Network Working Group   M. Tuexen
Internet-Draft   Univ. of Applied Sciences Muenster
Expires: August 24, 2005   R. Stewart
                                                            P. Lei
                                                Cisco Systems, Inc.
                                                       E. Rescorla
                                                         RTFM, Inc.
                                                 February 20, 2005

Authenticated Chunks for Stream Control Transmission Protocol (SCTP)
draft-tuexen-sctp-auth-chunk-03.txt

Status of this Memo

Copyright Notice

Abstract

   This document describes a new chunk type, several parameters and
   procedures for SCTP. This new chunk type can be used to authenticate

   SCTP chunks by using a shared key between the sender and receiver.
   The new parameters are used to establish the shared key.

Table of Contents

1. Introduction

SCTP uses 32 bit verification tags to protect itself against blind attackers. These values are not changed during the lifetime of an SCTP association.

Looking at new SCTP extensions there is the need to have a method of proving that an SCTP chunk(s) was really sent by the original peer that started the association and not by a malicious attacker.

Using TLS as defined in RFC3436 [8] does not help here because it only secures SCTP user data.

Therefore an SCTP extension is presented in this document which allows an SCTP sender to sign chunks using a shared key between the sender and receiver. The receiver can then verify, that the chunks are sent from the sender and not from a malicious attacker.

This extension also provides a mechanism for deriving a shared key for each association. This association shared key is derived from a endpoint pair shared key, which is preconfigured and might be empty.

2. Conventions

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, NOT RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in RFC2119 [4].

3. New Parameter Types

This section defines the new parameter types that will be used to negotiate the authentication during association setup. Figure 1 illustrates the new parameter types.

```
Parameter Type  Parameter Name
---------------------------------------------------------------
0x8002          Random Parameter (RANDOM)
0x8003          Chunk List Parameter (CHUNKS)
```

                         Figure 1

It should be noted that the parameter format requires the receiver to ignore the parameter and continue processing if it is not understood. This is accomplished as described in RFC2960 [7] section 3.2.1. by the use of the upper bit of the parameter type.

3.1 Random Parameter (RANDOM)

   This parameter is used to carry an arbitrary length random number.


```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Parameter Type = 0x8002   |       Parameter Length       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
\                      Random Number                            /
/                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

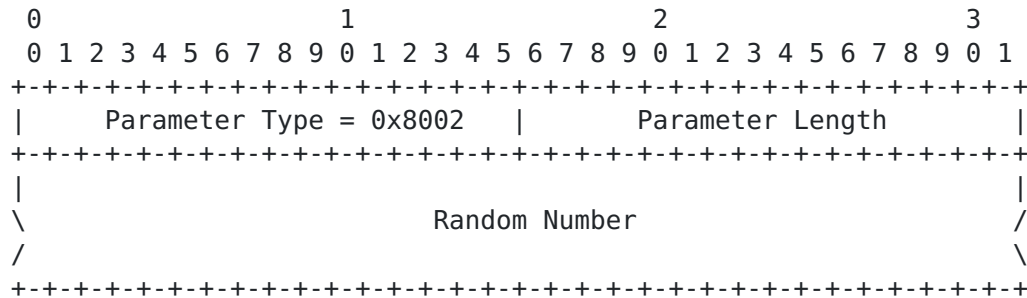   Figure 2

   Parameter Type: 2 bytes (unsigned integer) This value MUST be set to
      0x8002.

   Parameter Length: 2 bytes (unsigned integer) This value is the length
      of the Random Number plus 4.

   Random Number: n bytes (unsigned integer) This value represents an
      arbitrary Random Number in network byte order.

   The RANDOM parameter MUST be included once in the INIT or INIT-ACK
   chunk if the sender wants to send or receive authenticated chunks.

3.2 Chunk List Parameter (CHUNKS)

   This parameter is used to specify which chunk types are required to
   be sent authenticated by the peer.


```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Parameter Type = 0x8003   |       Parameter Length       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Chunk Type 1  | Chunk Type 2   | Chunk Type 4  | Chunk Type 4  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                                                               /
\                              ...                              \
/                                                               /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Chunk Type n  |                  Padding                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
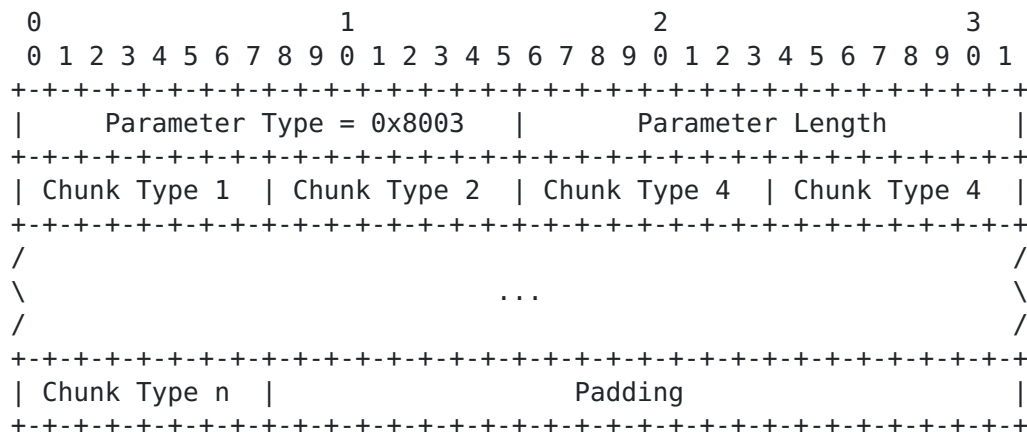
Figure 3

   Parameter Type: 2 bytes (unsigned integer) This value MUST be set to
      0x8003.

   Parameter Length: 2 bytes (unsigned integer) This value is the number
      of listed Chunk Types plus 4.

   Chunk Type n: 1 byte (unsigned integer) Each Chunk Type listed is
      required to be authenticated when sent by the peer.

   The CHUNKS parameter MUST be included once in the INIT or INIT-ACK
   chunk if the sender wants to receive authenticated chunks. Its
   maximum length is 260 bytes.

   The chunk types for INIT, INIT-ACK, COOKIE-ECHO, COOKIE-ACK,
   SHUTDOWN-COMPLETE and AUTH chunks MUST not be listed in the CHUNKS
   parameter. However, if a CHUNKS parameter is received then the types
   for INIT, INIT-ACK, COOKIE-ECHO, COOKIE-ACK, SHUTDOWN-COMPLETE and
   AUTH chunks MUST be ignored.

4. New Chunk Type

   This section defines the new chunk type that will be used to
   authenticate chunks. Figure 4 illustrates the new chunk type.

   Chunk Type   Chunk Name

   0x10   Authentication Chunk (AUTH)

                              Figure 4

   It should be noted that the AUTH-chunk format requires the receiver
   to ignore the chunk if it is not understood and silently discard all
   chunks that follow. This is accomplished as described in RFC2960 [7]
   section 3.2. by the use of the upper bit of the chunk type.

4.1 Authentication Chunk (AUTH)

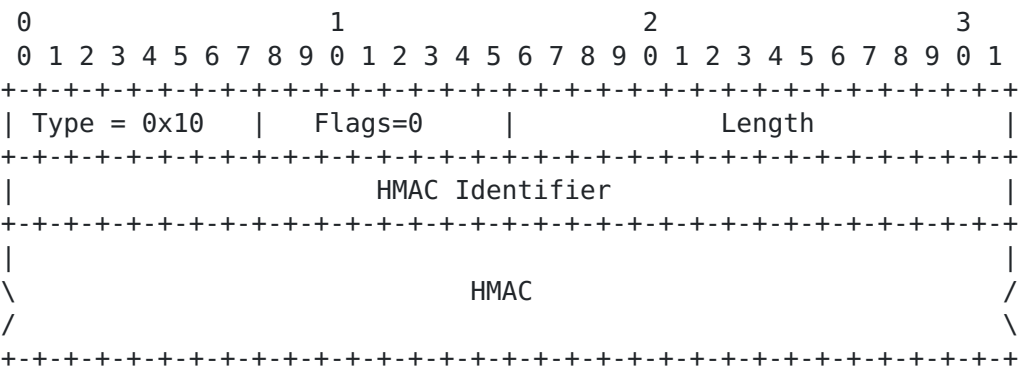   This chunk is used to hold the result of the HMAC calculation.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Type = 0x10   |  Flags=0      |             Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       HMAC Identifier                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
\                           HMAC                                /
/                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

                          Figure 5

   Type: 1 byte (unsigned integer) This value MUST be set to 0x83 for
      all AUTH-chunks.

   Flags: 1 byte (unsigned integer) Set to zero on transmit and ignored
      on receipt.

   Length: 2 bytes (unsigned integer) This value holds the length of the
      HMAC plus 8.

   HMAC Identifier: 4 bytes (unsigned integer) This value describes
      which message digest is being used. The following Figure 6 shows
      the currently defined values.

```
HMAC Identifier  Message Digest Algorithm
-----------------------------------------------------------------
0                MD-5 defined in [1]
1                SHA-1 defined in [10]
```

                          Figure 6

   HMAC: n bytes (unsigned integer) This hold the result of the HMAC
      calculation.

   The control chunk AUTH can appear at most once in an SCTP packet.
   All control and data chunks which are placed after the AUTH chunk in
   the packet are sent in an authenticated way. Those chunks placed in
   a packet before the AUTH chunk are not authenticated.

5. Procedures

5.1 Establishment of an association shared key

   An SCTP endpoint willing to receive or send authenticated chunks has
   to send one RANDOM parameter in its INIT or INIT-ACK chunk. The

RANDOM parameter MUST contain a 32 byte random number. This random
number is handled like the verification tag in case of INIT
collisions. Therefore each endpoint knows its own random number and
the peers random number after the association has been established.

An SCTP endpoint has a list of chunks it only accepts if they are
received in an authenticated way. This list is included in the INIT
and INIT-ACK and MAY be omitted if it is empty. Since this list is
for an endpoint there is no problem in case of INIT collision.

Both enpoints of an association have an endpoint pair shared key
which is a byte vector and preconfigured or established by another
mechanism. If it is not preconfigured or established by another
mechanism it is set to the empty byte vector.

From this endpoint pair shared key the association shared key is
computed by concatenating the endpoint pair shared key with the
random numbers exchanged in the INIT and INIT-ACK. This is performed
by selecting the smaller random number and concatenating it to the
endpoint pair shared key. Then concatenating the larger of the
random numbers to that. If both random numbers are equal they may be
concatenated to the endpoint pair key in any order. The
concatenation is performed on byte vectors representing all numbers
in network byte order. The result is the association shared key.

5.2 Sending authenticated chunks

Chunks can only be authenticated when the SCTP association is in the
ESTABLISHED state. Both endpoints MUST send all those chunks
authenticated where this has been requested by the peer. The other
chunks MAY be sent authenticated.

To send chunks in an authenticated way, the sender has to include
these chunks after an AUTH chunk. This means that a sender MUST
bundle chunks in order to authenticate them.

The sender MUST calculate the MAC using the hash function H as
described by the MAC Identifier and the shared association key K.
The 'data' used for the computation is the AUTH-chunk as given by
Figure 7 and all chunks that are placed after the AUTH chunk in the
SCTP packet. RFC2104 [3] can be used as a guideline for generating
the MAC.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Type = 0x10   |   Flags=0     |          Chunk Length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        HMAC Identifier                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
\                              0                                /
/                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7

Please note that all fields are in network byte order.

The sender fills the HMAC then into the HMAC field and sends the
packet.

5.3 Receiving authenticated chunks

The receiver has a list of chunk types which it expects to be
received only after an AUTH-chunk. This list has been sent to the
peer during the association setup. It MUST silently discard these
chunks if they are not placed after an AUTH chunk in the packet.

The receiver MUST use the HMAC algorithm indicated in the HMAC
Identifier field. If this algorithm is not known the AUTH chunk and
all chunks after it MUST silently be discarded.

The receiver now performs the same calculation as described for the
sender based on Figure 7. If the result of the calculation is the
same as given in the HMAC field, all chunks following the AUTH chunk
are processed. If the field does not match the result of the
calculation all these chunks MUST be silently discarded.

6. Examples

This section gives examples of message exchanges for association
setup.

The simplest way of using the extension described in this document is
given by the following message exchange.

```
---------------- INIT[RANDOM; CHUNKS] --------------->
<------------ INIT-ACK[RANDOM; CHUNKS] --------------
------------------- COOKIE-ECHO -------------------->
<------------------- COOKIE-ACK ---------------------
```

Please note that the CHUNKS parameter is optional in the INIT and

INIT-ACK.

If the server wants to receive DATA chunks in an authenticated way,
the following message exchange is possible:

```
---------------- INIT[RANDOM; CHUNKS] --------------->
<------------- INIT-ACK[RANDOM; CHUNKS] --------------
--------------- COOKIE-ECHO; AUTH; DATA ------------->
<---------------- COOKIE-ACK; SACK ------------------
```

Please note that if the endpoint pair shared key depends on the
client and the server and that it is only known by the upper layer
this message exchange requires an upper layer intervention between
the processing of the COOKIE-ECHO chunk (COMMUNICATION-UP
notification followed by the presentation of the endpoint pair shared
key by the upper layer to the SCTP stack) and the processing of the
AUTH and DATA chunk. If this intervention is not possible due to
limitations of the API the server might discard the AUTH and DATA
chunk making a retransmission of the DATA chunk necessary. If the
same endpoint pair shared key is used for multiple endpoints and does
not depend on the client this intervention might not be necessary.

7. IANA Considerations

A chunk type for the AUTH chunk has to be assigned by IANA. It is
suggested to use the value given above.

Parameter types have to be assigned for the RANDOM and CHUNKS
parameter by IANA. It is suggested to use the values given above.

8. Security Considerations

This section is still incomplete.

If no endpoint pair shared key is used an attacker which captures the
association setup message exchange can later insert arbitrary packets
in an authenticated way. However, if the attacker did not capture
this initial message exchange he can not successfully inject chunks
which are required to be authenticated.

If an enpoint pair shared key is used even a true man in the middle
can not inject chunks which are required to be authenticated even if
he intercepts the initial message exchange.

Because SCTP has already a mechanism built-in that handles the
reception of duplicated chunks the presented solution makes use of
this functionality and does not provide a method to avoid replay
attacks by itself. Of course, this only works within each SCTP

association. Therefore a separate shared key is used for each SCTP
association to handle replay attacks covering multiple SCTP
associations.

9. References

9.1 Normative References

[1]   Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April
        1992.

[2]   Bradner, S., "The Internet Standards Process -- Revision 3",
        BCP 9, RFC 2026, October 1996.

[3]   Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing
        for Message Authentication", RFC 2104, February 1997.

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

[5]   Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)",
        RFC 2409, November 1998.

[6]   Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631,
        June 1999.

[7]   Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer,
        H., Taylor, T., Rytina, I., Kalla, M., Zhang, L. and V. Paxson,
        "Stream Control Transmission Protocol", RFC 2960, October 2000.

[8]   Jungmaier, A., Rescorla, E. and M. Tuexen, "Transport Layer
        Security over Stream Control Transmission Protocol", RFC 3436,
        December 2002.

[9]   Kivinen, T. and M. Kojo, "More Modular Exponential (MODP)
        Diffie-Hellman groups for Internet Key Exchange (IKE)",
        RFC 3526, May 2003.

[10]   National Institute of Standards and Technology, "Secure Hash
        Standard", FIPS PUB 180-1, April 1995,
        <http://www.itl.nist.gov/fipspubs/fip180-1.htm>.

9.2 Informative References

[11]   Stewart, R., "Stream Control Transmission Protocol (SCTP)
        Dynamic Address Reconfiguration",
        Internet-Draft draft-ietf-tsvwg-addip-sctp-10, January 2005.

Authors' Addresses

   Michael Tuexen
   Univ. of Applied Sciences Muenster
   Stegerwaldstr. 39
   48565 Steinfurt
   Germany

   Email: tuexen@fh-muenster.de

   Randall R. Stewart
   Cisco Systems, Inc.
   4875 Forest Drive
   Suite 200
   Columbia, SC 29206
   USA

   Email: rrs@cisco.com

   Peter Lei
   Cisco Systems, Inc.
   8735 West Higgins Road
   Suite 300
   Chicago, IL 60631
   USA

   Phone:
   Email: peterlei@cisco.com

   Eric Rescorla
   RTFM, Inc.
   2064 Edgewood Drive
   Palo Alto, CA 94303
   USA

   Phone: +1 650-320-8549
   Email: ekr@rtfm.com

Intellectual Property Statement

Acknowledgment