IP Flow Information Export WG	D. Mentz	
Internet-Draft	G. Muenz	
Intended status: Informational	L. Braun	
Expires: January 7, 2010	TU Muenchen	
	July 06, 2009	

TOC

Recommendations for Implementing IPFIX over DTLS <draft-mentz-ipfix-dtls-recommendations-00>

Status of this Memo

Copyright Notice

Abstract

This document discusses problems and solutions regarding the implementation of the IPFIX protocol over SCTP and DTLS.

Table of Contents

---

1. Introduction

TOC

All implementations of the IPFIX protocol conforming to [RFC5101] (Claise, B., “Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information,” January 2008.) must support DTLS [RFC4347] (Rescorla, E. and N. Modadugu, “Datagram Transport Layer Security,” April 2006.) if SCTP or UDP is used as transport protocol.
This document discusses specific issues that have arisen during the implementation of the IPFIX protocol over DTLS. These issues may degrade the performance of an IPFIX Exporter as they require to periodically interrupt the data export. As a solution, we proposes workarounds which solve these problems without requiring any changes to DTLS and the IPFIX protocol.
This document is to be considered as a possible update of the IPFIX Implementation Guidelines [RFC5153] (Boschi, E., Mark, L., Quittek, J., Stiemerling, M., and P. Aitken, “IP Flow Information Export (IPFIX) Implementation Guidelines,” April 2008.).

TOC

2. Terminology

This document adopts the IPFIX terminology used in [RFC5101] (Claise, B., “Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information,” January 2008.). As in all IPFIX document, all IPFIX specific terms have the first letter of a word capitalized when used in this document.

---

3. Issues and Recommendations Regarding IPFIX over DTLS/UDP

TOC

3.1. Undetected Collector Crashes

TOC

DTLS has been conceived for deployment on top of unreliable transport protocols, such as UDP. Hence, it is able to cope with lost datagrams and datagrams that arrive out of order at the receiver. In contrast to UDP, which does not maintain any connection state, DTLS has to maintain state across multiple datagrams at both endpoints. This state is established during the DTLS handshake.

During the DTLS handshake, the two peers authenticate each other and agree upon several parameters which are necessary to communicate over DTLS. Among these parameters are a cipher suite as well as a shared key that is usually established using a Diffie-Hellman key exchange. If one of the peers crashes unexpectedly, these parameters as well as the maintained DTLS state usually get lost. As a consequence, the peer is not able to check the integrity of newly arrived datagrams or to decrypted the datagrams' payload.

In the case of connection oriented transport protocols, such as TCP or SCTP, a connection endpoint will be informed about the crash of its correspondent by the transport protocol. UDP, however, is connection less, which means that the crash of the receiver is not noticed by the sender. There are situations in which the sender might receive ICMP messages that indicate that the receiver is experiencing problems, but there is no guarantee that those ICMP messages will be sent. As DTLS itself does not provide any mechanisms for dead peer detection, the crash of one of the peers has to be detected and handled by protocols in the upper layers.

As IPFIX is a unidirectional protocol, a conform implementation of an IPFIX Exporter only sends but does not receive any data. Hence, the Exporter cannot tell from the absence of returning traffic that the Collector has crashed. Instead, the Exporter keeps on sending data which must be discarded by the recovered Collector because the information needed to check the integrity and to decrypt the data is lost.

3.2. Possible Solutions

TOC

There are three options to circumvent this problem.

1. The first option is to let the Exporter periodically trigger renegotiations on the DTLS layer. This means that both peers have to participate in a new handshake, implying the exchange of datagrams in both direction. Hence, due to a timeout, the Exporter will notice that the Collector has crashed.

   Under normal conditions, such a renegotiation is used to renew the keying material in a long living connection. Depending on whether a full or abbreviated handshake is carried out, such a renegotiation can be very costly in terms of computational overhead because it involves public key operations. In addition, the DTLS specification [RFC4347] (Rescorla, E. and N. Modadugu, “Datagram Transport Layer Security,” April 2006.) leaves open if application data can be sent during the handshake or not. Typical implementations, such as OpenSSL [OpenSSL] (, “OpenSSL Cryptography and SSL/TLS Toolkit,” 2009.), require that sending data is interrupted until the handshake is finished. Consequently, the export of IPFIX Messages must be stalled for at least two round trip times, which could lead to IPFIX Messages queuing up in the buffer of the Exporting Process and potential loss of data.

   The most substantial argument against this solution is that the renegotiation was simply not conceived to serve as a dead peer detection mechanism. To make sure that the Exporter learns quickly about a crashed Collector, renegotiations would have to be carried out in short intervals.

2. The authors of this draft endorse the second option which is to periodically establish new DTLS connections and replace the active DTLS connection by a new one. Establishing a new DTLS connection involves a bidirectional handshake which requires both peers to be alive. If the Collector crashes unexpectedly and recovers quickly, then the time during which he receives meaningless data is limited until a new DTLS connection is established. Care should be taken that two successive connections overlap in a way such that no data is lost at the Exporting Process. This can be achieved by swapping the connections only after all active templates have been sent out

on the new DTLS connection.

As regards the computational overhead, this solution suffers from the same limitations as the first one. Every new DTLS connection might involve costly public key operations and a small overhead in terms of the transmitted data volume. However, public key operations do not have to be carried out if the DTLS implementations support a feature called session resumption which allows the reuse of keying material from an earlier session. This feature could also be used to simplify the renegotiation proposed in the first solution.

The main advantage over periodical renegotiations is that this solution does not suffer from the data stall that is due to the fact that OpenSSL does not allow sending application data during handshakes. IPFIX records can be transmitted without interruption due to the overlap of the old and the new DTLS connection.

From the point of view of IPFIX, every new DTLS connection represents a new Transport Session. At the collector side, however, it should be straight forward to associate the different Transport Sessions to the same Exporter since the exporter IP address remains the same. At the beginning of every new Transport Session, not only all active Templates have to be sent, but also certain Data Records defined by Option Templates. In the case of UDP, however, this does not cause significant additional overhead because Templates and Data Records defined by Option Templates are periodically resent anyway.

3. A third alternative to detect a dead or recovered collector is to implement the DTLS Heartbeat Extension which has been very recently suggested in [I-D.seggelmann-tls-dtls-heartbeat] (Seggelmann, R., Tuexen, M., and M. Williams, “Datagram Transport Layer Security Heartbeat Extension,” July 2009.). This DTLS extension allows detecting a dead peer without interfering with the ongoing data transfer. The computational and bandwidth overhead is negligible plus the data transmission does not stall.

   The problems with this solution are that the necessary DTLS extension has not yet been standardized and that there are literally no implementations available at the time of writing.

4. Issues and Recommendations Regarding IPFIX over DTLS/SCTP

TOC

4.1. Renegotiation for DTLS and SCTP-AUTH

TOC

The DTLS binding for SCTP is more sophisticated than the DTLS/UDP binding. This is due to the fact that SCTP provides a connection oriented service to upper layers. It also carries additional data items with each user message. Among those items are:

- stream ID
- payload protocol identifiers

DTLS only protects the bare user data. Without additional security mechanisms, a man-in-the-middle attacker could easily tamper with the stream ID or the payload protocol identifier. He could also defeat SCTP's efforts to provide a reliable or partially reliable service by forging SACK and Forward-TSN chunks.
The solution to this problem is SCTP-AUTH [RFC4895] (Tuexen, M., Stewart, R., Lei, P., and E. Rescorla, “Authenticated Chunks for the Stream Control Transmission Protocol (SCTP),” August 2007.) which allows the SCTP implementation to insert an authentication chunk which authenticates certain types of subsequent chunks in the same packet using a Hashed Message Authentication Code (HMAC). While SCTP-AUTH allows for the negotiation of the hash algorithm it does not provide means for secure key agreement. Therefore a cross layer approach is used to extract keying material from the DTLS layer and use it on the SCTP layer. This approach is described in [I-D.ietf-tsvwg-dtls-for-sctp] (Tuexen, M., Seggelmann, R., and E. Rescorla, “Datagram Transport Layer Security for Stream Control Transmission Protocol,” October 2008.) and is readily available in OpenSSL.
Due to limitations of DTLS, no renegotiation (i.e., change of keying material) can be performed without impeding the ongoing data transfer. The implementation has to make sure that there is no data in flight at the point in time that the keying material is swapped. This means that data transfer on all streams has to stop before a renegotiation can be initiated. Moreover, there must not be any unacknowledged messages in the send buffers of the SCTP sockets. In practice, the renegotiation has to wait until the SCTP sockets at both endpoints return SCTP_SENDER_DRY_EVENT [I-D.ietf-tsvwg-sctpsocket] (Stewart, R., Poon, K., Tuexen, M., Yasevich, V., and P. Lei, “Sockets API Extensions for Stream Control Transmission Protocol (SCTP),” March 2010.). Only after the handshake has been completed, the data transfer can continue because DTLS does not guarantee the proper authentication and

decryption of user messages that were secured with outdated keying material.
In the case of IPFIX, this means that the Exporting Process has to interrupt the export of IPFIX Messages for a certain period of time. IPFIX Messages generated in the meantime have to be buffered or dropped until the renegotiation is over.

4.2. Possible Solutions

TOC

To solve this problem of data stall, the authors of this draft suggest to employ the same mechanism as in the UDP case, which is to periodically establish a new DTLS/SCTP association between Exporter and Collector in parallel to the existing one. Only after the handshakes of SCTP and DTLS are completed and the IPFIX Templates are sent on the new association, the Exporter starts sending Data Records on the new Transport Session.
Again, the establishment of a new SCTP association represents a new IPFIX Transport Session. Some overhead is produced because Templates as well as certain Data Records defined by Option Template have to be resent, which would not be necessary if the old Transport Session was kept. However, the amount of additional data that has to be sent is assumed to be rather small.

---

5. Security Considerations

TOC

The recommendations in this document do not introduce any additional security issues to those already mentioned in [RFC5101] (Claise, B., “Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information,” January 2008.).

Appendix A. Acknowledgements

TOC

The authors thank Michael Tuexen and Robin Seggelmann for their contribution on the standardization and implementation of DTLS for SCTP as well as for their valuable advice regarding the implementation of IPFIX over DTLS.

TOC

6. References

6.1. Normative References

TOC

[RFC5101]	Claise, B., "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information," RFC 5101, January 2008 (TXT).

6.2. Informative References

TOC

[RFC5153]	Boschi, E., Mark, L., Quittek, J., Stiemerling, M., and P. Aitken, "IP Flow Information Export (IPFIX) Implementation Guidelines," RFC 5153, April 2008 (TXT).
[RFC3758]	Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension," RFC 3758, May 2004 (TXT).
[RFC4960]	Stewart, R., "Stream Control Transmission Protocol," RFC 4960, September 2007 (TXT).
[RFC4347]	Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security," RFC 4347, April 2006 (TXT).
[OpenSSL]	"OpenSSL Cryptography and SSL/TLS Toolkit," Homepage http://www.openssl.org/, 2009.
[I-D.seggelmann-tls-dtls-heartbeat]	Seggelmann, R., Tuexen, M., and M. Williams, "Datagram Transport Layer Security Heartbeat Extension," draft-seggelmann-tls-dtls-heartbeat-00 (work in progress), July 2009 (HTML).
[RFC4895]	Tuexen, M., Stewart, R., Lei, P., and E. Rescorla, "Authenticated Chunks for the Stream Control Transmission Protocol (SCTP)," RFC 4895, August 2007 (TXT).
[I-D.ietf-tsvwg-dtls-for-sctp]	Tuexen, M., Seggelmann, R., and E. Rescorla, "Datagram Transport Layer Security for Stream Control Transmission Protocol," draft-seggelmann-tls-dtls-heartbeat-00 (work in progress), October 2008 (HTML).
[I-D.ietf-tsvwg-sctpsocket]	Stewart, R., Poon, K., Tuexen, M., Yasevich, V., and P. Lei, "Sockets API Extensions for Stream Control Transmission Protocol (SCTP)," draft-ietf-tsvwg-sctpsocket-22 (work in progress), March 2010 (TXT).

Authors' Addresses

TOC

	Daniel Mentz
	Technische Universitaet Muenchen
	Department of Informatics
	Chair for Network Architectures and Services (I8)
	Boltzmannstr. 3
	Garching D-85748
	DE
Email:	mentz@in.tum.de
	Gerhard Muenz
	Technische Universitaet Muenchen
	Department of Informatics
	Chair for Network Architectures and Services (I8)
	Boltzmannstr. 3
	Garching D-85748
	DE
Phone:	+49 89 289-18008
Email:	muenz@net.in.tum.de
URI:	http://www.net.in.tum.de/~muenz
	Lothar Braun
	Technische Universitaet Muenchen
	Department of Informatics
	Chair for Network Architectures and Services (I8)
	Boltzmannstr. 3
	Garching D-85748
	DE
Phone:	+49 89 289-18010
Email:	braun@net.in.tum.de
URI:	http://www.net.in.tum.de/~braun