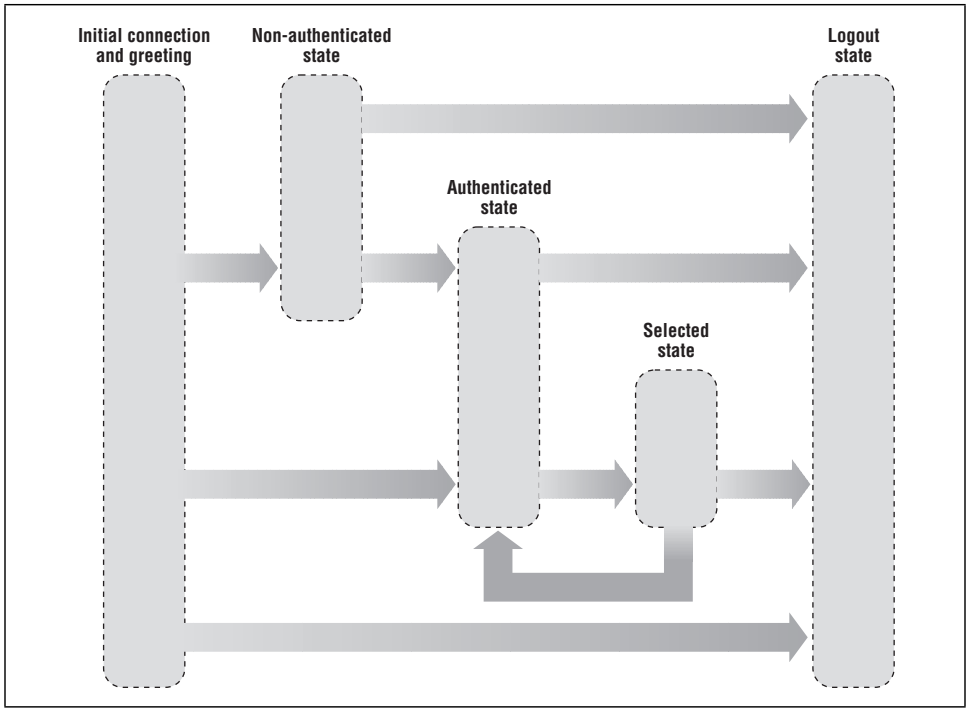


*Figure 3-2. IMAP state diagram*

## Mailboxes

Although there are extensions to provide attributes to mailboxes (such as access control lists), in the core IMAP protocol all the interesting attributes are associated with the messages themselves, not the mailboxes.

## *Messages*

There's much more to an email message than the body content. Attributes can range from core values like message numbers and headers to a road map of the MIME contents contained in the message.

*Message sequence number*

Message sequence numbers begin with 1 and continue sequentially up to the number of messages in the mailbox. Sequence numbers can be reassigned during a session. For example, when a message is deleted and expunged, each message with a higher sequence number than the deleted message is decremented by 1.

*UIDs*

The UID (Unique Identifier) is a 32-bit message identifier that is guaranteed to be unique within a mailbox. UIDs are preserved across sessions and are used to allow a client that operates in disconnected or offline mode to synchronize its state with the server. Message UIDs are assigned to messages in ascending order, but are different from message sequence numbers in that UIDs do not necessarily ascend contiguously. Although UIDs are mentioned for completeness, message sequence numbers are what you'll actually use when testing and troubleshooting IMAP sessions. You'll see mention of UIDVALIDITY later in this chapter. UIDVALIDITY is the unique identifier associated with a mailbox, not with a message. UID is associated with a message.

*Flags*

Each message in an IMAP mailbox may have zero or more flags. Flags are tokens that carry information about the message and are usually used to preserve message attributes between IMAP sessions. Several flags are defined. The message flags defined by IMAP are shown in Table 3-1.

*Table 3-1. IMAP Message Flags*

| Flag | Meaning When Flag Is Set |
|---|---|
| *Answered* | The message has been answered. |
| *Deleted* | The message has been marked for deletion. |
| *Draft* | The message is partially composed and being saved for later revision before sending. |
| *Recent* | The message is "recent"—the current session is the first session to see it. It will not be flagged as "recent" in subsequent sessions. |
| *Flagged* | The message has been marked as "important." |

*Internal date*
: The internal date reflects the date and time the message was received by the IMAP server. It is not the same attribute as the date value in the RFC 822 header.

*Size*
: The message size expressed in number of octets (8-bit bytes) in the message, expressed in RFC 822 format.

*Envelope structure*
: The envelope structure contains a condensed representation of the RFC 822 header.

*Body structure*
: The body structure contains a condensed representation of the MIME information contained in the message.

# An IMAP Session Play-by-Play

In this section we show and describe an actual IMAP session play-by-play. Our goal is to familiarize you with the most common IMAP operations. Knowing the operations will help you quickly troubleshoot problems independently of the client or server you're working with.

In our examples, we use the *tcpflow* program (see Chapter 18, *IMAP Tools*, for information on where to obtain *tcpflow*) to examine the IMAP client-server interactions. *tcpflow* is a special protocol-analysis program that permits you to watch a conversation take place between a TCP-based client and server. *tcpdump* or your favorite protocol analysis program would work equally well.

Our session was generated in a Telnet session to the IMAP port on the server:

```
% telnet localhost imap
```

Commands are shown in bold, responses from the server are shown in plaintext:

```
A00001 CAPABILITY
* CAPABILITY IMAP4REV1 MAILBOX-REFERRALS LOGIN-REFERRALS AUTH=CRAM-MD5
A00001 OK Completed
```

Each command is preceded by an arbitrary tag. As we mentioned earlier, the tag is an arbitrary string that the server "tags" its responses with. The purpose is to help the client keep track of which response goes with which command. Appendix C, *IMAP Commands*, provides a complete list of IMAP commands. The commands used in the sample session are, for the most part, self-explanatory. In any case, they're discussed in the explanation following Example 3-1, which shows the client/server conversation during an actual IMAP session and discusses each request and response.

*Example 3-1. Simple IMAP Session*

```
ROOT@Server # tcpflow -c 'host Client and port 143'
tcpflow[3000]: listening on le0
Server: * OK Server Cyrus IMAP4 v1.5.19 server ready

Client: 00000000 CAPABILITY007
Server: * CAPABILITY IMAP4 IMAP4rev1 ACL QUOTA LITERAL+ NAMESPACE UIDPLUS \
        X-NON-HIERARCHICAL-RENAME NO_ATOMIC_RENAME UNSELECT
        00000000 OK Completed

Client: 00000001 LOGIN dianna "xxxxxxxx"
Server: 00000001 OK User logged in

Client: 00000002 SELECT INBOX
Server: * FLAGS (\Answered \Flagged \Draft \Deleted \Seen)
        * OK [PERMANENTFLAGS (\Answered \Flagged \Draft \Deleted \Seen \*)]
        * 1 EXISTS
        * 0 RECENT
        * OK [UNSEEN 1]
        * OK [UIDVALIDITY 929804083]
        00000002 OK Completed

Client: 00000003 NOOP
Server: 00000003 OK Completed

Client: 00000004 FETCH 1 FLAGS
Server: * 1 FETCH (FLAGS ())
        00000004 OK Completed

Client: 00000005 FETCH 1 UID
Server: * 1 FETCH (UID 26888)
        00000005 OK Completed

Client: 00000006 FETCH 1 (ENVELOPE BODY.PEEK[HEADER.FIELDS \
        (Path Message-ID Newsgroups Followup-To References)] INTERNALDATE \
        RFC822.SIZE FLAGS)
Server: * 1 FETCH (FLAGS () INTERNALDATE "5-Mar-2000 10:58:50 -0600" \
        RFC822.SIZE 853 ENVELOPE ("Sun, 5 Mar 2000 11:00:52 -0600 (CST)" \
        "Testing" ((NIL NIL "drm" "nec.unt.edu")) \
        ((NIL NIL "drm" "nec.unt.edu")) ((NIL NIL "drm" "nec.unt.edu")) \
        ((NIL NIL "dianna" "europa.acs.unt.edu")) NIL NIL NIL \
        "<Pine.GS4.4.10.10003051100260.3034-100000@nec.unt.edu>") \
        BODY[HEADER.FIELDS (Path Message-ID Newsgroups Followup-To References)] \
        {70} Message-ID: Pine.GS4.4.10.10003051100260.3034-100000@nec.unt.edu

        )
        00000006 OK Completed

Client: 00000007 FETCH 1 (BODYSTRUCTURE FLAGS)
Server: * 1 FETCH (FLAGS () BODYSTRUCTURE ("TEXT" "PLAIN" \
        ("CHARSET" "US-ASCII") NIL NIL "7BIT" 29 2 NIL NIL NIL))
        00000007 OK Completed
```

*Example 3-1. Simple IMAP Session (continued)*

```
Client: 00000008 FETCH 1 BODY.PEEK[HEADER.FIELDS (Resent-Date Resent-From \
        Resent-To Resent-cc Resent-Subject)]
Server: * 1 FETCH (BODY[HEADER.FIELDS (Resent-Date Resent-From Resent-To \
        Resent-cc Resent-Subject)] {2}

        )
        00000008 OK Completed

Client: 00000009 FETCH 1 BODY[1]
Server: * 1 FETCH (FLAGS (\Seen) BODY[1] {29}
        This is the message body.

        )
        00000009 OK Completed

Client: 0000000a STORE 1 +Flags (\DELETED)
Server: * 1 FETCH (FLAGS (\Deleted \Seen))
        0000000a OK Completed

Client: 0000000b NOOP
Server: 0000000b OK Completed

Client: 0000000c EXPUNGE
Server: * 1 EXPUNGE
        * 0 EXISTS
        * 0 RECENT
        0000000c OK Completed

Client: 0000000d NOOP
Server: 0000000d OK Completed

Client: 0000000e LOGOUT
Server: * BYE LOGOUT received
        0000000e OK Completed
```

What follows is a description of the play-by-play in the previous example. We'll use the IMAP command tags to refer to different parts of the listing:

*Tag 00000000*
: The client asks the server for its capabilities using the CAPABILITY command. The server responds with its capabilities.

*Tag 00000001*
: The client logs in the user "dianna" with plaintext password "xxxxxxxx" using the LOGIN command. The server responds that the user was successfully logged in.

*Tag 00000002*
: The client selects the INBOX. The server responds with several pieces of information:

— Flags and permanent flags that are defined for the mailbox (*\Answered*, *\Flagged*, *\Draft*, *\Deleted*, *and \Seen*)

— Number of messages in the mailbox (1)

— Number of messages that have arrived since the session started (0)

— Number of unseen messages (1)

— The UIDVALIDITY (929804083)

Finally, the server returns the command result (OK), indicating that the INBOX was selected successfully and that the mailbox is both read and write.

*Tag 00000003*

The client sends a NOOP, probably something that it does periodically to ping the server to reset the inactivity timer and make sure the connection doesn't time out.

*Tag 00000004*

The client uses the FETCH command to request a list of flags that are set on the message that corresponds to sequence number 1. The server answers that there are no flags set.

*Tag 00000005*

The client uses the FETCH command again, this time to request the UID number of the message corresponding to message sequence number 1.

*Tag 00000006*

The client asks the server to send some of message number 1's header information (the RFC 822 fields Path, Message-ID, Newsgroups, Followup-To, and References, the message internal date, the RFC 822 size, and the flags that are set on the message).

The server returns, not necessarily in the order requested, the data items requested. Each item is labeled appropriately for consumption by the client.

*Tag 00000007*

In this sequence, the client is eyeballing the body of the message. Fetching the BODYSTRUCTURE data item, in fact, is one of the more powerful capabilities of IMAP. It permits the client to retrieve the skeletal structure of the message without retrieving the message itself. That capability allows the client to exercise discretion by downloading some parts of a message, such as a short *text/plain* part, and not others, for example a very large *video/mpg2* part.

*Tag 00000008*

In this sequence, the client is apparently checking to see if the message was "bounced." If so, it would have various "Resent-" header lines. That isn't the case here, so the server returns blank values for the requested header fields.

*Tag 00000009*

The client requests the body for the message with sequence number [1]. The server complies without a lot of to-do.

*Tag 0000000a*

The client deletes message number 1, which involves setting the *Deleted* flag on the message. The flag is set using the STORE command. The server responds that the flag was set successfully, and hence, the message is now marked "deleted."

*Tag 0000000b*

The client sends another NOOP to keep the connection alive and poll the server for new mail.

*Tag 0000000c*

The client expunges, or permanently removes, all messages marked "deleted" using the EXPUNGE command. The server responds with the message sequence number of each expunged message (in this case, message number 1, the only message in the mailbox). It also responds that the number of messages in the mailbox (EXISTS) is now 0 and that the number of messages that have not been read (UNSEEN) is also 0.

*Tag 0000000d*

The client sends another NOOP, and the server responds with OK.

*Tag 0000000e*

Finally, the client sends the LOGOUT command. The server responds by saying "BYE" and closing the connection.

## *A POP3 Session for Comparison*

Veteran Internet messaging techs will recognize Example 3-2 at first glance. Once upon a time, all email was retrieved using POP3. Here's a POP3 session for comparison.

As was mentioned in Chapter 2, POP is much less complex than IMAP, but also has fewer features. Here's a list of differences between POP and IMAP that are evident in comparing the POP3 session (Example 3-2) with the IMAP session (Example 3-1):

- POP operates in offline mode only.
- It has only one mailbox per user, so there is no SELECT command in POP.
- POP has no extensions, such as a Quota or ACL extension.
- It's not possible to determine the structure of a message in POP—all messages are single, contiguous entities. The entire message body must be downloaded.

- POP allows for the selective download of headers and message body separately, and most servers support that option. It's unusual, however, for clients to take advantage of that option. It's not done in our example.
- POP doesn't handle new mail that is received while a POP session is already open. The new mail is unavailable until the current session is closed and a new one open.

*Example 3-2. Simple POP3 Session*

```
ROOT@Server # tcpflow -c 'host Client and port 110'
tcpflow[3829]: listening on le0
Server: +OK POP3 Server v7.59 server ready

Client: USER drm
Server: +OK User name accepted, password please

Client: PASS XXXXXXXX
Server: +OK Mailbox open, 1 message

Client: STAT
Server: +OK 1 1031

Client: LIST
Server: +OK Mailbox scan listing follows
1 1031
.

Client: RETR 1
Server: +OK 1031 octets
        Received: from Mercury.acs.unt.edu (mercury.acs.unt.edu [129.120.220.1])
        .by security.unt.edu (8.8.8/8.8.8) with ESMTP id FAA03831
        .for <drm@nec.unt.edu>; Sun, 27 Feb 2000 05:57:28 -0600 (CST)
        Received: from venus.acs.unt.edu (venus.acs.unt.edu [129.120.220.72])
        .by Mercury.acs.unt.edu (8.8.8/8.8.8) with ESMTP id FAA25269
        .for <drm@nec.unt.edu>; Sun, 27 Feb 2000 05:55:35 -0600 (CST)
        Received: from SUNFLOWER (rooster.themullets.net [209.223.13.243])
        .by venus.acs.unt.edu (8.8.8/8.8.8) with ESMTP id FAA25103
        .for <drm@nec.unt.edu>; Sun, 27 Feb 2000 05:55:35 -0600 (CST)
        Date: Sun, 27 Feb 2000 05:54:43 -0600
        From: Dianna Mullet <dianna@unt.edu>
        To: drm@nec.unt.edu
        Subject: This is subject.
        Message-ID: <2443111424.951630883@localhost>
        Originator-Info: login-id=; server=
        X-Mailer: Mulberry (Win32) [1.4.4, s/n U-301284]
        MIME-Version: 1.0
        Content-Type: text/plain; charset=us-ascii
        Content-Transfer-Encoding: 7bit
        Content-Disposition: inline
        Content-Length: 28
        Status:
```

*Example 3-2. Simple POP3 Session (continued)*

```
        This is the message body.


        .

Client: DELE 1
Server: +OK Message deleted

Client: QUIT
Server: +OK Sayonara
```

There are two reasons for diving down to the protocol level. First, when evaluating or comparing IMAP products, nothing is quite as helpful as benchtesting them against one another and watching what happens on the network. When clients, and perhaps servers, first started providing IMAP support, many grafted IMAP functionality on top of their POP engines. Additionally, because IMAP is a protocol that isn't easily grasped in a simple five-minute cruise of the RFC, some vendors will take the short, error-fraught path and try to fake their way to IMAP compliance. You will run into these products, no doubt about it.

The other reason is that once you've chosen your IMAP products and put them into operation, the occasional problem will surface. Often it will be due to a lack of 100% IMAP compliance in either the server or client (usually the client). Unless you care to spend several weeks doing the finger-pointing dance with your vendors, the only way to get the critical smoking gun is to analyze the IMAP traffic on your network. In doing so you will be able to determine whether the problem lies with server or client, and exactly what the problem is.