Network Working Group Arnt Gulbrandsen
Request for Comments: DRAFT Oryx Mail Systems GmbH
July 2006

The IMAP COMPRESS Extension
draft-ietf-lemonade-compress-03.txt

Status of this Memo

Copyright Notice

Abstract

The COMPRESS extension allows an IMAP connection to be effectively and efficiently compressed.

Table of Contents

1. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [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.

2. Introduction and Overview

A server which supports the COMPRESS extension indicates this with one or more capability names consisting of "COMPRESS=" followed by a supported compression algorithm name as described in this document.

The goal of COMPRESS is to reduce the bandwidth usage of IMAP.

Compared to PPP/MNP compression, COMPRESS offers much better compression efficiency, and can be used together with TLS, SASL encryption, VPNs etc. Compared to TLS compression [TLSCOMP], COMPRESS has the following (dis)advantages:

- COMPRESS can be implemented easily by IMAP servers and clients. At present, TLS compression is not widely implemented. In the LEMONADE WG, the general consent is that libraries implementing TLS compression will not be available soon enough for LEMONADE.

- IMAP compression efficiency benefits from an API that permits flushing the compressor's dictionary at the right point. This is

practical for COMPRESS, whereas typical TLS libraries don't currently allow that.

- When a TLS librarly implements compression, all protocols that use TLS automatically are compressed (in LEMONADE's case, SMTP, IMAP, and some notification protocol), whereas COMPRESS is specific to IMAP.

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.

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

3. The COMPRESS Command

Arguments: Name of compression mechanism: "DEFLATE".

Responses: None

Result: OK The server will compress its responses and expects the client to compress its commands.
NO The server doesn't support the requested mechanism.
BAD Command unknown, invalid argument, or COMPRESS already active.

The COMPRESS command instructs the server to use the named compression mechanism ("DEFLATE" is the only one defined) for all commands and/or responses after COMPRESS.

The client MUST NOT send any commands until it has seen the result of COMPRESS. If the response was OK, the client MUST compress starting with the first command after COMPRESS, and the server MUST compress starting with the first response after the OK.

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.

If both COMPRESS and STARTTLS and/or a [SASL] security layer 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.

The following example illustrates how commands and responses are compressed during a simple login sequence:


```
S: * OK [CAPABILITY IMAP4REV1 STARTTLS COMPRESS=DEFLATE]
C: a starttls
S: a OK TLS active

From this point on, everything is encrypted.

C: b compress deflate
S: b OK DEFLATE active

From this point on, everything is compressed before being
encrypted.

C: c login arnt tnra
S: c OK Logged in as arnt
```

4. Compression Efficiency

This section is informative, not normative.

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 repetition works efficiently. The APPEND command is an exception; clients which send many APPEND commands may want to send flushes in the same way that servers do.

Downstream has the unusual property that several kinds of data are

sent, confusing all dictionary-based compression algorithms.

One 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.

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

A 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 its dictionary from IMAP to the attachment's format, and then back. A few file formats aren't compressible at all using deflate, e.g. .gz, .zip and .jpg files.

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 6-bit bytes, hiding most of the 8-bit file format from the compressor.

When using the zlib library (see [DEFLATE]), the functions deflateInit2(), deflate(), inflateInit2() and inflate() suffice to implement this extension. deflateParams() can be used to improve compression rate and resource use.

A client can improve downstream compression by implementing [BINARY] and using FETCH BINARY instead of FETCH BODY. In the author's experience, the improvement ranges from 5% to 40% depending on the

attachment being downloaded.

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). Small literals are best left alone.

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 to 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.

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.

5. 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

capability  =/ "COMPRESS=" algorithm
              ;; multiple COMPRESS capabilities allowed
```

```
   algorithm    = "DEFLATE"
```

Note that due the syntax of capability means, future algorithm names must be atoms.

6. Security Considerations

As for [TLSCOMP] RFC 3749.

7. IANA Considerations

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

The IANA is requested to maintain one new registry: IMAP Compression Algorithms. The registry's purpose is to register compression algorithms that may be used with this extension. New IMAP algorithms MUST be defined in a standards track or IESG approved experimental RFC. New IMAP compression algorithms MUST include the following information as part of their definition:

```
     algorithm identifier
     standard commands affected
     specification reference
     discussion
```

This registry is available at URL [RFC-EDITOR NOTE: please insert URL of registry]

One IMAP compression algorithm is defined in this document, with the following registration definition:


```
     algorithm identifier: DEFLATE
     standard commands affected: none
     specification reference: RFC 1951 and XXXX
     discussion: see RFC XXXX
```

[RFC-EDITOR NOTE: change XXXX to this RFC number]

8. Acknowledgements

Eric Burger, Dave Cridland, Tony Finch, Ned Freed, Philip Guenther, Randall Gellens, Tony Hansen, Alexey Melnikov, Lyndon Nerenberg and Zoltan Ordogh have all helped with this document.

The author would also like to thank various people in the rooms at meetings, whose help is real, but not reflected in the author's mailbox.

9. References

9.1. Normative References

[ABNF] Crocker, Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, Brandenburg Internetworking, Demon Internet Ltd, October 2005.

[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.

9.2. Informative References

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

[SASL] A. Melnikov, K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, Isode Limited, June 2006

[BINARY] Nerenberg, "IMAP4 Binary Content Extension", Orthanc

Systems, April 2003.

10. Author's Address

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

Fax: +49 89 4502 9758

Email: arnt@oryx.com

11. Open Issues

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

Intellectual Property Statement

Copyright Statement

Disclaimer of Validity

Acknowledgment

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