Transport Layer Security (TLS) Protocol Compression Using Lempel-Ziv-Stac (LZS)

Status of this Memo

Copyright Notice

Abstract

   The Transport Layer Security (TLS) protocol (RFC 2246) includes
   features to negotiate selection of a lossless data compression method
   as part of the TLS Handshake Protocol and then to apply the algorithm
   associated with the selected method as part of the TLS Record
   Protocol. TLS defines one standard compression method, which
   specifies that data exchanged via the record protocol will not be
   compressed. This document describes an additional compression method
   associated with the Lempel-Ziv-Stac (LZS) lossless data compression
   algorithm for use with TLS. This document also defines the
   application of the LZS algorithm to the TLS Record Protocol.

Table of Contents

[1](1). Introduction

[1.1](1.1). General

   The Transport Layer Security (TLS) protocol ([RFC 2246](RFC 2246), [[2](2)]) includes
   features to negotiate selection of a lossless data compression method
   as part of the TLS Handshake Protocol and then to apply the algorithm
   associated with the selected method as part of the TLS Record
   Protocol. TLS defines one standard compression method,
   CompressionMethod.null, which specifies that data exchanged via the
   record protocol will not be compressed. Although this single
   compression method helps ensure that TLS implementations are
   interoperable, the lack of additional standard compression methods
   has limited the ability to develop interoperative implementations
   that include data compression.

   TLS is used extensively to secure client-server connections on the
   World Wide Web. Although these connections can often be
   characterized as short-lived and exchanging relatively small amounts
   of data, TLS is also being used in environments where connections can
   be long-lived and the amount of data exchanged can extend into
   thousands or millions of octets. For example, TLS is now
   increasingly being used as an alternative Virtual Private Network
   (VPN) connection. Compression services have long been associated
   with IPSec and PPTP VPN connections, so extending compression
   services to TLS VPN connections preserves the user experience for any
   VPN connection. Compression within TLS is one way to help reduce the
   bandwidth and latency requirements associated with exchanging large
   amounts of data while preserving the security services provided by
   TLS.

   This document describes an additional compression method associated
   with a lossless data compression algorithm for use with TLS. This
   document specifies the application of Lempel-Ziv-Stac (LZS)
   compression, a lossless compression algorithm, to TLS record
   payloads. This specification also assumes a thorough understanding
   of the TLS protocol [2].

1.2. Specification of Requirements

   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 BCP 14, RFC 2119 [1].

2. Compression Methods

   As described in section 6 of RFC 2246 [2], TLS is a stateful
   protocol. Compression methods used with TLS can be either stateful
   (the compressor maintains its state through all compressed records)
   or stateless (the compressor compresses each record independently),
   but there seems to be little known benefit in using a stateless
   compression method within TLS. The LZS compression method described
   in this document is stateful.

   Compression algorithms can occasionally expand, rather than compress,
   input data. The worst-case expansion factor of the LZS compression
   method is only 12.5%. Thus, TLS records of 15K bytes can never
   exceed the expansion limits described in section 6.2.2 of RFC 2246
   [2]. If TLS records of 16K bytes expand to an amount greater than
   17K bytes, then the uncompressed version of the TLS record must be
   transmitted, as described below.

2.1. **LZS CompressionMethod**

   The LZS CompressionMethod is a 16-bit index and is negotiated as
   described in RFC 2246 [2] and RFC 3749 [3]. The LZS
   CompressionMethod is stored in the TLS Record Layer connection state
   as described in RFC 2246 [2].

   IANA has assigned 64 as compression method identifier for applying
   LZS compression to TLS record payloads.

2.2. **Security Issues with Compression Histories**

   Sharing compression histories between or among more than one TLS
   session may potentially cause information leakage between the TLS
   sessions, as pathological compressed data can potentially reference
   data prior to the beginning of the current record. LZS
   implementations guard against this situation. However, to avoid this
   potential threat, implementations supporting TLS compression MUST use
   separate compression histories for each TLS session. This is not a
   limitation of LZS compression but is an artifact for any compression
   algorithm.

   Furthermore, the LZS compression history (as well as any compression
   history) contains plaintext. Specifically, the LZS history contains
   the last 2K bytes of plaintext of the TLS session. Thus, when the
   TLS session terminates, the implementation SHOULD treat the history
   as it does any plaintext (e.g., free memory, overwrite contents).

3. **LZS Compression**

3.1. **Background of LZS Compression**

   Starting with a sliding window compression history, similar to LZ1
   [8], a new, enhanced compression algorithm identified as LZS was
   developed. The LZS algorithm is a general-purpose lossless
   compression algorithm for use with a wide variety of data types. Its
   encoding method is very efficient, providing compression for strings
   as short as two octets in length.

   The LZS algorithm uses a sliding window of 2,048 bytes. During
   compression, redundant sequences of data are replaced with tokens
   that represent those sequences. During decompression, the original
   sequences are substituted for the tokens in such a way that the
   original data is exactly recovered. LZS differs from lossy
   compression algorithms, such as those often used for video
   compression, that do not exactly reproduce the original data. The
   details of LZS compression can be found in section 3.5 below.

3.2. **LZS Compression History and Record Processing**

   This standard specifies "stateful" compression -- that is,
   maintaining the compression history between records within a
   particular TLS compression session. Within each separate compression
   history, the LZS CompressionMethod can maintain compression history
   information when compressing and decompressing record payloads.
   Stateful compression provides a higher compression ratio to be
   achieved on the data stream, as compared to stateless compression
   (resetting the compression history between every record),
   particularly for small records.

   Stateful compression requires both a reliable link and sequenced
   record delivery to ensure that all records can be decompressed in the
   same order they were compressed. Since TLS and lower-layer protocols
   provide reliable, sequenced record delivery, compression history
   information MAY be maintained and exploited when the LZS
   CompressionMethod is used.

   Furthermore, there MUST be a separate LZS compression history
   associated with each open TLS session. This not only provides
   enhanced security (no potential information leakage between sessions
   via a shared compression history), but also enables superior
   compression ratio (bit bandwidth on the connection) across all open
   TLS sessions with compression. A shared history would require
   resetting the compression (and decompression) history when switching
   between TLS sessions, and a single history implementation would
   require resetting the compression (and decompression) history between
   each record.

   The sender MUST reset the compression history prior to compressing
   the first TLS record of a TLS session after TLS handshake completes.
   It is advantageous for the sender to maintain the compression history
   for all subsequent records processed during the TLS session. This
   results in the greatest compression ratio for a given data set. In
   either case, this compression history MUST NOT be used for any other
   open TLS session, to ensure privacy between TLS sessions.

   The sender MUST "flush" the compressor each time it transmits a
   compressed record. Flushing means that all data going into the
   compressor is included in the output, i.e., no data is retained in
   the hope of achieving better compression. Flushing ensures that each
   compressed record payload can be decompressed completely. Flushing is
   necessary to prevent a record's data from spilling over into a later
   record. This is important for synchronizing compressed data with the
   authenticated and encrypted data in a TLS record. Flushing is
   handled automatically in most LZS implementations.

   When the TLS session terminates, the implementation SHOULD dispose of
   the memory resources associated with the related TLS compression
   history. That is, the compression history SHOULD be handled as the
   TLS key material is handled.

   The LZS CompressionMethod also features "decompressing" uncompressed
   data in order to maintain the history if the "compressed" data
   actually expanded. The LZS CompressionMethod record format
   facilitates identifying whether records contain compressed or
   uncompressed data. The LZS decoding process accommodates
   decompressing either compressed or uncompressed data.

3.3. LZS Compressed Record Format

   Prior to compression, the uncompressed data (TLSPlaintext.fragment)
   is composed of a plaintext TLS record. After compression, the
   compressed data (TLSCompressed.fragment) is composed of an 8-bit
   TLSComp header followed by the compressed (or uncompressed) data.

3.4. TLSComp Header Format

   The one-octet header has the following structure:

```
    0
    0 1 2 3 4 5 6 7
    +-+-+-+-+-+-+-+-+
    |     Flags     |
    |               |
    +-+-+-+-+-+-+-+-+
```

3.4.1. Flags

   The format of the 8-bit Flags TLSComp field is as follows:

```
       0     1     2     3     4     5     6     7
    +-----+-----+-----+-----+-----+-----+-----+-----+
    | Res | Res | Res | Res | Res | Res | RST | C/U |
    +-----+-----+-----+-----+-----+-----+-----+-----+
```

   Res-Reserved

      Reserved for future use. MUST be set to zero. MUST be ignored by
      the receiving node.

RST-Reset Compression History

   The RST bit is used to inform the decompressing peer that the
   compression history in this TLS session was reset prior to the
   data contained in this TLS record being compressed. When the RST
   bit is set to "1", a compression history reset is performed; when
   RST is set to "0", a compression history reset is not performed.

   This bit MUST be set to a value of "1" for the first compressed
   TLS transmitted record of a TLS session. This bit may also be
   used by the transmitter for other exception cases when the
   compression history must be reset.

C/U-Compressed/Uncompressed Bit

   The C/U indicates whether the data field contains compressed or
   uncompressed data. A value of 1 indicates compressed data (often
   referred to as a compressed record), and a value of 0 indicates
   uncompressed data (or an uncompressed record).

3.5. **LZS Compression Encoding Format**

The LZS compression method, encoding format, and application examples
are described in RFC 1967 [6], RFC 1974 [5], and RFC 2395 [4].

Some implementations of LZS allow the sending compressor to select
from among several options to provide varying compression ratios,
processing speeds, and memory requirements. Other implementations of
LZS provide optimal compression ratio at byte-per-clock speeds.

The receiving LZS decompressor automatically adjusts to the settings
selected by the sender. Also, receiving LZS decompressors will
update the decompression history with uncompressed data. This
facilitates never obtaining less than a 1:1 compression ratio in the
session and never transmitting with expanded data.

The input to the payload compression algorithm is TLSPlaintext data
destined to an active TLS session with compression negotiated. The
output of the algorithm is a new (and hopefully smaller)
TLSCompressed record. The output payload contains the input
payload's data in either compressed or uncompressed format. The
input and output payloads are each an integral number of bytes in
length.

The output payload is always prepended with the TLSComp header. If
the uncompressed form is used, the output payload is identical to the
input payload, and the TLSComp header reflects uncompressed data.

If the compressed form is used, encoded as defined in ANSI X3.241
[7], and the TLSComp header reflects compressed data. The LZS
encoded format is repeated here for informational purposes ONLY.

```
<Compressed Stream> := [<Compressed String>*] <End Marker>
<Compressed String> := 0 <Raw Byte> | 1 <Compressed Bytes>

<Raw Byte> := <b><b><b><b><b><b><b><b>            (8-bit byte)
<Compressed Bytes> := <Offset> <Length>

<Offset> := 1 <b><b><b><b><b><b><b> |             (7-bit offset)
            0 <b><b><b><b><b><b><b><b><b><b><b> (11-bit offset)
<End Marker> := 110000000
<b> := 1 | 0

<Length> :=
00        = 2    1111 0110      = 14
01        = 3    1111 0111      = 15
10        = 4    1111 1000      = 16
1100      = 5    1111 1001      = 17
1101      = 6    1111 1010      = 18
1110      = 7    1111 1011      = 19
1111 0000 = 8    1111 1100      = 20
1111 0001 = 9    1111 1101      = 21
1111 0010 = 10   1111 1110      = 22
1111 0011 = 11   1111 1111 0000 = 23
1111 0100 = 12   1111 1111 0001 = 24
1111 0101 = 13      ...
```

3.6. Padding

A datagram payload compressed with LZS always ends with the last
compressed data byte (also known as the <end marker>), which is used
to disambiguate padding. This allows trailing bits, as well as
bytes, to be considered padding.

The size of a compressed payload MUST be in whole octet units.

4. Sending Compressed Datagrams

All TLS records processed with a TLS session state that includes LZS
compression are processed as follows. The reliable and efficient
transport of LZS compressed records in the TLS session depends on the
following processes.

4.1. Transmitter Process

   The compression operation results in either compressed or
   uncompressed data. When a TLS record is received, it is assigned to
   a particular TLS context that includes the LZS compression history
   buffer. It is processed according to ANSI X3.241-1994 to form
   compressed data or used as is to form uncompressed data. For the
   first record of the session, or for exception conditions, the
   compression history MUST be cleared. In performing the compression
   operation, the compression history MUST be updated when either a
   compressed record or an uncompressed record is produced.
   Uncompressed TLS records MAY be sent at any time. Uncompressed TLS
   records MUST be sent if compression causes enough expansion to make
   the data compression TLS record size exceed the MTU defined in
   section 6.2.2 in RFC 2246. The output of the compression operation
   is placed in the fragment field of the TLSCompressed structure
   (TLSCompressed.fragment).

   The TLSComp header byte is located just prior to the first byte of
   the compressed TLS record in TLSCompressed.fragment. The C/U bit in
   the TLSComp header is set according to whether the data field
   contains compressed or uncompressed data. The RST bit in the TLSComp
   header is set to "1" if the compression history was reset prior to
   compressing the TLSplaintext.fragment that is composed of a
   TLSCompressed.fragment. Uncompressed data MUST be transmitted (and
   the C/U bit set to 0) if the "compressed" (expanded) data exceeded
   17K bytes.

4.2. Receiver Process

   Prior to decompressing the first compressed TLS record in the TLS
   session, the receiver MUST reset the decompression history.
   Subsequent records are decompressed in the order received. The
   receiver decompresses the Payload Data field according to the
   encoding specified in section 3.5 above.

   If the received datagram is not compressed, the receiver does not
   need to perform decompression processing, and the Payload Data field
   of the datagram is ready for processing by the next protocol layer.

   After a TLS record is received from the peer and decrypted, the RST
   and C/U bits MUST be checked.

   If the C/U bit is set to "1", the resulting compressed data block
   MUST be decompressed according to section 3.5 above.

   If the C/U bit is set to "0", the specified decompression history
   MUST be updated with the received uncompressed data.

If the RST bit is set to "1", the receiving decompression history MAY
be reset to an initial state prior to decompressing the TLS record.
(However, due to the characteristics of the Hifn LZS algorithm, a
decompression history reset is not required). After reset, any
compressed or uncompressed data contained in the record is processed.

4.3. Anti-expansion Mechanism

During compression, there are two workable options for handling
records that expand:

1) Send the expanded data (as long as TLSCompressed.length is 17K or
   less) and maintain the history, thus allowing loss of current
   bandwidth but preserving future bandwidth on the link.

2) Send the uncompressed data and do not clear the compression
   history; the decompressor will update its history, thus conserving
   the current bandwidth and future bandwidth on the link.

The second option is the preferred option and SHOULD be implemented.

There is a third option:

3) Send the uncompressed data and clear the history, thus conserving
   current bandwidth but allowing possible loss of future bandwidth
   on the link.

This option SHOULD NOT be implemented.

5. Internationalization Considerations

The compression method identifiers specified in this document are
machine-readable numbers. As such, issues of human
internationalization and localization are not introduced.

6. IANA Considerations

Section 2 of RFC 3749 [3] describes a registry of compression method
identifiers to be maintained by the IANA and to be assigned within
three zones.

IANA has assigned an identifier for the LZS compression method from
the RFC 2434 Specification Required IANA pool, as described in
sections 2 and 5 of RFC 3749 [3].

The IANA-assigned compression method identifier for LZS is 64 decimal
(0x40).

7. Security Considerations

   This document does not introduce any topics that alter the threat
   model addressed by TLS. The security considerations described
   throughout RFC 2246 [2] apply here as well.

   However, combining compression with encryption can sometimes reveal
   information that would not have been revealed without compression.
   Data that is the same length before compression might be a different
   length after compression, so adversaries that observe the length of
   the compressed data might be able to derive information about the
   corresponding uncompressed data. Some symmetric encryption
   ciphersuites do not hide the length of symmetrically encrypted data
   at all. Others hide it to some extent but not fully. For example,
   ciphersuites that use stream cipher encryption without padding do not
   hide length at all; ciphersuites that use Cipher Block Chaining (CBC)
   encryption with padding provide some length hiding, depending on how
   the amount of padding is chosen. Use of TLS compression SHOULD take
   into account that the length of compressed data may leak more
   information than the length of the original uncompressed data.

   Another security issue to be aware of is that the LZS compression
   history contains plaintext. In order to prevent any kind of
   information leakage outside the system, when a TLS session with
   compression terminates, the implementation SHOULD treat the
   compression history as it does plaintext -- that is, care should be
   taken not to reveal the compression history in any form or to use it
   again. This is described in sections 2.2 and 3.2 above.

   This information leakage concept can be extended to the situation of
   sharing a single compression history across more than one TLS
   session, as addressed in section 2.2 above.

   Other security issues are discussed in RFC 3749 [3].

8. Acknowledgements

   The concepts described in this document were derived from RFC 1967
   [6], RFC 1974 [5], RFC 2395 [4], and RFC 3749 [3]. The author
   acknowledges the contributions of Scott Hollenbeck, Douglas Whiting,
   and Russell Dietz, and help from Steve Bellovin, Russ Housley, and
   Eric Rescorla.

9. References

9.1. Normative References

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

[2]   Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC
      2246, January 1999.

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

9.2. Informative References

[4]   Friend, R. and R. Monsour, "IP Payload Compression Using LZS",
      RFC 2395, December 1998.

[5]   Friend, R. and W. Simpson, "PPP Stac LZS Compression Protocol",
      RFC 1974, August 1996.

[6]   Schneider, K. and R. Friend, "PPP LZS-DCP Compression Protocol
      (LZS-DCP)", RFC 1967, August 1996.

[7]   American National Standards Institute, Inc., "Data Compression
      Method for Information Systems," ANSI X3.241-1994, August 1994.

[8]   Lempel, A. and J. Ziv, "A Universal Algorithm for Sequential
      Data Compression", IEEE Transactions On Information Theory, Vol.
      IT-23, No. 3, September 1977.

Author's Address

   Robert Friend
   Hifn
   5973 Avenida Encinas
   Carlsbad, CA 92008
   US

   EMail: rfriend@hifn.com

Full Copyright Statement

    This document is subject to the rights, licenses and restrictions
    contained in BCP 78, and at www.rfc-editor.org, 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 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

    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 ISOC's procedures with respect to rights in ISOC 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.

Acknowledgement