Network Working Group
Request for Comments: 4978
Category: Standards Track

A. Gulbrandsen
Oryx Mail Systems GmbH

August 2007

The IMAP COMPRESS Extension

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

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

Table of Contents

1. 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 compression (see [RFC1962]) and modem-based compression (see [MNP] and [V42BIS]), COMPRESS offers much better compression efficiency. COMPRESS can be used together with Transport Security Layer (TLS) [RFC4346], Simple Authentication and Security layer (SASL) encryption, Virtual Private Networks (VPNs), etc. Compared to TLS compression [RFC3749], COMPRESS has the following (dis)advantages:

- COMPRESS can be implemented easily both by IMAP servers and clients.

- IMAP COMPRESS benefits from an intimate knowledge of the IMAP protocol's state machine, allowing for dynamic and aggressive optimization of the underlying compression algorithm's parameters.

- When the TLS layer implements compression, any protocol using that layer can transparently benefit from that compression (e.g., SMTP and IMAP). 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 (defined in [RFC1951]) is standard, widely available and fairly efficient, so it is the only algorithm defined by this document.

In order to increase interoperation, IMAP servers that advertise this extension SHOULD also advertise the TLS DEFLATE compression mechanism as defined in [RFC3749]. IMAP clients MAY use either COMPRESS or TLS compression, however, if the client and server support both, it is RECOMMENDED that the client choose TLS compression.

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

2. 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 [RFC2119].

Formal syntax is defined by [RFC4234] as modified by [RFC3501].

In the examples, "C:" and "S:" indicate lines sent by the client and server respectively. "[...]" denotes elision.

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 Compression is already active via another layer.
BAD Command unknown, invalid or unknown 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 further 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. If the server response was BAD or NO, the client MUST NOT turn on compression.

If the server responds NO because it knows that the same mechanism is active already (e.g., because TLS has negotiated the same mechanism), it MUST send COMPRESSIONACTIVE as resp-text-code (see [RFC3501], Section 7.1), and the resp-text SHOULD say which layer compresses.

If the server issues an OK response, the server MUST compress starting immediately after the CRLF which ends the tagged OK response. (Responses issued by the server before the OK response will, of course, still be uncompressed.) If the server issues a BAD or NO response, the server MUST NOT turn on compression.

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.

When COMPRESS is combined with TLS (see [RFC4346]) or SASL (see [RFC4422]) security layers, the sending order of the three extensions MUST be first COMPRESS, then SASL, and finally TLS. That is, before data is transmitted it is first compressed. Second, if a SASL security layer has been negotiated, the compressed data is then signed and/or encrypted accordingly. Third, if a TLS security layer has been negotiated, the data from the previous step is signed and/or

encrypted accordingly. When receiving data, the processing order MUST be reversed. This ensures that before sending, data is compressed before it is encrypted, 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 login arnt tnra
S: b OK Logged in as arnt
C: c compress deflate
S: d OK DEFLATE active

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

The following example demonstrates how a server may refuse to compress twice:

```
S: * OK [CAPABILITY IMAP4REV1 STARTTLS COMPRESS=DEFLATE]
[...]
C: c compress deflate
S: c NO [COMPRESSIONACTIVE] DEFLATE active via TLS
```

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 algorithm that can exploit repetition works efficiently. The APPEND command is an exception; clients that send many APPEND commands may want to surround large literals with flushes in the same way as is recommended for servers later in this section.

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

A third type 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 [RFC1951]), the functions deflateInit2(), deflate(), inflateInit2(), and inflate() suffice to implement this extension. The windowBits value must be in the range -8 to -15, or else deflateInit2() uses the wrong format. deflateParams() can be used to improve compression rate and resource use. The Z_FULL_FLUSH argument to deflate() can be used to clear the dictionary (the receiving peer does not need to do anything).

A client can improve downstream compression by implementing BINARY (defined in [RFC3516]) 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 possible boundary is 5k.

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, to avoid trying to compress incompressible attachments. A very simple strategy is to change the level to 0 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. More complex strategies are possible.

5. Formal Syntax

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) notation as specified in [RFC4234]. This syntax augments the grammar specified in [RFC3501]. [RFC4234] defines SP and [RFC3501] defines command-auth, capability, and resp-text-code.

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-auth =/ compress

compress    = "COMPRESS" SP algorithm

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

algorithm   = "DEFLATE"

resp-text-code =/ "COMPRESSIONACTIVE"
```

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

6. Security Considerations

As for TLS compression [RFC3749].

7. IANA Considerations

The IANA has added COMPRESS=DEFLATE to the list of IMAP capabilities.

8. Acknowledgements

Eric Burger, Dave Cridland, Tony Finch, Ned Freed, Philip Guenther, Randall Gellens, Tony Hansen, Cullen Jennings, Stephane Maes, 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

[RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.

[RFC4234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.

9.2. Informative References

[RFC1962] Rand, D., "The PPP Compression Control Protocol (CCP)", RFC 1962, June 1996.

[RFC3516] Nerenberg, L., "IMAP4 Binary Content Extension", RFC 3516, April 2003.

[RFC3749] Hollenbeck, S., "Transport Layer Security Protocol Compression Methods", RFC 3749, May 2004.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

[RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.

[V42BIS] ITU, "V.42bis: Data compression procedures for data circuit-terminating equipment (DCE) using error correction procedures", http://www.itu.int/rec/T-REC-V.42bis, January 1990.

[MNP] Gilbert Held, "The Complete Modem Reference", Second Edition, Wiley Professional Computing, ISBN 0-471-00852-4, May 1994.

Author's Address

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

Fax: +49 89 4502 9758
EMail: arnt@oryx.com

Full Copyright Statement

Copyright (C) The IETF Trust (2007).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.