TLS Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2018

M. Tiloca
L. Seitz
RISE SICS AB
M. Hoeve
ENCS
O. Bergmann
Universitaet Bremen TZI
March 05, 2018

Extension for protecting (D)TLS handshakes against Denial of Service
draft-tiloca-tls-dos-handshake-02

Abstract

This document describes an extension for TLS and DTLS to protect the server from Denial of Service attacks against the handshake protocol, carried out by an on-path adversary. The extension includes a nonce and a Message Authentication Code (MAC) over that nonce, encoded as a Handshake Token that a Trust Anchor entity computes and provides to the client. The server registered at the Trust Anchor verifies the MAC to determine whether continuing or aborting the handshake.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Servers running TLS [RFC5246][I-D.ietf-tls-tls13] and DTLS [RFC6347][I-D.ietf-tls-dtls13] are vulnerable to Denial of Service (DoS) attacks during the very first step of the handshake protocol. That is, an adversary can repeatedly send ClientHello messages to the server and induce it to perform computations and execute handshakes, before stopping handshake executions and make the server hold state open.

DTLS 1.2 as well as both TLS 1.3 and DTLS 1.3 provide the optional Cookie exchange as possible solution to mitigate this DoS attack. This mechanism is specifically oriented towards adversaries that are not on-path. That is, the Cookie exchange makes the attack more complicated to mount. However, a well determined and resourceful on-path adversary, able to spoof valid IP addresses, can still

successfully perform the DoS attack, by intercepting the possible server response including the Cookie and then echoing it in the second ClientHello. This is in particular possible if the handshake does not use Pre-Shared Key exchange modes.

More specifically, the handshake protocol is exposed to DoS attacks mounted by an on-path adversary, ranging minimally from a man-on-the-side (i.e. able to read and inject traffic, but not block) to maximally a full active adversary (i.e. able also to block traffic).

Depending on the specific protocol version and the key establishment mode used in the handshake, the attack impact can range from a single reply triggered by invalid ClientHello messages, to the server performing advanced handshake steps with consequent setup of invalid half-open sessions. Especially if performed in a large-scale and distributed manner, this attack can thwart performance and service availability of (D)TLS servers. Moreover, the attack can be particularly effective in application scenarios where servers are resource-constrained devices running DTLS over low-power, low bandwidth and lossy networks.

This specification describes a "dos_protection" extension for TLS and DTLS, included into ClientHello messages in order to mark them as valid and neutralize the DoS attacks mentioned above. In essence, the "dos_protection" extension includes a Handshake Token encoding a nonce and a Message Authentication Code (MAC) computed over that nonce. Upon receiving the ClientHello message, the server checks the MAC conveyed in the Handshake Token, and determines whether to either continue the handshake or to immediately abort it.

The proposed method relies on a Trust Anchor (TA) entity, which is in a trust relation with the server, and authorizes the client to establish a secure session with the server. In particular, the Trust Anchor computes the MAC encoded in the Handshake Token, before providing the latter to the client.

1.1. Terminology

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.

Readers are expected to be familiar with terms and concepts related to TLS 1.2 [RFC5246] and DTLS 1.2 [RFC6347], as well as to TLS 1.3 [I-D.ietf-tls-tls13] and DTLS 1.3 [I-D.ietf-tls-dtls13], with particular reference to their respective handshake protocol.

This document refers also to the following terminology.

- Trust Anchor (TA): a trusted third party with a security association with the (D)TLS server. Compared to each single (D)TLS server it is associated to, the Trust Anchor is usually equipped with significant larger amounts of resources, especially in terms of computing power and memory availability.

- Master Key (K_M): a long-term symmetric key shared between the Trust Anchor and the server.

- Handshake Token (T): piece of information provided by the Trust Anchor to a client intending to start a handshake with the server. The Handshake Token is opaque to the client, i.e. the semantics of the Handshake Token are intelligible only to the Trust Anchor and the server.

- Nonce (N): an unsigned integer value used by the Trust Anchor to produce a fresh Handshake Token. The Trust Anchor maintains a pairwise counter separately for each associated server, in order to produce Nonce values.

2. DoS Protection Extension

2.1. Extension Type

This specification extends the ExtensionType enum as follows:

```
   enum {
       ...,
       dos_protection(TBD),
       (65535)
   } ExtensionType;
```

2.2. Extension Data

The "extension_data" field of the "dos_protection" extension contains the following information:

```
    struct {
      opaque handshake_token;
    } extension_data_content;
```

The "handshake_token" field is intended to include the Handshake Token generated by the Trust Anchor. The Handshake Token encodes a nonce and a Message Authentication Code (MAC) computed over the nonce.

3. Protocol overview

Before becoming fully operational, the server S registers at the TA through a secure communication channel or other out-of-band means. A server is registered at one TA only, while the same TA can be associated to multiple servers.

For each registered server S, the TA and S maintain a pairwise counter z_S, associated to that server and encoded as an unsigned integer. Upon S's registration, S and the TA initialize z_S to 0 and establish a long-term symmetric key K_M. The specific means to establish K_M are out of the scope of this specification.

The rest of this document refers to H as a hash function and to an HMAC [RFC2104] relying on H. The TA and the server MUST support the hash function SHA-256.

Figure 1 shows the messages exchanged between the client (C), the Trust Anchor (TA) and the server (S).

```
           C                                   TA                   S
           |                                   |                    |
           |                                   | { Shared key K_M } |
           |                                   |                    |
           | --- Request handshake with S ---> |                    |
     (1)   |                                   |                    |
           | <------- Handshake Token -------- |                    |
           |                                   |                    |
     ---   |                                   |                    |
           |                                                        |
           |      ClientHello with "dos_protection" extension       |
     (2)   | -----------------------------------------------------> |
           |             Including the Handshake Token              |
           |                                                        |
     ---   |                                   |                    |
           |                                   |                    |
           |                                                        |
     (3)   | [<--------------- Next handshake steps -------------->] |
           |                                                        |
           |                                   |                    |
```

Figure 1: Protocol Overview

Step (1) concerns a client C that intends to start a (D)TLS session with the server S. That is, C contacts the TA and specifies its intention to start a (D)TLS handshake with S. The client C can rely on services such as [I-D.ietf-core-resource-directory] to know what is the specific TA associated to S. All communications between C and

the TA must be secured, ensuring integrity, source authentication, confidentiality and replay protection of exchanged messages. The specific means to secure communications between C and the TA are out of the scope of this specification.

The TA must verify that C is authorized to establish a (D)TLS session with S. To this end, the TA can directly authorize the client, or expect the client to upload authorization evidence previously obtained from a trusted entity. Compared with models based on proxies, this approach does not require particular adaptations to the communication between clients and servers. The specific authorization process of clients is out of the scope of this specification.

In case of successful authorization, the TA provides C with a fresh Handshake Token, which encodes a nonce as well as a Message Authentication Code (MAC) computed over the nonce using the key K_M. The Handshake Token is opaque to the client. Besides, the client must consume this Handshake Token right away, and in particular before asking the TA for a new Handshake Token intended for the same server S.

During Step (2), C prepares the ClientHello message addressed to S, including the "dos_protection" extension defined in Section 2. In particular, the extension includes the Handshake Token received by the TA, as content of the field "handshake_token". Then, C sends the ClientHello message to S. The overall content and format of the ClientHello message depend on the specific version of (D)TLS.

Upon receiving the ClientHello message, the server S retrieves the Handshake Token from the "dos_protection" extension. Then, S relies on the nonce included in the Handshake Token to check that the ClientHello message is not a replay. After that, S uses the key K_M to recompute the MAC, and checks it against the MAC encoded in the received Handshake Token.

In case the ClientHello message is fresh and the MAC is valid, S continues to Step (3), i.e., it proceeds with the handshake with C. Otherwise, S discards the ClientHello message and aborts the handshake.

4. Client to Trust Anchor

The client C requests from the TA an authorization to open a new (D)TLS session with the server S. That is, this step does not take place if C intends to resume a (D)TLS session previously established with S. Considerations about session resumption are provided in Section 8.

In case of successful authorization, the TA selects the nonce N as the current value of the pairwise counter z_S associated to S. Then, the TA performs the following actions.

1. It sets the variable token_nonce to the nonce N.

2. It computes a MAC as the output of HMAC(K_M, H(token_nonce)).

3. It builds a Handshake Token including token_nonce and the MAC.

After that, the TA provides the Handshake Token to C, and increments the counter z_S by 1.

The TA handles a wrap-around of the counter z_S by renewing the Master Key K_M as described in Section 9.3.

5. Client to Server

This section considers a client C intending to establish a new (D)TLS session with S. Considerations about session resumption are provided in Section 8.

Once it has received the Handshake Token from the TA, the client C must consume it right away, by including it in a ClientHello message addressed to the server S. In particular, the client C must consume this Handshake Token before asking the TA for a new one intended for the same server S. The client C considers the Handshake Token consumed, and hence discards it, once received a valid ServerHello message during the same handshake with the server S.

Furthermore, the client discards a Handshake Token also in case of handshake abortion due to too many retransmissions of a same ClientHello message. In such a case, the client must ask the TA for a new, i.e. fresh, Handshake Token and start over a new handshake with the server S.

When preparing the ClientHello message, the client C proceeds as follows.

1. It builds the "dos_protection" extension defined in Section 2.

2. It includes the Handshake Token received from the TA in the "handshake_token" field of the "dos_protection" extension.

3. It includes the "dos_protection" extension into the ClientHello message, consistently with what is mandated and recommended by the specific version of (D)TLS.

Once the ClientHello message has been completely prepared, C transmits it to S. Note that C retransmits exactly the same "dos_protection" extension from this first ClientHello message, in case it sends a second ClientHello message as a reply to a HelloVerifyRequest in DTLS 1.2 or a HelloRetryRequest in (D)TLS 1.3.

6. Server Processing

This section considers a server S receiving a ClientHello message from C for initiating a new (D)TLS session. Considerations on session resumption are provided in Section 8.

A server MAY require clients to send a valid "dos_protection" extension. A server requiring this MUST respond to a ClientHello lacking a "dos_protection" extension by terminating the handshake, with a "missing_extension" alert if the client has shown support for (D)TLS 1.3, or a "handshake_failure" alert otherwise.

Upon receiving the first ClientHello message from C, the server S retrieves the Handshake Token from the "handshake_token" field of the "dos_protection" extension.

Then, the server S MUST check that the ClientHello message is not a replay. Section 7 of this specification describes a possible method to perform the anti-replay check, based on the nonce encoded in the Handhshake Token. If the ClientHello message is found to be not fresh, then S discards it and terminates the handshake with a "handshake_failure" alert.

If the ClientHello message is found to be fresh, then S performs the following actions.

1. It retrieves token_nonce from the Handshake Token.

2. It computes a MAC as the output of HMAC(K_M, H(token_nonce)).

If the computed MAC differs from the MAC encoded in the Handshake Token, S discards the ClientHello message and terminates the handshake with a "handshake_failure" alert. Otherwise, S continues performing the handshake with C.

7. Replay Protection

This section describes a possible method to perform anti-replay checks on received ClientHello messages, based on the nonce encoded in the Handshake Token as token_nonce.

The server S maintains a sliding window W of size A, as a pair {w, w_b}, where w is an A-bit vector and w_b indicates the current left bound of W. That is, w_b indicates the lowest value that S can accept as the nonce N encoded in the Handshake Token as token_nonce. Upon startup, S sets w_b to 0 and all bits in w to 0.

Upon receiving a ClientHello message for establishing a new (D)TLS session, the server S considers the nonce N encoded in the Handshake Token as token_nonce, and performs the following checks. As an example, the following considers a 32-bit nonce N.

o If N < w_b, then S discards the ClientHello message and terminates the handshake.

o If w_b <= N < min(w_b + A, 2^32), then S defines i = (N - w_b), and checks the i-th bit of vector w. If such bit is set to 1, i.e. the same nonce N has been already used, then S discards the ClientHello message and terminates the handshake. Instead, if such bit is set to 0, then S proceeds with processing the "dos_protection" extension as described in Section 6.

o If (w_b + A) <= N < 2^32, then S proceeds with processing the "dos_protection" extension as described in Section 6.

During this handshake execution, S discards any possible first ClientHello message including the same nonce N encoded in the Handshake Token as token_nonce.

Once the handshake has been successfully completed, S checks whether the condition N >= w_b is still valid. In such a case, S updates the window W as follows.

o If w_b <= N < min(w_b + A, 2^32), then S defines i = (N - w_b) and sets the i-th bit of vector w to 1, so marking N as used. Instead,

o if (w_b + A) <= N < 2^32, then S defines w* = (N - A + 1) and updates vector w as w = w >> (w* - w_b), where '>>' is the unsigned right bit shift operator. After that, S updates w_b as w_b = w*. Finally, S defines i = (N - w_b) and sets the i-th bit of vector w to 1, so marking N as used.

The window size A should be determined based on the expected frequency of new session establishments on the server S. Evidently, the larger the window, the more accurate is the replay protection, but the greater the memory overhead on the server side.

Furthermore, the window size A should take into account the time required for a client to request and get a Handshake Token from the TA, as well as to to deliver it to the (D)TLS server in the ClientHello message. This is necessary in order to avoid that the sliding window advances too fast, and hence that the (D)TLS server discards such ClientHello messages as stale.

8. Session Resumption

In case a client C sends a ClientHello message asking to resume a session, the server S relies on the existing association with C and hence does not need a further assertion of client's validity from the TA. In addition, S can rely on the Client Hello Recording mechanism described in Section 8 of [I-D.ietf-tls-tls13], in order to perform anti-replay checks on ClientHello messages asking for session resumption.

As a consequence, the "dos_protection" extension defined in Section 2 is not strictly necessary in ClientHello messages sent for session resumption.

However, Section 7.4.1.4 of [RFC5246] states that a client asking for session resumption SHOULD send the same extensions as it would if it was not attempting resumption. At the same time, it states that most extensions are relevant only when a new session is initiated, and hence the server would not process them in case of session resumption.

In accordance with such guidelines, a server S can possibly instruct the TA to also provide requesting clients with a small number R of additional Resumption Tokens.

In order to compute each of the Resumption Tokens for a same request from a given client, the TA MUST use the same nonce value N used to compute the Handshake Token (see Section 4). In particular, the TA computes the i-th Resumption Token, 0 <= i < R, as follows.

1. It sets the variable token_nonce to (N + i), where '+' is the concatenate operator.

2. It computes a MAC as the output of HMAC(K_M, H(token_nonce)).

3. It builds the i-th Resumption Token including token_nonce and the MAC.

Finally, the TA provides the requesting client with the Handshake Token and the additional Resumption Tokens. The client MUST use the Handshake Token during a handshake with S for session initiation, as

described in Section 5. The client MUST use the i-th Resumption Token upon attempting the i-th resumption of that session. After it has used all the Resumption Tokens received from the TA, the client must assume that S does not support further resumptions of the same session.

Upon receiving a ClientHello message from C asking to resume a session, the server S verifies the MAC encoded in the Resumption Token as described in Section 6. However, S does not rely on the "dos_protection" extension and the token_nonce in the Resumption Token to perform an anti-replay check.

Further details about session resumption are defined in the (D)TLS specifications of the different respective versions.

9. Security Considerations

This specification does not change the intended security properties of TLS and DTLS. Additional security aspects are discussed below.

9.1. Security Effectiveness

The MAC encoded in the "dos_protection" extension as part of the Handshake Token is computed only over the 'token_nonce' part of the same Handshake Token. That is, a server S can actually assert the validity and freshness of the Handshake Token only, rather than of the whole ClientHello message.

As a consequence, an on-path adversary can intercept ClientHello messages sent by legitimate clients, retrieve the "dos_protection" extension, and then use it inside forged ClientHello messages injected and addressed to the server. However, this practically displays negligible consequences in terms of additional impact on the server, as discussed in the following.

On one hand, a man-on-the-side adversary, namely able to intercept and inject traffic but not block, can, with reasonable effort, exploit the limitation above in order to induce the server to negotiate more expensive cipher suites, which is fair to consider as a weak attack achievement. Furthermore, the injection of such forged ClientHello messages including a stolen "dos_protection" extension is anyway rate limited by the number of legitimate clients and the frequency of their handshake executions.

On the other hand, a full active adversary, namely able to also block traffic, would not even bother to inject forged ClientHello messages including a stolen "dos_protection" extension. In fact, (s)he can more easily let the server process handshake messages from legitimate

clients during handhshake early phases, and later on block specific client messages during handshake advanced phases, so leaving the server with several half-open sessions and open states. Again, this is anyway rate limited by the number of legitimate clients and the frequency of their handshake executions.

9.2. Trust Anchor as Target

Communications between clients and the TA may be secured by means of (D)TLS, with the TA acting as server. In such a case, the TA becomes also a target for the DoS attack addressed in this specification.

On the other hand, TAs are expected to be equipped with plentiful of resources, i.e. in significant larger amounts than each of the associated (D)TLS servers. That is, given a class of adversary targeting a number of (D)TLS servers, the corresponding TA is practically not a feasible target for that adversary.

Besides, while it is infeasible to expect a considerably high number of (resource-contrained) (D)TLS servers to be robust against DoS by construction, it is instead feasible to have relatively few deployed TAs which are able to endure this attack when carried out against them. This might in turn encourage an adversary to rather target a TA, in order to indirectly make the (D)TLS servers unavailable to serve clients. However, as discussed above, a class of adversary targeting a (D)TLS server is not supposed to have sufficient resources to effectively compromise the availability of the corresponding TA.

Furthermore, a typical starting point for an adversary consists in identifying the set of victim servers, as belonging to the same application/administrative domain(s) or network segment(s). Hence, the adversary would be motivated in targeting the TA(s) associated to the (D)TLS servers in those segments. As an additional deterrent, (D)TLS servers in a same segment or domain can thus be registered at different TAs, in order to further reduce the feasibility and spread the effectiveness of attacks rather addressed against those TAs.

9.3. Renewal of Long-Term Key K_M

While it can practically take a long amount of time, the pairwise counter z_S maintained by the TA and associated to S eventually wraps around. When this happens, the TA MUST revoke the key K_M shared with S, in order to not reuse {K_M, N} pairs when building Handshake Tokens for requesting clients.

In particular, when the counter z_S wraps-around, the TA MUST perform the following actions.

1. It stops accepting requests related to S from clients.

2. It securely generates a new long-term key K_M and securely provides it to S.

3. It resumes serving requests related to S from clients, using the new K_M to compute MACs when building Handshake Tokens.

9.4. Rate Limit to Nonce Release

It is RECOMMENDED that the TA does not release Handshake Tokens to clients beyond a maximum rate. This prevents a client with legitimate credentials from quickly consuming the nonce space associated to S, and thus making the TA unable to serve other clients.

10. IANA Considerations

IANA is requested to allocate an entry to the existing TLS "ExtensionType" registry defined in [RFC5246] and originally created in [RFC4366], for dos_protection (TBD) defined in this document.

11. Acknowledgments

The authors are sincerely thankful to Santiago Aragon, Rolf Blom and Eric Rescorla for their comments and feedback.

The work on this document has been partly supported by the EU FP7 project SEGRID (Grant Agreement no. 607109) and the EIT-Digital High Impact Initiative ACTIVE.

12. References

12.1. Normative References

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-22 (work in progress), November 2017.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-24 (work in progress), February 2018.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

[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>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[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>.

12.2. Informative References

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-12 (work in progress), October 2017.

[RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, DOI 10.17487/RFC4366, April 2006, <https://www.rfc-editor.org/info/rfc4366>.

Authors' Addresses

Marco Tiloca
RISE SICS AB
Isafjordsgatan 22
Kista SE-164 29
Sweden

Phone: +46 70 604 65 01
Email: marco.tiloca@ri.se

Ludwig Seitz
RISE SICS AB
Scheelevaegen 17
Lund SE-223 70
Sweden

Phone: +46 70 349 92 51
Email: ludwig.seitz@ri.se

Maarten Hoeve
ENCS
Regulusweg 5
The Hague 2516 AC
The Netherlands

Phone: +31 62 015 75 51
Email: maarten.hoeve@encs.eu

Olaf Bergmann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49 421 218 63904
Email: bergmann@tzi.org