Stream: Internet Engineering Task Force (IETF)
RFC: 9653
Category: Standards Track
Published: September 2024
ISSN: 2070-1721
Authors: M. Tüxen, *Münster Univ. of Appl. Sciences*; V. Boivie, *Google*; F. Castelli, *Google*; R. Jesup, *Mozilla*

RFC 9653
Zero Checksum for the Stream Control Transmission Protocol

Abstract

The Stream Control Transmission Protocol (SCTP) uses a 32-bit checksum in the common header of each packet to provide some level of data integrity. If another method used by SCTP already provides the same or a higher level of data integrity, computing this checksum does not provide any additional protection but does consume computing resources.

This document provides a simple extension allowing SCTP to save these computing resources by using zero as the checksum in a backwards-compatible way. It also defines how this feature can be used when SCTP packets are encapsulated in Datagram Transport Layer Security (DTLS) packets.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9653.

Copyright Notice

Table of Contents

1. Introduction

SCTP as specified in [RFC9260] uses a CRC32c checksum to provide some level of data integrity. When using, for example, Datagram Transport Layer Security (DTLS) as the lower layer for SCTP as specified in [RFC8261], using the CRC32c checksum does not provide any additional protection

over that already provided by DTLS. However, computing the CRC32c checksum at the sender and receiver sides does consume computational resources for no benefit. This is particularly important for endpoints that are computationally limited and use SCTP over DTLS.

The extension described in this document allows an SCTP endpoint to declare that it accepts SCTP packets with a checksum of zero when using a specific alternate error detection method. This declaration happens during the setup of the SCTP association and allows endpoints that support this extension to be interoperable with endpoints that don't. To provide this backwards compatibility, endpoints using this extension still need to implement the CRC32c checksum algorithm.

2. Conventions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Alternate Error Detection Methods

SCTP uses a CRC32c checksum to provide some level of data integrity. The CRC32c checksum is computed based on the SCTP common header and the chunks contained in the packet. In particular, the computation of the CRC32c checksum does not involve a pseudo header for IPv4 or IPv6 like the computation of the TCP checksum, as specified in [RFC9293], or the UDP checksum, as specified in [RFC0768].

Zero is a valid result of the CRC32c checksum algorithm. For example, the following figure depicts an SCTP packet containing a minimal INIT chunk with a correct CRC32c checksum of zero.

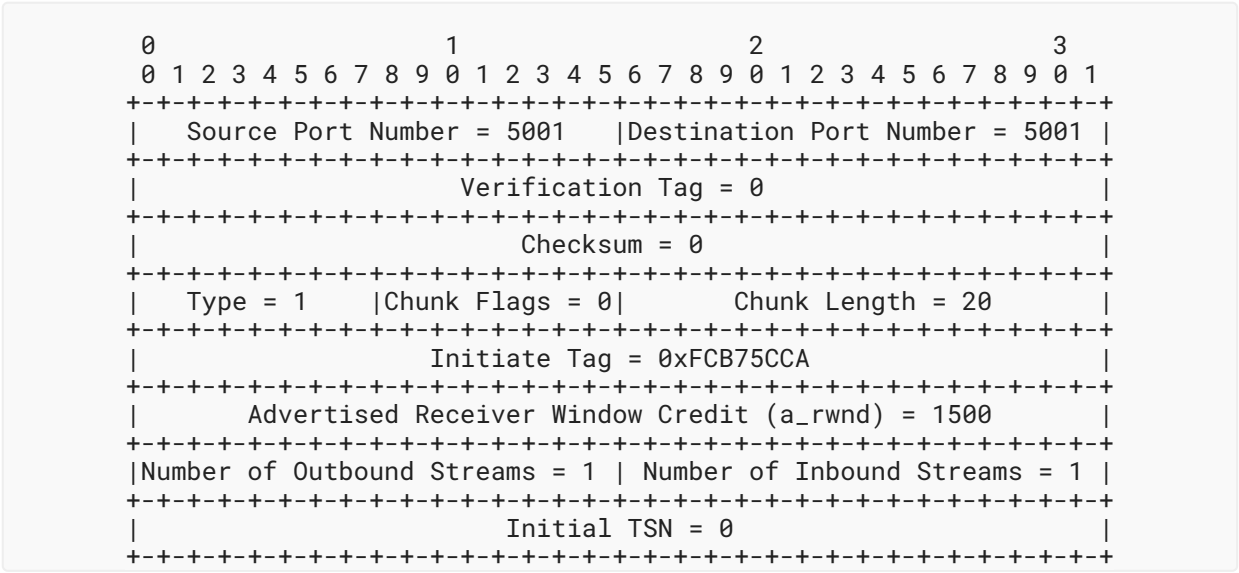

Figure 1: SCTP Packet with a Correct CRC32c Checksum of Zero

Using SCTP in combination with other mechanisms or protocol extensions might provide data integrity protection with an equal or lower probability of false negatives than the one provided by using the CRC32c checksum algorithm. When using such alternate error detection methods, the SCTP common header containing the 32-bit checksum field might or might not be visible to middleboxes on the paths between the two endpoints.

Alternate error detection methods have two requirements:

1. An alternate error detection method **MUST** provide an equal or lower probability of false negatives than the one provided by using the CRC32c checksum algorithm. This **MAY** only apply to packets satisfying some method-specific constraints.
2. Using an alternate error detection method **MUST NOT** result in a path failure for more than two retransmission timeouts (RTOs) due to middleboxes on the path expecting correct CRC32c checksums.

To fulfill the second requirement, alternate error detection methods could use a heuristic to detect the existence of such middleboxes and use correct CRC32c checksums on these affected paths.

Using DTLS as the lower layer of SCTP as specified in [RFC8261] is one example that fulfills the first requirement. Another example is using SCTP Authentication as specified in [RFC4895]. Of course, this only applies to each SCTP packet having an AUTH chunk as its first chunk. However, using SCTP Authentication without any heuristic does not fulfill the second requirement. Since using DTLS as the lower layer of SCTP as specified in [RFC8261] also fulfills the second requirement, it can be used as an alternate error detection method (see Section 6).

If an alternate error detection method is used, the computation of the CRC32c checksum consumes computational resources without providing any benefit. To avoid this, an SCTP endpoint could be willing to accept SCTP packets with an incorrect CRC32c checksum value of zero in addition to SCTP packets with correct CRC32c checksum values.

Because zero is a valid result of the CRC32c checksum algorithm, a receiver of an SCTP packet containing a checksum value of zero cannot determine whether the sender included an incorrect CRC32c checksum of zero to reduce the CPU cost or the result of the CRC32c checksum computation was actually zero. However, if the receiver is willing to use an alternate error detection method, this ambiguity is irrelevant, since the receiver is fine with not using the CRC32c checksum to protect incoming packets.

4. A New Chunk Parameter

The Zero Checksum Acceptable Chunk Parameter is defined by the following figure.

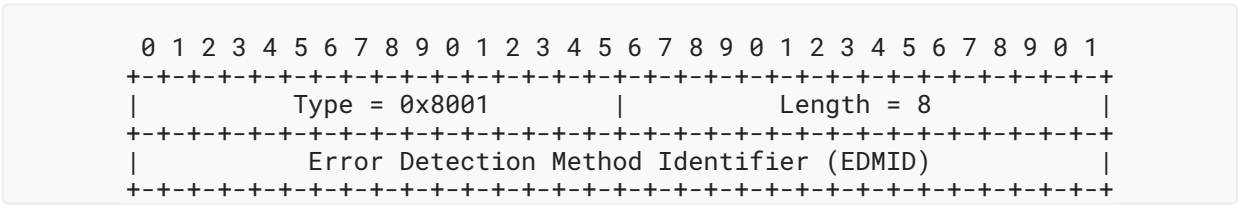

Figure 2: Zero Checksum Acceptable Chunk Parameter

Type: 16 bits (unsigned integer)
: This field holds the IANA-defined parameter type for the "Zero Checksum Acceptable" chunk parameter. IANA has assigned the value 32769 (0x8001) for this parameter type.

Length: 16 bits (unsigned integer)
: This field holds the length in bytes of the chunk parameter; the value **MUST** be 8.

Error Detection Method Identifier (EDMID): 32 bits (unsigned integer)
: An IANA-registered value specifying the alternate error detection method the sender of this parameter is willing to use for received packets.

All transported integer numbers are in network byte order, a.k.a. big endian.

The Zero Checksum Acceptable Chunk Parameter **MAY** appear in INIT and INIT ACK chunks and **MUST NOT** appear in any other chunk. The Parameter **MUST NOT** appear more than once in any chunk.

If an endpoint not supporting the extension described in this document receives this parameter in an INIT or INIT ACK chunk, it is **REQUIRED** to skip this parameter and continue to process further parameters in the chunk. This behavior is specified by [RFC9260] because the highest-order two bits of the Type are '10'.

5. Procedures

5.1. Declaration of Feature Support

An endpoint willing to accept SCTP packets with an incorrect checksum of zero **MUST** include the Zero Checksum Acceptable Chunk Parameter indicating the alternate error detection method it is willing to use in the INIT or INIT ACK chunk it sends.

An SCTP implementation **MAY** also require the upper layer to indicate that it is fine to use a specific alternate error detection method before including the corresponding Zero Checksum Acceptable Chunk Parameter.

5.2. Sender-Side Considerations

An SCTP endpoint cannot just use an incorrect CRC32c checksum value of zero for all SCTP packets it sends. The following restrictions apply:

1. If an endpoint has not received an INIT or INIT ACK chunk containing a Zero Checksum Acceptable Chunk Parameter indicating an alternate error detection method it supports from its peer during the association setup, it **MUST** use a correct CRC32c checksum. In particular, when an endpoint
   a. sends a packet containing an INIT chunk, it **MUST** include a correct CRC32c checksum in the packet containing the INIT chunk.
   b. responds to an "Out of the Blue" (OOTB) SCTP packet, it **MUST** include a correct CRC32c checksum in the response packet.
2. When an endpoint sends a packet containing a COOKIE ECHO chunk, it **MUST** include a correct CRC32c checksum in the packet containing the COOKIE ECHO chunk.
3. When an endpoint supports the dynamic address reconfiguration specified in [RFC5061] and sends a packet containing an ASCONF chunk, it **MUST** include a correct CRC32c checksum in the packet containing the ASCONF chunk.
4. If an alternate error detection method has some method-specific constraints, the sender **MUST** include a correct CRC32c checksum in all packets that don't fulfill these method-specific constraints.

The first restriction allows backwards compatibility. The second and third restrictions allow a simpler implementation of the extension defined in this document, because looking up the association for SCTP packets containing a COOKIE ECHO chunk or an ASCONF chunk might be more complex than for other packets. Finally, the last restriction covers constraints specific to the alternate error detection method.

An SCTP endpoint **MAY** require that the upper layer allow the use of the alternate error detection method that was announced by the peer before sending packets with an incorrect checksum of zero.

If none of the above restrictions apply, an endpoint **SHOULD** use zero as the checksum when sending an SCTP packet.

5.3. Receiver-Side Considerations

If an endpoint has sent the Zero Checksum Acceptable Chunk Parameter indicating the support of an alternate error detection method in an INIT or INIT ACK chunk, in addition to SCTP packets containing the correct CRC32c checksum value it **MUST** accept SCTP packets that have an incorrect checksum value of zero and that fulfill the requirements of the announced alternate error detection method used for this association. Otherwise, the endpoint **MUST** drop all SCTP packets with an incorrect CRC32c checksum.

In addition to processing OOTB packets with a correct CRC32c checksum as specified in [RFC9260], an SCTP implementation **MAY** also process OOTB packets having an incorrect zero checksum. Doing so might result in faster SCTP association failure detection.

6. Error Detection via SCTP over DTLS

Using SCTP over DTLS as specified in [RFC8261] provides a stronger error detection method than using the CRC32c checksum algorithm. Since middleboxes will not observe the unencrypted SCTP packet, there is no risk in interfering with using zero as an incorrect checksum. There are no additional constraints (specific to the error detection method) on packets when using DTLS encapsulation.

7. Socket API Considerations

This section describes how the socket API defined in [RFC6458] needs to be extended to provide a way for the application to control the acceptance of a zero checksum.

A 'Socket API Considerations' section is contained in all SCTP-related specifications published after [RFC6458] describing an extension for which implementations using the socket API as specified in [RFC6458] would require some extension of the socket API. Please note that this section is informational only.

A socket API implementation based on [RFC6458] is extended by supporting one new write-only IPPROTO_SCTP-level socket option.

7.1. Set Accepting a Zero Checksum (SCTP_ACCEPT_ZERO_CHECKSUM)

This IPPROTO_SCTP-level socket option with the name SCTP_ACCEPT_ZERO_CHECKSUM can be used to control the acceptance of a zero checksum. It is a write-only socket option and applies only to future SCTP associations on the socket.

This option expects an unsigned integer. Possible values include:

SCTP_EDMID_NONE:

Disable the use of any alternate error detection method. This means that all SCTP packets being received are only accepted if they have a correct CRC32c checksum value.

SCTP_EDMID_LOWER_LAYER_DTLS: Use the alternate error detection method described in Section 6.

An implementation might only send packets with an incorrect checksum of zero, if the alternate error detection method announced by the peer is also enabled locally via this socket option.

The default for this socket option is that the use of alternate error detection methods is disabled.

8. IANA Considerations

A new chunk parameter type has been assigned by IANA in the "Chunk Parameter Types" registry for SCTP:

ID Value	Chunk Parameter Type	Reference
32769	Zero Checksum Acceptable (0x8001)	RFC 9653

Table 1: New Entry in "Chunk Parameter Types" Registry

Furthermore, IANA has established a new "Error Detection Method" registry for SCTP. The assignment of new error detection methods is done through the Specification Required policy as defined in [RFC8126]. Documentation for a new error detection method **MUST** contain the following information:

1. A name of an alternate error detection method.
2. A reference to a specification describing:
   (a) the alternate error detection method,
   (b) why the alternate error detection method provides an equal or lower probability of false negatives than the one provided by using the CRC32c checksum,
   (c) any constraints (specific to the alternate error detection method) that are referred to in the fourth exception in Section 5.2, and
   (d) why using the alternate error detection method does not result in path failures due to middleboxes expecting correct CRC32c checksums for more than two RTOs. In case the alternate error detection method uses a heuristic for detecting such middleboxes, this heuristic needs to be described.

The initial contents of the registry are as follows:

ID Value	Error Detection Method	Reference
0	Reserved	RFC 9653

ID Value	Error Detection Method	Reference
1	SCTP over DTLS	RFC 9653
2 - 4294967295	Unassigned	

Table 2: Initial Contents of the "Error Detection Method" Registry

A designated expert (DE) is expected to ascertain the existence of suitable documentation (a specification) as described in [RFC8126] and to verify that the document is permanently and publicly available. Furthermore, the DE is expected to ensure that the above four points have been addressed appropriately.

9. Security Considerations

This document does not change the considerations given in [RFC9260].

Due to the first requirement in Section 3, using an alternate error detection method provides an equal or better level of data integrity than the one provided by using the CRC32c checksum algorithm. The second requirement in Section 3 ensures that the existence of middleboxes expecting correct CRC32c checksums does not result in permanent path failures.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5061] Stewart, R., Xie, Q., Tuexen, M., Maruyama, S., and M. Kozuka, "Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration", RFC 5061, DOI 10.17487/RFC5061, September 2007, <https://www.rfc-editor.org/info/rfc5061>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8261] Tuexen, M., Stewart, R., Jesup, R., and S. Loreto, "Datagram Transport Layer Security (DTLS) Encapsulation of SCTP Packets", RFC 8261, DOI 10.17487/RFC8261, November 2017, <https://www.rfc-editor.org/info/rfc8261>.

[RFC9260] Stewart, R., Tüxen, M., and K. Nielsen, "Stream Control Transmission Protocol", RFC 9260, DOI 10.17487/RFC9260, June 2022, <https://www.rfc-editor.org/info/rfc9260>.

10.2. Informative References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC4895] Tuexen, M., Stewart, R., Lei, P., and E. Rescorla, "Authenticated Chunks for the Stream Control Transmission Protocol (SCTP)", RFC 4895, DOI 10.17487/RFC4895, August 2007, <https://www.rfc-editor.org/info/rfc4895>.

[RFC6458] Stewart, R., Tuexen, M., Poon, K., Lei, P., and V. Yasevich, "Sockets API Extensions for the Stream Control Transmission Protocol (SCTP)", RFC 6458, DOI 10.17487/RFC6458, December 2011, <https://www.rfc-editor.org/info/rfc6458>.

[RFC9293] Eddy, W., Ed., "Transmission Control Protocol (TCP)", STD 7, RFC 9293, DOI 10.17487/RFC9293, August 2022, <https://www.rfc-editor.org/info/rfc9293>.

Acknowledgments

The authors wish to thank Bernard Aboba, Deb Cooley, Martin Duke, Gorry Fairhurst, Mike Heard, Peter Lei, Nils Ohlmeier, Claudio Porfiri, Greg Skinner, Timo Völker, Éric Vyncke, and Magnus Westerlund for their invaluable comments.

Authors' Addresses

Michael Tüxen
Münster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
Germany
Email: tuexen@fh-muenster.de

Victor Boivie
Google
Kungsbron 2
SE-11122 Stockholm
Sweden
Email: boivie@google.com

Florent Castelli
Google
Kungsbron 2
SE-11122 Stockholm
Sweden
Email: orphis@google.com

Randell Jesup
Mozilla Corporation
1835 Horse Shoe Trl
Malvern, PA 19355
United States of America
Email: randell-ietf@jesup.org