IP Flow Information Export WG
Internet-Draft
Intended status: Informational
Expires: September 15, 2011

D. Mentz
G. Muenz
L. Braun
TU Muenchen
March 14, 2011

Recommendations for Implementing IPFIX over DTLS <draft-mentz-ipfix-dtls-recommendations-02>

Abstract

This document discusses problems and solutions regarding the implementation of the IPFIX protocol over DTLS. It updates the "IPFIX Implementation Guidelines" [RFC5153].

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 15, 2011.

Table of Contents

1. Introduction

All implementations of the IPFIX protocol conforming to [RFC5101] must support DTLS [RFC4347] if SCTP or UDP is selected as IPFIX transport protocol. This document discusses specific issues that have arisen during the implementation of the IPFIX protocol over DTLS (the source code of the implementation is available as part of VERMONT [VERMONT]).

Section 3 discusses two issues which may lead to the loss of IPFIX Messages if DTLS is used with UDP as transport protocol: unexpected Collector crashes and wrong path MTU values. In the first case, the data loss may even not be recognized by the Collector. By following the recommendations of this document, these two problems can be avoided.

Section 4 discusses one issue which corresponds to the implementation of IPFIX over DTLS/SCTP. In this case, DTLS renegotiations require the interruption of the data export for a short period of time, which may lead to the queuing and potential loss of IPFIX Messages at the Exporting Process. For Exporters that operate at a high data rate, it is recommended to switch over to a newly established DTLS/SCTP Transport Session instead of triggering DTLS renegotiation for an existing Transport Session.

When the "IPFIX Implementation Guidelines" were published [RFC5153], no implementation of IPFIX over DTLS/UDP or DTLS/SCTP actually existed. Therefore, Sections 8.4 and 8.5 of [RFC5153] are incomplete and do not cover the issues described in this document. Hence, the recommendations of this document complement and update the "IPFIX Implementation Guidelines" [RFC5153].

Finally, Section 5 suggests to support the pre-shared key ciphersuites for TLS for mutual authentication. These ciphersuites can do without a public-key infrastructure (PKI) and can therefore facilitate the setup of an environment with a limited number of IPFIX devices.

2. Terminology

This document adopts the IPFIX terminology used in [RFC5101]. As in all IPFIX documents, 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

Regarding IPFIX over DTLS/UDP, [RFC5101] and [RFC5153] refer to [RFC4347] which specifies the usage of DTLS over the transport protocol UDP. [RFC5153] explains that Exporting Processes and Collecting Processes should behave as if UDP without DTLS was transport protocol.

During the implementation of IPFIX over DTLS/UDP, it turned out that the specification of [RFC4347] is insufficient for IPFIX data export because the loss of DTLS state at the Collecting Process may not be detected by the Exporting Process. As a consequence, it remains unnoticed that all further IPFIX Messages arriving at the Collecting Process must be discarded. This issue as well as recommendations how to solve it are discussed in Section 3.1.

For IPFIX export over UDP, [RFC5101] specifies that the total packet size of IPFIX Messages must not exceed the path MTU (PMTU). Section 8.4 of [RFC5153] points out that DTLS introduces overhead which affects the packet size. In fact, the utilization of DTLS affects the packet size, yet it does not generally result in larger packet sizes. In particular, if the IPFIX Message is compressed before being encrypted, the size of the DTLS record is likely to be smaller than the original IPFIX Message. However, since the compression ratio cannot be predicted, it is save to make conservative assumptions about the DTLS record size.

Another general problem regarding the utilization of UDP as transport protocol is that the total packet size should not exceed 512 octets if the PMTU is not available [RFC5101]. Since the PMTU is usually larger than 512 octets, this limitation causes overhead due to unnecessarily small IPFIX Messages. Hence, there is an interest to provide the Exporting Process with a correct PMTU value.

If the PMTU is known, it can be configured by the user. Otherwise, the PMTU can be determined by PMTU discovery mechanisms defined in [RFC1191] and [RFC1981]. However, these mechanisms do not always provide reliable results. Section 3.2 discusses this issue in more detail and presents a better PMTU discovery mechanism for DTLS/UDP.

3.1. Undetected Collector Crashes

3.1.1. Problem Description

DTLS has been conceived for deployment on top of unreliable transport protocols, such as UDP. Hence, the handshaking protocol of DTLS 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 and initialized during the DTLS handshake [RFC4347].

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 indicating that the receiver is experiencing problems, for example if an ICMP port unreachable message is returned because the UDP port is closed. However, there is no guarantee that these ICMP messages will be sent. Also, implementations should ignore these messages as they are not authenticated and might therefore be forged. DTLS as specified in [RFC4347] does not provide any mechanisms for dead peer detection, thus 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 conforming 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.

3.1.2. Recommendation

The DTLS heartbeat extension which has been suggested in [I-D.seggelmann-tls-dtls-heartbeat] allows a DTLS endpoint to detect a dead peer. With this extension, each endpoint may transmit DTLS heartbeat request messages to the other peer. Each peer is supposed to send back a heartbeat response message for every heartbeat request message it receives. As UDP provides unreliable transport, it may happen that heartbeat request or response messages are lost. Nevertheless, a peer can be declared dead if it fails to respond to a certain number of consecutive heartbeat requests.

The computational and bandwidth overhead of the heartbeat messages is very small. As another advantage, the exchange of heartbeat messages does not affect the transport of user data. In particular, the transport of user data does not have to be interrupted.

IPFIX Exporters and Collectors should support the DTLS heartbeat extension to allow an Exporting Process to check whether the Collecting Process is still able to decrypt the exported IPFIX Messages. To detect a crashed Collector, the Exporting Process must actively trigger the sending of a DTLS heartbeat request message. This should be done on a regular basis (e.g., periodically). It must be noted that a dead peer remains undetected in the time interval between two successive heartbeat requests.

The only problem with this solution is that the DTLS heartbeat extension has not yet been standardized.

3.1.3. Alternative Workarounds

If the DTLS heartbeat extension is not available, there exist two workarounds which also enable the detection of a crashed Collector. However, these approaches have several disadvantages compared to heartbeat messages.

1. The first option is to let the Exporting Process periodically trigger renegotiations on the DTLS layer. During a renegotiation, the Collecting Process has to participate in a new handshake, implying the exchange of datagrams in both direction. If a Collector has crashed, it cannot respond to the handshake messages. Thus, the absence of any return messages during the renegotiation tells the Exporter that the Collector has probably lost the DTLS state.

   Under normal conditions, renegotiations are used to renew the keying material in a long living connection. Depending on whether a full or abbreviated handshake is carried out, a renegotiation can be very costly in terms of computational overhead because it involves public key operations. In addition, the DTLS specification [RFC4347] leaves open if user data can be sent while the rehandshake is in progress or if data transmission has to pause. Typical implementations, such as OpenSSL [OpenSSL], require data transmission to pause until the handshake is completed. 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.

   To make sure that the Exporter learns quickly about a crashed

Collector, renegotiations would have to be carried out on a regular basis.

2. Another approach is to periodically establish new DTLS connections and replace the existing DTLS connection by a new one. Establishing a new DTLS connection involves a bidirectional handshake which requires both peers to be alive. Two successive connections should overlap in a way such that no IPFIX Message is lost. This can be achieved by switching to the new connections only after all Templates have been sent.

   This solution has the same computational overhead as the first workaround. Every DTLS connection setup might involve costly public key operations and a small overhead in terms of the transmitted packets. However, public key operations do not have to be carried out if both DTLS implementations support a feature called session resumption which allows the reuse of keying material from an earlier session.

   The main advantage over periodical DTLS renegotiations is that this solution does not require to stall the transmission of user data. IPFIX records can be transmitted without interruption thanks 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, the different Transport Sessions can be easily associated 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 need to be resent periodically anyway.

3.2. Incorrect Path MTU Values

3.2.1. Problem Description

[RFC5101] states that the Exporter must not generate IPFIX Messages that result in IP packets which are larger than the PMTU. The mechanism that is commonly used to discover the PMTU is described in [RFC1191] and [RFC1981] and works as follows: The sender sets the Don't Fragment (DF) bit on all outgoing IP packets, which bans the routers on the path from fragmenting these IP packets. If a router on the path cannot forward a packet because it is larger than the MTU of the outbound link, it discards the packet and sends back an ICMP "fragmentation needed and DF set" message [RFC0792]. This message

also includes a hint about the MTU of the outbound link. Upon receiving this ICMP message, the sender updates its PMTU estimate for this specific destination IP address. In order to avoid that future packets are discarded, the sender limits, from now on, the size of IP packets to the current PMTU estimate. This new estimate may or may not be the final PMTU estimate as there are potentially other links further down the path with even smaller MTUs. The PMTU discovery process is therefore repeated until all IP packets that are as big as the PMTU estimate are delivered to the destination.

An important characteristic of this mechanism is that at least one UDP datagram is lost per update of the PMTU estimate. Hence, if deployed by an IPFIX Exporting Process, a certain number of IPFIX Messages will be lost until the final PMTU estimate is found. A more severe problem is that ICMP messages may be blocked by firewalls. As a result, the PMTU discovery mechanism fails without being noticed by the Exporting Process. Instead, the Exporting Process sticks to an incorrect PMTU estimate which is larger than the true PMTU. As a consequence, all packets which exceed the actual PMTU will be discarded on their way to the Collector, given that the "don't fragment" bit is set for all packets.

3.2.2. Recommendation

If DTLS is used, the PMTU can be determined with the DTLS heartbeat extension [I-D.seggelmann-tls-dtls-heartbeat] which has already been presented as solution to the dead peer detection problem in Section 3.1.2. This DTLS extension enables the Exporting Process to send heartbeat request messages which have the size of the PMTU estimate. If the Collecting Process acknowledges the reception of such a heartbeat request messages with a heartbeat response message, the Exporting Process knows that the PMTU estimate is less than or equal to the real PMTU to the Collector. If there is no response, the Exporting Process reduces the PMTU estimate and tries to send another heartbeat request message with the size of the new PMTU estimate. This procedure is repeated until the Exporting Process receives a heartbeat response messages. Since packets may be lost due to other reasons as well, every PMTU estimate should be probed in multiple attempts.

The described PMTU discovery mechanism can be used in conjunction with [RFC1191]. If a heartbeat request messages triggers an ICMP "fragmentation needed and DF set" message, the Exporting Process may decrease the PMTU estimate according to the returned MTU value. As a general advantage, only DTLS heartbeat messages are involved in the PMTU discovery. Hence, if the PMTU discovery using heartbeat messages is completed before starting the IPFIX export, no IPFIX Messages will be lost because of their size.

Since the PMTU may change over time due to routing changes, PMTU discovery with heartbeat messages should be repeated on a regular basis in order to ensure that the PMTU estimate is kept up to date.

4. Issues and Recommendations Regarding IPFIX over DTLS/SCTP

When [RFC5153] was published, the standardization of DTLS for SCTP was not yet completed. Therefore, the guidelines regarding the implementation of IPFIX over DTLS/SCTP are incomplete as well. In particular, [RFC5153] does not mention that DTLS for SCTP, as specified in [RFC6083], requires that the SCTP implementation supports the SCTP-AUTH extension [RFC4895]. The relationship between SCTP-AUTH and DTLS is explained in Section 4.1. As another change to [RFC5153], an implementation of DTLS for SCTP is now available at <http://sctp.fh-muenster.de/>.

If IPFIX data is exported over DTLS/SCTP, the export needs to be interrupted during DTLS renegotiations. For situations where this is unacceptable, Section 4.2 presents a workaround.

4.1. SCTP-AUTH

DTLS only protects the user data transported by SCTP. SCTP-AUTH is needed to protect SCTP control information which could otherwise be tampered with by an attacker. For example, a man-in-the-middle attacker could easily tamper with the stream ID or the payload protocol identifier of a data chunk. If PR-SCTP is used, an attacker may even suppress data chunks without being detected by forging SACK and FORWARD-TSN chunks.

SCTP-AUTH [RFC4895] deploys authentication chunks to authenticate certain types of subsequent chunks in the same packet using a hashed message authentication code (HMAC). While SCTP-AUTH enables the negotiation of the hash algorithm, it provides no means for secure key agreement. Therefore, a cross layer approach is used to extract keying material from the DTLS layer and use it in the SCTP layer. This approach is described in [RFC6083] and is readily available in OpenSSL.

4.2. Renegotiation for DTLS and SCTP-AUTH

4.2.1. Problem Description

A DTLS renegotiation (i.e., change of keying material) requires to interrupt the ongoing data transfer because DTLS does not guarantee the proper authentication and decryption of user messages that were secured with outdated keying material. The implementation has to

make sure that no data is in flight when the keying material is exchanged. This means that data transfer on all SCTP streams has to stop before a renegotiation can be initiated. Moreover, all data chunks in the send buffer need to be acknowledged before the renegotiation can start. In practice, the renegotiation has to wait until the SCTP sockets at both endpoints return SCTP_SENDER_DRY_EVENT [I-D.ietf-tsvwg-sctpsocket]. Only after the handshake has been completed, the data transfer can be resumed.

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 completed.

4.2.2. Recommendation

If an Exporting Process exports IPFIX Messages at a very high rate, it is probably impossible to buffer IPFIX Messages during a DTLS renegotiation. In order to avoid that IPFIX Messages need to be dropped at the Exporter, DTLS renegotiations should not be performed in such situations. If the keying material needs to be changed, a better solution is to establish a new DTLS/SCTP association to the same Collector. After completing the handshakes of SCTP and DTLS and after sending the IPFIX Templates on the new association, the Exporting Process switches to the new Transport Session.

Compared to a renegotiation, 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. Mutual Authentication via Pre-Shared Keys

[RFC5101] mandates strong mutual authentication of Exporters and Collectors via asymmetric keys which are stored in X.509 certificates. This enables the user to take advantage of a public-key infrastructure (PKI) and let the endpoints verify the identity of their peers by using this infrastructure.

While a PKI is beneficial in an environment with a large number of endpoints that potentially communicate with each other, the cost of maintaining a PKI maybe disproportionate in smaller environments. [RFC4279] defines a set of new ciphersuites that use pre-shared keys instead of asymmetric keys for mutual authentication and therefore do not require a PKI. Allowing IPFIX implementations to use these ciphersuites can lower the administrative burden of setting up an

IPFIX connection that is based on DTLS or TLS. These ciphersuites are also of benefit to performance-constrained environments as they do not require computational expensive public key operations.

If the IPFIX specification allows these new ciphersuites to be used, it still has to be decided which identity type Exporters send with the ClientKeyExchange message. Refer to Section 5 of [RFC4279] for more details. The authors recommend to use the Fully Qualified Domain Name (FQDN) of the Exporter as the identity when initiating a connection. The security considerations outlined in Section 7 of [RFC4279] apply.

6. Security Considerations

The recommendations in this document do not introduce any additional security issues to those already mentioned in [RFC5101] and [RFC4279].

Appendix A. Acknowledgements

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.

7. References

7.1. Normative References

[RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.

[RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.

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

7.2. Informative References

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.

[RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.

[RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, May 2004.

[RFC4895] Tuexen, M., Stewart, R., Lei, P., and E. Rescorla, "Authenticated Chunks for the Stream Control Transmission Protocol (SCTP)", RFC 4895, August 2007.

[RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.

[RFC5153] Boschi, E., Mark, L., Quittek, J., Stiemerling, M., and P. Aitken, "IP Flow Information Export (IPFIX) Implementation Guidelines", RFC 5153, April 2008.

[RFC6083] Tuexen, M., Seggelmann, R., and E. Rescorla, "Datagram Transport Layer Security (DTLS) for Stream Control Transmission Protocol (SCTP)", RFC 6083, January 2011.

[VERMONT] "VERMONT (VERsatile MONitoring Toolkit)", Homepage http://vermont.berlios.de/, 2010.

[OpenSSL] "OpenSSL Cryptography and SSL/TLS Toolkit", Homepage http://www.openssl.org/, 2010.

[I-D.seggelmann-tls-dtls-heartbeat] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security and Datagram Transport Layer Security Heartbeat Extension", draft-seggelmann-tls-dtls-heartbeat-02 (work in progress), February 2010.

[I-D.ietf-tsvwg-sctpsocket] Stewart, R., Tuexen, M., Poon, K., Lei, P., and V. Yasevich, "Sockets API Extensions for Stream Control Transmission Protocol (SCTP)", draft-ietf-tsvwg-sctpsocket-27 (work in progress), March 2011.

Authors' Addresses

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