            Transport Layer Security (TLS) and
   Datagram Transport Layer Security (DTLS) Heartbeat Extension

Abstract

   This document describes the Heartbeat Extension for the Transport
   Layer Security (TLS) and Datagram Transport Layer Security (DTLS)
   protocols.

   The Heartbeat Extension provides a new protocol for TLS/DTLS allowing
   the usage of keep-alive functionality without performing a
   renegotiation and a basis for path MTU (PMTU) discovery for DTLS.

Copyright Notice

Table of Contents

1. Introduction

1.1. Overview

   This document describes the Heartbeat Extension for the Transport
   Layer Security (TLS) and Datagram Transport Layer Security (DTLS)
   protocols, as defined in [RFC5246] and [RFC6347] and their
   adaptations to specific transport protocols described in [RFC3436],
   [RFC5238], and [RFC6083].

   DTLS is designed to secure traffic running on top of unreliable
   transport protocols. Usually, such protocols have no session
   management. The only mechanism available at the DTLS layer to figure
   out if a peer is still alive is a costly renegotiation, particularly
   when the application uses unidirectional traffic. Furthermore, DTLS
   needs to perform path MTU (PMTU) discovery but has no specific
   message type to realize it without affecting the transfer of user
   messages.

TLS is based on reliable protocols, but there is not necessarily a
feature available to keep the connection alive without continuous
data transfer.

The Heartbeat Extension as described in this document overcomes these
limitations. The user can use the new HeartbeatRequest message,
which has to be answered by the peer with a HeartbeartResponse
immediately. To perform PMTU discovery, HeartbeatRequest messages
containing padding can be used as probe packets, as described in
[RFC4821].

1.2. Conventions

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 [RFC2119].

2. Heartbeat Hello Extension

The support of Heartbeats is indicated with Hello Extensions. A peer
cannot only indicate that its implementation supports Heartbeats, it
can also choose whether it is willing to receive HeartbeatRequest
messages and respond with HeartbeatResponse messages or only willing
to send HeartbeatRequest messages. The former is indicated by using
peer_allowed_to_send as the HeartbeatMode; the latter is indicated by
using peer_not_allowed_to_send as the Heartbeat mode. This decision
can be changed with every renegotiation. HeartbeatRequest messages
MUST NOT be sent to a peer indicating peer_not_allowed_to_send. If
an endpoint that has indicated peer_not_allowed_to_send receives a
HeartbeatRequest message, the endpoint SHOULD drop the message
silently and MAY send an unexpected_message Alert message.

The format of the Heartbeat Hello Extension is defined by:

```
enum {
   peer_allowed_to_send(1),
   peer_not_allowed_to_send(2),
   (255)
} HeartbeatMode;

struct {
   HeartbeatMode mode;
} HeartbeatExtension;
```

Upon reception of an unknown mode, an error Alert message using
illegal_parameter as its AlertDescription MUST be sent in response.

3. Heartbeat Protocol

   The Heartbeat protocol is a new protocol running on top of the Record
   Layer. The protocol itself consists of two message types:
   HeartbeatRequest and HeartbeatResponse.

   enum {
      heartbeat_request(1),
      heartbeat_response(2),
      (255)
   } HeartbeatMessageType;

   A HeartbeatRequest message can arrive almost at any time during the
   lifetime of a connection. Whenever a HeartbeatRequest message is
   received, it SHOULD be answered with a corresponding
   HeartbeatResponse message.

   However, a HeartbeatRequest message SHOULD NOT be sent during
   handshakes. If a handshake is initiated while a HeartbeatRequest is
   still in flight, the sending peer MUST stop the DTLS retransmission
   timer for it. The receiving peer SHOULD discard the message
   silently, if it arrives during the handshake. In case of DTLS,
   HeartbeatRequest messages from older epochs SHOULD be discarded.

   There MUST NOT be more than one HeartbeatRequest message in flight at
   a time. A HeartbeatRequest message is considered to be in flight
   until the corresponding HeartbeatResponse message is received, or
   until the retransmit timer expires.

   When using an unreliable transport protocol like the Datagram
   Congestion Control Protocol (DCCP) or UDP, HeartbeatRequest messages
   MUST be retransmitted using the simple timeout and retransmission
   scheme DTLS uses for flights as described in Section 4.2.4 of
   [RFC6347]. In particular, after a number of retransmissions without
   receiving a corresponding HeartbeatResponse message having the
   expected payload, the DTLS connection SHOULD be terminated. The
   threshold used for this SHOULD be the same as for DTLS handshake
   messages. Please note that after the timer supervising a
   HeartbeatRequest messages expires, this message is no longer
   considered in flight. Therefore, the HeartbeatRequest message is
   eligible for retransmission. The retransmission scheme, in
   combination with the restriction that only one HeartbeatRequest is
   allowed to be in flight, ensures that congestion control is handled
   appropriately in case of the transport protocol not providing one,
   like in the case of DTLS over UDP.

When using a reliable transport protocol like the Stream Control
Transmission Protocol (SCTP) or TCP, HeartbeatRequest messages only
need to be sent once. The transport layer will handle
retransmissions. If no corresponding HeartbeatResponse message has
been received after some amount of time, the DTLS/TLS connection MAY
be terminated by the application that initiated the sending of the
HeartbeatRequest message.

4. Heartbeat Request and Response Messages

   The Heartbeat protocol messages consist of their type and an
   arbitrary payload and padding.

```
struct {
   HeartbeatMessageType type;
   uint16 payload_length;
   opaque payload[HeartbeatMessage.payload_length];
   opaque padding[padding_length];
} HeartbeatMessage;
```

   The total length of a HeartbeatMessage MUST NOT exceed 2^14 or
   max_fragment_length when negotiated as defined in [RFC6066].

   type: The message type, either heartbeat_request or
      heartbeat_response.

   payload_length: The length of the payload.

   payload: The payload consists of arbitrary content.

   padding: The padding is random content that MUST be ignored by the
      receiver. The length of a HeartbeatMessage is TLSPlaintext.length
      for TLS and DTLSPlaintext.length for DTLS. Furthermore, the
      length of the type field is 1 byte, and the length of the
      payload_length is 2. Therefore, the padding_length is
      TLSPlaintext.length - payload_length - 3 for TLS and
      DTLSPlaintext.length - payload_length - 3 for DTLS. The
      padding_length MUST be at least 16.

   The sender of a HeartbeatMessage MUST use a random padding of at
   least 16 bytes. The padding of a received HeartbeatMessage message
   MUST be ignored.

   If the payload_length of a received HeartbeatMessage is too large,
   the received HeartbeatMessage MUST be discarded silently.

When a HeartbeatRequest message is received and sending a
HeartbeatResponse is not prohibited as described elsewhere in this
document, the receiver MUST send a corresponding HeartbeatResponse
message carrying an exact copy of the payload of the received
HeartbeatRequest.

If a received HeartbeatResponse message does not contain the expected
payload, the message MUST be discarded silently. If it does contain
the expected payload, the retransmission timer MUST be stopped.

5. Use Cases

Each endpoint sends HeartbeatRequest messages at a rate and with the
padding required for the particular use case. The endpoint should
not expect its peer to send HeartbeatRequests. The directions are
independent.

5.1. Path MTU Discovery

DTLS performs path MTU discovery as described in Section 4.1.1.1 of
[RFC6347]. A detailed description of how to perform path MTU
discovery is given in [RFC4821]. The necessary probe packets are the
HeartbeatRequest messages.

This method of using HeartbeatRequest messages for DTLS is similar to
the one for the Stream Control Transmission Protocol (SCTP) using the
padding chunk (PAD-chunk) defined in [RFC4820].

5.2. Liveliness Check

Sending HeartbeatRequest messages allows the sender to make sure that
it can reach the peer and the peer is alive. Even in the case of
TLS/TCP, this allows a check at a much higher rate than the TCP keep-
alive feature would allow.

Besides making sure that the peer is still reachable, sending
HeartbeatRequest messages refreshes the NAT state of all involved
NATs.

HeartbeatRequest messages SHOULD only be sent after an idle period
that is at least multiple round-trip times long. This idle period
SHOULD be configurable up to a period of multiple minutes and down to
a period of one second. A default value for the idle period SHOULD
be configurable, but it SHOULD also be tunable on a per-peer basis.

6. IANA Considerations

   IANA has assigned the heartbeat content type (24) from the "TLS
   ContentType Registry" as specified in [RFC5246]. The reference is to
   RFC 6520.

   IANA has created and now maintains a new registry for Heartbeat
   Message Types. The message types are numbers in the range from 0 to
   255 (decimal). IANA has assigned the heartbeat_request (1) and the
   heartbeat_response (2) message types. The values 0 and 255 should be
   reserved. This registry uses the Expert Review policy as described
   in [RFC5226]. The reference is to RFC 6520.

   IANA has assigned the heartbeat extension type (15) from the TLS
   "ExtensionType Values" registry as specified in [RFC5246]. The
   reference is to RFC 6520.

   IANA has created and now maintains a new registry for Heartbeat
   Modes. The modes are numbers in the range from 0 to 255 (decimal).
   IANA has assigned the peer_allowed_to_send (1) and the
   peer_not_allowed_to_send (2) modes. The values 0 and 255 should be
   reserved. This registry uses the Expert Review policy as described
   in [RFC5226]. The reference is to RFC 6520.

7. Security Considerations

   The security considerations of [RFC5246] and [RFC6347] apply to this
   document. This document does not introduce any new security
   considerations.

8. Acknowledgments

   The authors wish to thank Pasi Eronen, Adrian Farrel, Stephen
   Farrell, Adam Langley, Nikos Mavrogiannopoulos, Tom Petch, Eric
   Rescorla, Peter Saint-Andre, and Juho Vaehae-Herttua for their
   invaluable comments.

9. References

9.1. Normative References

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119, March 1997.

   [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an
              IANA Considerations Section in RFCs", BCP 26, RFC 5226,
              May 2008.

   [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
              (TLS) Protocol Version 1.2", RFC 5246, August 2008.

   [RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions:
              Extension Definitions", RFC 6066, January 2011.

   [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer
              Security Version 1.2", RFC 6347, January 2012.

9.2. Informative References

   [RFC3436] Jungmaier, A., Rescorla, E., and M. Tuexen, "Transport
              Layer Security over Stream Control Transmission Protocol",
              RFC 3436, December 2002.

   [RFC4820] Tuexen, M., Stewart, R., and P. Lei, "Padding Chunk and
              Parameter for the Stream Control Transmission Protocol
              (SCTP)", RFC 4820, March 2007.

   [RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU
              Discovery", RFC 4821, March 2007.

   [RFC5238] Phelan, T., "Datagram Transport Layer Security (DTLS) over
              the Datagram Congestion Control Protocol (DCCP)",
              RFC 5238, May 2008.

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

Authors' Addresses

    Robin Seggelmann
    Muenster University of Applied Sciences
    Stegerwaldstr. 39
    48565 Steinfurt
    DE

    EMail: seggelmann@fh-muenster.de

    Michael Tuexen
    Muenster University of Applied Sciences
    Stegerwaldstr. 39
    48565 Steinfurt
    DE

    EMail: tuexen@fh-muenster.de

    Michael Glenn Williams
    GWhiz Arts & Sciences
    2885 Denise Court
    Newbury Park, CA, 91320
    USA

    EMail: michael.glenn.williams@gmail.com