Design and Implementation of SCTP-aware DTLS

R. Seggelmann[1], M. Tüxen[2] and E. Rathgeb[3]

[1]Münster University of Applied Sciences, Steinfurt, Germany - seggelmann@fh-muenster.de
[2]Münster University of Applied Sciences, Steinfurt, Germany - tuexen@fh-muenster.de
[3]University of Duisburg-Essen, Essen, Germany - erwin.rathgeb@iem.uni-due.de

Abstract

There is currently no widely accepted and deployed approach to provide security for the Stream Control Transmission Protocol (SCTP). A promising new approach is to use the Datagram Transport Layer Security (DTLS) protocol, adding an encryption and authentication layer between transport and application protocol. The adaption for SCTP supports all features without compromising security or requiring major changes in already standardized protocols. Therefore, this solution has been introduced into the Internet Engineering Task Force (IETF) standardization by the authors. This paper analyses existing solutions to outline the requirements and how SCTP-aware DTLS has to be designed in detail to meet them. Performance measurements show that an SCTP association secured by using the modified DTLS provides almost the performance of a TCP connection secured with TLS.

Keywords

Network, Security, Protocol design

1 Introduction

The Stream Control Transmission Protocol (SCTP) [9] was originally designed to transport telephony signaling data via IP networks, but has evolved into a transport protocol for generic use. Since a security solution like TLS has been developed only for TCP, it has several drawbacks when used with SCTP. The use is therefore limited to scenarios, where distinctive new SCTP features like unordered delivery or partially reliable transfer cannot be used.

A promising approach is to adapt Datagram Transport Layer Security (DTLS) [7] for the use with SCTP. The basic idea of this solution has been introduced in [2]. This paper provides a structured analysis of the requirements and identifies the modifications required for an actual implementation of the concept.

The following two sections of this paper provide an overview of SCTP and DTLS. In section 4 an analysis of the features of existing solutions results in the requirements for a new approach, which is presented in section 5. After describing how an implementation of the proposed solution has been tested, performance measurements are discussed in section 6.

2 Stream Control Transmission Protocol

SCTP is a reliable and message oriented transport protocol. The connection between two hosts, called association, is established with a four-way handshake. Furthermore, SCTP supports multihoming, i.e. it can use more than one address per association for fast failover. An SCTP packet consists of a `Common Header` and chunks, providing an extensible packet format. Multiple smaller chunks can be bundled within a packet. DATA chunks carry user data and control chunks are used to transfer SCTP related control information between the SCTP endpoints, for example for association setup and teardown.

Within an association, multiple unidirectional streams can be used to reduce head-of-line blocking, because the sequence of messages is only assured within each single stream. The application assigns a `Stream Identifier` (SID) to each message, indicating which stream

it belongs to. The order of messages within a stream is maintained with a `Stream Sequence Number` (SSN), which is assigned by SCTP before the message is sent. However, messages within streams can also be sent unordered. This is done by setting the U-bit in the DATA chunk header, which causes the receiver to ignore the SSN.

Additional features can be added by defining new chunk and parameter types, such as SCTP-AUTH [11] which allows to protect the integrity of certain chunks. An HMAC is calculated of the chunks the user has chosen to be protected and a new chunk is placed before them in the packet to carry this HMAC. Also available as an extension is partially reliable transfer, PR-SCTP [8]. This extension adds the new FORWARD-TSN chunk, which allows the sender to notify the receiver to ignore the loss of certain messages. The sender can use different policies to decide when not to transmit or retransmit a message. These policies might be limited lifetime, limited retransmissions, priorities and others.

3 Datagram Transport Layer Security

DTLS is a modification of Transport Layer Security (TLS) and basically has the same structure. The Record protocol is the base layer on top of the transport protocol. It transmits the TLS protocol version, the length and the type of its payload. This can either be application data or the Handshake, ChangeCipherSpec or Alert protocol. The Handshake protocol negotiates encryption and compression parameters, while the ChangeCipherSpec protocol is a single message to announce the use of new key material. Errors can be reported with the Alert protocol.

DTLS has the same handshake sequence like TCP. However, as a new security feature of DTLS, the server can decide to send a `HelloVerifyRequest` message in response to a `ClientHello` first. It contains a cookie of arbitrary content, for example a keyed hash of the client's IP address and other parameters. The client has to repeat its `ClientHello` message with the cookie attached. After verifying the cookie, the server can be sure that the sender can receive packets at its claimed address and therefore did not spoof its source address. This mechanism helps to avoid flooding attacks.

DTLS has been developed with the intention to make as few changes to TLS as possible. The main issues with TLS over an unreliable transport protocol are reordering and loss of messages. Since TLS relies on messages arriving reliable and in sequence, it otherwise assumes an error or attack and simply drops the connection. To support unordered reception, the cipher epoch and sequence number of the record, both used for hash calculations of the payload, have been added to the Record protocol to keep them synchronized. Every record message has a unique sequence number and the cipher epoch identifies the key material currently used and is increased with every successful handshake. Although the loss of data is acceptable for application data, handshake messages have to be reliable, otherwise a handshake may not be completed. Therefore, DTLS provides retransmission timers and message fragmentation for handshake messages. It also keeps track of the sequence numbers to restore the order of handshake messages and discard duplicate or delayed messages.

The application data transferred with DTLS is not assured with the reliability methods of the handshake. Because DTLS is designed to work with unreliable transport protocols, it may drop packets. This occurs not only when messages have been modified or are corrupt, but also when they arrive after a renegotiation and belong to an older epoch. With these modifications, DTLS is suitable for almost every transport protocol. However, DTLS preserves message boundaries, which limits the possible message size to the 2^{14} bytes [5], [7].

4 Motivation

4.1 TCP, UDP and DCCP Security

A common method to secure TCP connections is TLS. It provides encryption, authentication and integrity by inserting an additional layer between the transport and application protocols. Since this does not protect the transport protocol itself, an attacker can still modify TCP headers or intercept packets. To assure TCP's reliable and in order delivery, TLS prevents those attacks by maintaining an internal sequence number for each record message, which is used for hash calculations. If packets are removed or their order changed by tampering with TCP's sequence numbers, the hash comparison will fail and the connection is dropped.

Datagram Transport Layer Security (DTLS) [7] is the equivalent of TLS for UDP. It is basically a modification of TLS for unreliable transport protocols, therefore it offers the same features but does not assure or even require reliable and ordered transfer. However, message boundaries are preserved.

DTLS is in [6] also specified for the Datagram Congestion Control Protocol (DCCP), which is a message- and connection-oriented but unreliable transport protocol. It uses congestion control mechanisms for network overload protection and thus can delay a packet because of a congested network. Therefore it needs specific mechanisms for DTLS to avoid too early retransmissions of handshake messages.

4.2 Requirements

The need for encryption, authentication and integrity for the payload is obvious, but SCTP is also a reliable transport protocol, so the reliability should be assured as well. Contrary to TCP, SCTP does not necessarily retain the order of all messages. They can be unordered across streams or can even be sent unordered explicitly. This means the solution has to assure the order of messages sent in order but also be able to handle unordered messages. Like DCCP, SCTP uses a congestion control to protect the network, so the possibility of unnecessary retransmissions overloading the network has to be considered as well. The solution also has to allow the usage of any other SCTP feature, like multihoming, streams and the various extensions.

Additionally, it has to be avoided to modify the SCTP protocol itself, since this would require an elaborate standardization process and the corresponding adaption of existing implementations, which would make a wide deployment much more difficult. The same applies to the encryption layer, which preferably relies on existing work to also prevent an elaborate standardization and a new and therefore error-prone implementation.

4.3 Existing Solutions

There already are existing approaches to provide encryption and authentication for SCTP. The first and obvious one is just to use TLS with SCTP. Basically this is possible, but a feature of TLS is to detect reordered or lost messages and assume a potential attack since TCP should have avoided that. SCTP does not maintain the order across different streams, so either the transfer has to be limited to a single stream or a TLS connection has to be negotiated for pairs of unidirectional streams as described in [3], which results in high overhead. Other features of SCTP, like unordered or partially reliable (PR-SCTP) transfer cannot be used at all.

The Datagram Transport Layer Security (DTLS) [7] for unreliable protocols is also a possible candidate. It does not have any issues with reordering and message loss, so streams, unordered transfer and PR-SCTP can be used with a single DTLS connection per association. However, DTLS does not assure the order of messages at all and since it does not protect SCTP control chunks or the header of the DATA chunk, an attacker can easily modify the order or

drop packets entirely. DTLS also discards messages that it cannot process at the very moment, which is not compatible with the reliability of SCTP.

Other solutions are Secure SCTP [1] and Secure Socket SCTP [4] which have working concepts but require an elaborate and hardly to realize implementation and standardization.

5 Open Issues and Proposed Solutions

With the requirements identified above, it is now possible to design a solution which meets all of them. The most promising candidate to start with is DTLS as proposed in [2], since it is already standardized and does not prevent any SCTP features from being used. However, it has some issues with reliability and providing the required security features, like the assurance of the order within streams.

5.1 Encryption and Authentication

DTLS is an adaption of TLS, so it provides the same encryption, authentication and compression features. An HMAC is calculated of the compressed application data and both are encrypted and sent as the payload of the transport protocol, which itself is unprotected. The DATA chunks of SCTP contain more information than an TCP packet, that is the stream information and the `Payload Protocol Identifier` (PPI). This data can remain unencrypted, because an attacker cannot use this information to presume the application data and otherwise the encryption would have to be realized as part of SCTP in the kernel. However, in [2] is suggested to use SCTP-AUTH to ensure the integrity of control chunks and DATA chunk headers. This is discussed in the following section.

5.2 Assurance of Order and Reliability

TLS assures the order and reliability of TCP by maintaining internal sequence numbers for every record and using them for hash calculations. This approach cannot be used with SCTP because of streams and the possibility to send messages unordered. However, with DTLS these sequence numbers are part of the record header to allow hash calculation independently of the order of the messages, so no assurance is done at all anyway. An attacker could tamper with SCTP's sequence numbers to manipulate the order of messages.

In [2] is suggested to use SCTP-AUTH for DATA, SACK and FORWARD-TSN chunks to ensure their integrity since DTLS cannot protect any parts of the transport protocol. Unfortunately there is no reasoning why these chunks have to be protected. The DATA chunk is obvious, because an attacker must not be able to modify any sequence numbers or stream information to change the order or assigned stream of a message. Other chunks do not have to be protected because their modification would not have an impact on the content to be transmitted.

5.3 Message Loss Prevention

Except for handshakes, DTLS does not rely on reliable transfer and can discard out of order messages. This is no problem with the unreliable UDP, but with SCTP a reliable service is expected. To avoid message loss, certain situations require in order delivery, e.g. when application data might arrive after the epoch already changed and therefore the message became invalid.

Although SCTP offers in sequence transfer, this only applies to messages within a stream. Messages of different streams may be reordered and can still arrive belated. To allow the use of multiple streams anyway, the message sequence has to be enforced across streams in such situations. That is basically assuring that all previous messages have been received and continue on a single stream until application data transfer is transferred normal again.

Maintaining the message sequence across all streams can be achieved by using SCTP's sender dry event notification. This notification occurs as soon as every sent message has been acknowledged and there are no further messages pending or still in flight. Every time before protection against message loss is required, the sender dry event can be used to wait until there is no data left on any stream. Then sensitive messages can be sent on one stream only and the order is retained by SCTP. This assures in sequence transfer, even if multiple streams are used.

Unfortunately this does not assure the in sequence reception of the messages for the application. There still can be messages of other streams in the receive buffer which have not been read by the application when the first message after the sender dry event arrives. It may occur that the SCTP stack passes the latest message arrived to the application first, for example because it cycles through streams during delivery, so the messages are reordered again. Whenever a sender dry event is awaited on the sender side, the receiver has to read everything from the socket to empty the receive buffer first.

5.4 Renegotiations

A handshake can also be performed for an already established connection to renegotiate key material and cipher suite. This also changes the key for SCTP-AUTH which is extracted from every new key, except for the initial handshake which is entirely unprotected. As distinct from the initial handshake, application data has already been sent when renegotiating. This is critical when multiple streams are used, since data across different streams is likely to be unordered. If application data arrives after a `ChangeCipherSpec` message, the key for SCTP-AUTH already changed and the packet gets dropped because its HMAC does not match anymore.

To avoid such a message loss, it has to be assured that there is no application data in flight when changing keys. Therefore, before sending the `ChangeCipherSpec` message, a sender dry event should be awaited. For the client this is after sending the `CertificateVerify` message and for the server before sending the `ServerHelloDone` message, because after that the client continues and the server's next message is then the `ChangeCipherSpec`. To prevent application data passing the `ChangeCipherSpec` in the socket, the server has to read all pending messages from its buffer after the `CertificateVerify` and the client after the `ServerHelloDone` before continuing.

The server concludes the handshake with the `Finished` message and resumes sending application data. Unfortunately it cannot start immediately, because different streams can cause the application data to pass the `Finished` message, which is a protocol violation. The client may then discard the application data or drop the entire connection, which has to be avoided.

The server should also wait for a dry event after sending the `Finished` message, to assure the client has received it before continuing with application data. Since there is also still is a small chance that application data passes the `Finished` message in the receive buffer of the client, the client has to buffer all application data arriving between `ChangeCipherSpec` and `Finished` messages and process it after the `Finished` message has been read and thus the handshake is completed. The client cannot just read all data after the `ChangeCipherSpec` message, because this would be a security risk. If the client would pass all messages to the application without having seen the `Finished` message, an attacker could intercept the server's `Finished` message and application data can be sent without having the handshake verified with the hash value of the `Finished` message.

5.5 Shutdown

To gracefully shut down a DTLS connection, `CloseNotify` alerts are used. A peer announces that it finished sending data with the `CloseNotify` alert, but continues reading. The connection is shut down and the sockets can be closed once the other peer also confirmed to have

finished sending. After receiving a `CloseNotify`, a peer will discard every following message arriving. This can lead to data loss when shutting down a connection. Application data arriving after the alert message will be dropped. This has to be avoided at first with awaiting a sender dry event before sending a `CloseNotify` alert and at second at the receiver side, by processing all pending messages, which still could be read after the `CloseNotify` alert, before changing the state to '`CloseNotify received`'.

5.6 Session Resumption

DTLS supports session resumption like TLS, that is using the already negotiated parameters from an earlier connection. If the server still recognizes the client's identifier, an abbreviated handshake can be performed and the cipher suite parameters of the former connection are used further on without negotiation.

This kind of handshake is not only shorter than the normal full handshake, the sequence of messages also differs in such a way that the client sends the last `Finished` message. Hence, the precautions against losing data have to be changed. The client has to await a dry event notification before sending application data after the `Finished` message in this case, while the server has to buffer messages between `ChangeCipherSpec` and `Finished` messages.

This is sufficient if the abbreviated handshake is done for session resumption. If it is done just to refresh the key material while the connection remains established, more measures have to be taken. Application data has been sent before, which has to be kept from getting discarded because of arriving after the `ChangeCipherSpec` message. Before the client sends the `ClientHello` message, a sender dry event has to be awaited to assure no application data is in flight anymore. The server has to read all pending messages from the receive buffer and has to await a sender dry event before answering with the `ServerHello` message. Because the `ServerHello` message is followed by the `ChangeCipherSpec` and `Finished` messages, the client has to read all pending application data before both handshake messages can be processed.

5.7 Generic Adaptations

SCTP already performs retransmission and replay checks, so these have to be deactivated in DTLS. Otherwise a lost message may be retransmitted twice, which causes unnecessary load on the network. This also solves the problem that DTLS may trigger a retransmission despite the message is only delayed because of a network congestion. Fragmentation is also provided by SCTP, so DTLS can make use of it and does not need to discover the Path MTU. This can easily be achieved by increasing the maximum message size for DTLS to 2^{14} Bytes which is equal to the limitation of the record length caused by the encryption algorithms of TLS. Handshake messages are mostly smaller, so the fragmentation of DTLS will rarely be used.

5.8 Extensions

SCTP extensions also have to be considered for a solution with DTLS, in case they can pose a security risk. The PR-SCTP extension, which limits the reliability, can be a possible target for attacks. An attacker could drop a message by intercepting it and sending a fake FORWARD-TSN chunk to the receiver, so it ignores the loss, and a fake SACK to the sender, so it does not retransmit. Therefore the FORWARD-TSN chunk has to be protected and since this already makes a message drop attack impossible, the protection of the SACK is not necessary.

Other extensions not affecting the data transfer can be used in combination with DTLS. This includes the dynamic address reconfiguration and the stream reset extension.

6 Implementation and Evaluation

6.1 Implementation

The OpenSSL toolkit contains the most advanced open source implementation of DTLS for UDP. Therefore, we decided to base the prototype implementation of SCTP-aware DTLS on it to avoid starting from scratch. The implementations of both plain and SCTP-aware DTLS have been tested with several test applications. It is also used by an IPFIX implementation.

6.2 Measurement Setup

To measure the performance of SCTP-aware DTLS, a series of measurement has been made. The setup consisted of two identical hosts running FreeBSD 8.0 connected with GigE links over a switch and an MTU of 9000 Bytes. Each host was equipped with a Core 2 Duo 3.3 GHz CPU. In each measurement one host sent as many AES-256 encrypted messages as possible while the other received them and recorded the throughput. The duration of each measurements was always 30 seconds and the length of the messages was constant. These measurements have been chosen to determine the impact of DTLS on the performance of the hosts, that is how the additional computing power necessary for the security features slows them down.

6.3 Measurement Results

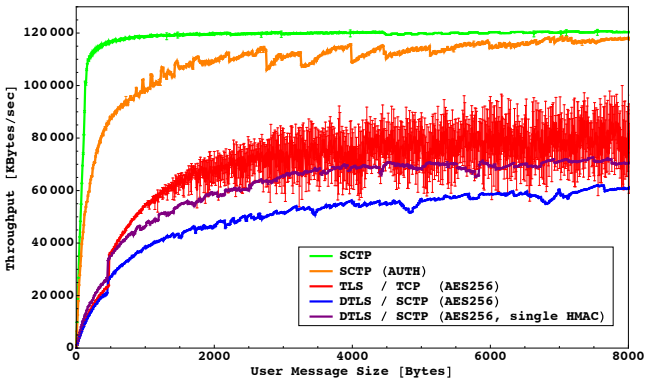

Figure 1: SCTP-aware DTLS and TLS / TCP (dual core)

A standard (unsecured) SCTP connection was measured as a reference, illustrated topmost in Figure 1. Modern CPUs have enough computing power to fully load the link, which limits the throughput. The next step was to activate SCTP-AUTH for DATA chunks to determine the impact on the ¡¡¡¡¡¡¡ dtls-sctp.tex performance to ensure the integrity. The second graph shows that the use of SCTP-AUTH barely reduces the throughput. A fully secured DTLS connection has about half as much throughput as SCTP with SCTP-AUTH.

The increase at 500 bytes is because OpenSSL uses a different AES-256 implementation for smaller messages. The systematic drops at specific message lengths are due to SCTP's message orientation and bundling feature. They indicate when SCTP could bundle fewer messages in a packet because of the increased message size, resulting in smaller packets and more overhead.

For comparison, TLS / TCP was also measured, which is slightly faster than DTLS / SCTP.

6.4 Avoiding Duplicate HMACs

An optimization is to define cipher suites with no HMAC for DTLS to avoid duplicate HMAC calculations. DTLS does not need to calculate its own HMAC to ensure integrity, since this is

done by SCTP-AUTH anyway. Although the HMAC is now calculated after encryption and is not encrypted itself anymore, this does not affect security because an attacker still cannot alter any data without recalculating the HMAC, which is impossible without knowing the secret key. The second lowest graph shows that this optimization is almost as fast a TLS over TCP.

7 Conclusion and Outlook

In this paper the design and implementation of SCTP-aware DTLS is described in detail extending the basic idea in [2] with respect to some crucial aspects, e.g. regarding message loss prevention. The change of key material and association shutdowns are critical with respect to data loss because SCTP messages on different streams do not have to be kept in sequence. The suggested solution uses SCTP sender dry event notifications to drain the data transfer.

Performance measurements proved that an optimized SCTP-aware DTLS can be almost competitive to TLS over TCP. The presented solution based on DTLS allows to use encryption and authentication with SCTP features fully supported without security issues but with a reasonable performance.

Our prototype implementation is based on the DTLS implementation of OpenSSL, and it is targeted to be included in an upcoming official OpenSSL release. Our future work will be to standardize this solution in the IETF [10] and to analyze how rekeying has an impact on the performance and possible optimizations for renegotiations.

References

[1] U. Esbold, E. P. Rathgeb, and A. Jungmaier. Secure SCTP: A versatile secure transport protocol. *Telecommunication Systems*, 27(2–4):273–296, 2004.

[2] C. Hohendorf, E. P. Rathgeb, E. Unurkhaan, and M. Tüxen. Secure end-to-end transport over SCTP. *JCP*, 2(4):31–40, 2007.

[3] A. Jungmaier, E. Rescorla, and M. Tüxen. Transport Layer Security over Stream Control Transmission Protocol. *RFC 3436*, December 2002.

[4] S. Lindskog and A. Brunstrom. An end-to-end security solution for sctp. In *ARES '08: Proceedings of the 2008 Third International Conference on Availability, Reliability and Security*, pages 526–531, Washington, DC, USA, 2008. IEEE Computer Society.

[5] N. Modadugu and E. Rescorla. The Design and Implementation of Datagram TLS. In *In Proc. NDSS*, 2004.

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

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

[8] M. Stewart, R. Ramalho, Q. Xie, M. Tüxen, and P. Conrad. Stream control transmission protocol (SCTP) Partial Reliability Extension. *RFC 3758*, May 2004.

[9] R. Stewart. Stream Control Transmission Protocol. *RFC 4960*, September 2007.

[10] M. Tüxen, R. Seggelmann, and E. Rescorla. Datagram Transport Layer Security for Stream Control Transmission Protocol. *IETF draft-ietf-tsvwg-dtls-for-sctp-04 (work in progress)*, February 2010.

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