Network Working Group
Request for Comments: DRAFT
draft-ietf-lemonade-compress-01.txt

Arnt Gulbrandsen
Oryx Mail Systems GmbH

June 2006

The IMAP COMPRESS=DEFLATE Extension

Status of this Memo

Copyright Notice

Abstract

The COMPRESS=DEFLATE extension allows an IMAP connection to be compressed using the DEFLATE algorithm, such that effective compression is available even when TLS is used.

Conventions Used in This Document

The key words "REQUIRED", "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", and "MAY" in this document are to be interpreted as described in "Key words for use in RFCs to Indicate Requirement Levels"

[KEYWORDS]. Formal syntax is defined by [ABNF] as modified by [IMAP].

In the example, "C:" and "S:" indicate lines sent by the client and server respectively.

Introduction and Overview

An IMAP server that supports this extension announces "COMPRESS=DEFLATE" as one of its capabilities.

The goal of COMPRESS=DEFLATE is to reduce the bandwidth usage of IMAP. On regular IMAP connections, the PPP or MNP compression used with many low-bandwidth links compresses IMAP well. However, when TLS is used, PPP/MNP compression is ineffective. TLS too may provide compression, but few or no implementations do so in practice.

In order to increase interoperation, it is desirable to have as few different compression algorithms as possible, so this document specifies only one. The DEFLATE algorithm is standard, widely available, unencumbered by patents and fairly efficient. Hopefully it will not be necessary to define additional algorithms.

The extension adds one new command (COMPRESS) and no new responses.

The COMPRESS Command

Arguments: Name of compression mechanism: "DEFLATE".
Direction: "UP", "DOWN" or "BOTH".

Responses: None

Result: OK The server will compress its responses (if the direction is DOWN or BOTH) and expects the client to compress its commands (if the direction is UP or BOTH).
NO The connection already is compressed, or the server doesn't support the requested mechanism, or the direction specified is unknown.
BAD Command unknown or invalid argument.

The COMPRESS command instructs the server to use the named compression mechanism ("DEFLATE" is the only one defined) for future commands and/or responses. If the direction specified is "UP", only commands are compressed. If the direction specified is "DOWN", only

For DEFLATE (as for many other compression mechanisms), the compressor can trade speed against quality. When decompressing there isn't much of a tradeoff. Consequently, the client and server are both free to pick the best reasonable rate of compression for the data they send.

The client MUST NOT send additional commands until it has seen the result of COMPRESS.

If both SASL/TLS and COMPRESS are in use, the data should be compressed before it is encrypted (and decrypted before it is decompressed), independent of the order in which the client issues COMPRESS, AUTHENTICATE and STARTTLS.

Example

This example shows a simple login sequence. The client uses TLS for privacy and [DEFLATE] for compression.

```
S: * OK [CAPABILITY IMAP4REV1 STARTTLS COMPRESS=DEFLATE]
C: a starttls
S: a OK
C: b compress deflate
S: b OK
C: c login arnt tnra
S: c OK
```

Compression Efficiency

IMAP poses some unusual problems for a compression layer.

Upstream is fairly simple. Most IMAP clients send the same few commands again and again, so any compression algorith which can exploit quotes works efficiently. The APPEND command is an exception; clients which send many APPEND commands may want to take special care.

Downstream has the unusual property that 3-4 kinds of data are sent, confusing all dictionary-based compression algorithms.

The first type is IMAP responses. These are highly compressible; zlib using its least CPU-intensive setting compresses typical responses to 25-40% of their original size.

The second is email headers. These are equally compressible, and benefit from using the same dictionary as the IMAP responses.

The third is email body text. Text is usually fairly short and includes much ASCII, so the same compression dictionary will do a good job here, too. When multiple messages in the same thread are read at the same time, quoted lines etc. can often be compressed almost to zero.

Finally, attachments (non-text email bodies) are transmitted, either in [BINARY] form or encoded with base-64.

When attachments are retrieved in [BINARY] form, DEFLATE may be able to compress them, but the format of the attachment is usually not IMAP-like, so the dictionary built while compressing IMAP does not help. The compressor has to adapt from IMAP to the attachment's format, and then back.

When attachments are retrieved in base-64 form, the same problems apply, but the base-64 encoding adds another problem. 8-bit compression algorithms such as deflate work well on 8-bit file formats, however base-64 turns a file into something resembling a 6-bit bytes in an 8-bit format.

A few file formats aren't compressible using deflate, e.g. .gz, .zip and .jpg files.

According to the author's measurements, the compression level used makes little difference. zlib's level 1 compresses IMAP almost as well as level 9, and for the receiver, level 1 seems to require (just a tiny bit) pmore CPU than level 9. Independent verification is strongly desired.

Implementation Notes

When using the zlib library (see [DEFLATE]), the functions deflateInit(), deflate(), inflateInit() and inflate() suffice to implement this extension.

Note that when using TLS, compression may actually decrease the CPU usage, depending on which algorithms are used in TLS. This is because fewer bytes need to be encrypted, and encryption is generally more expensive than compression.

A client can improve downstream compression by implementing [BINARY] and using FETCH BINARY instead of FETCH BODY.

A server can improve downstream compression if it hints to the compressor that the data type is about to change strongly, e.g. by sending a Z_FULL_FLUSH at the start and end of large non-text

literals (before and after '*CHAR8' in the definition of literal in RFC 3501, page 86).

A server can improve the CPU efficiency both of the server and the client if it adjusts the compression level (e.g. using the deflateParams() function in zlib) at these points. A very simple strategy is to change the level 0 to at the start of a literal provided the first two bytes are either 0x1F 0x8B (as in deflate-compressed files) or 0xFF 0xD8 (JPEG), and to keep it at 1-5 the rest of the time.

Formal Syntax

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) notation as specified in [ABNF]. Non-terminals referenced but not defined below are as defined by [ABNF] (SP, CRLF) or [IMAP] (all others).

Except as noted otherwise, all alphabetic characters are case-insensitive. The use of upper or lower case characters to define token strings is for editorial clarity only. Implementations MUST accept these strings in a case-insensitive fashion.

```
command-any =/ compress

compress    = "COMPRESS" SP algorithm SP ( "UP" / "DOWN" /
              "BOTH" )

algorithm   = "DEFLATE"
```

Security considerations

(As for [TLSCOMP] RFC 3749.)

IANA Considerations

The IANA is requested to add COMPRESS=DEFLATE to the list of IMAP extensions.

Credits

Quite a few people on the LEMONADE mailing list have offered comments, including Dave Cridland, Ned Freed and Tony Hansen. And various people in the rooms at meetings. Send me mail, I'll add you.

Open Issues

Both ends can already disable compression at any point by calling deflateParams(). The only missing feature is for the client to request that the server stop compressing - are there use-cases for that? It requires adding more server-side state, so I'm wary.

What text and numbers are needed wrt. compression levels? A bit of solid information is not amiss.

Normative References

[ABNF] Crocker, Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium, Demon Internet Ltd, November 1997.

[IMAP] Crispin, "Internet Message Access Protocol - Version 4rev1", RFC 3501, University of Washington, June 2003.

[KEYWORDS] Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Harvard University, March 1997.

[DEFLATE] Deutsch, "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, Aladdin Enterprises, May 1996.

[STARTTLS] Newman, C. "Using TLS with IMAP, POP3 and ACAP", RFC 2595, June 1999.

Informative References

[TLSCOMP] Hollenbeck, "Transport Layer Security Protocol Compression Methods", RFC 3749, VeriSign, May 2004.

Author's Address

Arnt Gulbrandsen
Oryx Mail Systems GmbH
Schweppermannstr. 8
D-81671 Muenchen
Germany

Fax: +49 89 4502 9758

Email: arnt@oryx.com

Intellectual Property Statement

Copyright Statement

Disclaimer of Validity

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.