Internet Engineering Task Force (IETF)
Request for Comments: 8449
Updates: 6066
Category: Standards Track
ISSN: 2070-1721

M. Thomson
Mozilla

August 2018

Record Size Limit Extension for TLS

Abstract

An extension to Transport Layer Security (TLS) is defined that allows endpoints to negotiate the maximum size of protected records that each will send the other.

This replaces the maximum fragment length extension defined in RFC 6066.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8449.

Table of Contents

1. Introduction 2
2. Conventions and Definitions 3
3. Limitations of the "max_fragment_length" Extension 3
4. The "record_size_limit" Extension 4
4.1. Record Expansion Limits 6
5. Deprecating "max_fragment_length" 6
6. Security Considerations 7
7. IANA Considerations 7
8. References 7
8.1. Normative References 7
8.2. Informative References 8
Acknowledgments 8
Author's Address 8

1. Introduction

Implementing Transport Layer Security (TLS) [TLS] or Datagram TLS (DTLS) [DTLS] for constrained devices can be challenging. However, recent improvements to the design and implementation of cryptographic algorithms have made TLS accessible to some highly limited devices (see, for example, [RFC7925]).

Receiving large protected records can be particularly difficult for a device with limited operating memory. TLS versions 1.2 [RFC5246] and earlier permit senders to generate records 16384 octets in size, plus any expansion from compression and protection up to 2048 octets (though typically this expansion is only 16 octets). TLS 1.3 reduces the allowance for expansion to 256 octets. Allocating up to 18K of memory for ciphertext is beyond the capacity of some implementations.

An Authentication Encryption with Additional Data (AEAD) cipher (see [RFC5116]) API requires that an entire record be present to decrypt and authenticate it. Similarly, other ciphers cannot produce authenticated data until the entire record is present. Incremental processing of records exposes endpoints to the risk of forged data.

The "max_fragment_length" extension [RFC6066] was designed to enable constrained clients to negotiate a lower record size. However, "max_fragment_length" suffers from several design problems (see Section 3).

This document defines a "record_size_limit" extension (Section 4). This extension replaces "max_fragment_length" [RFC6066], which this document deprecates. This extension is valid in all versions of TLS.

A smaller protected record size is just one of many problems that a constrained implementation might need to address. The "record_size_limit" extension only addresses the memory allocation problem; it does not address limits of code size, processing capability, or bandwidth capacity.

2. Conventions and Definitions

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.

3. Limitations of the "max_fragment_length" Extension

The "max_fragment_length" extension has several limitations that make it unsuitable for use.

A client that has no constraints preventing it from accepting a large record cannot use "max_fragment_length" without risking a reduction in the size of records. The maximum value that the extension permits is 2^12, much smaller than the maximum record size of 2^14 that the protocol permits.

For large data transfers, small record sizes can materially affect performance. Every record incurs additional costs, both in the additional octets for record headers and for expansion due to encryption. Processing more records also adds computational overheads that can be amortized more effectively for larger record sizes. Consequently, clients that are capable of receiving large records could be unwilling to risk reducing performance by offering the extension, especially if the extension is rarely needed.

This would not be an issue if a codepoint were available or could be added for fragments of 2^14 octets. However, RFC 6066 requires that

servers abort the handshake with an "illegal_parameter" alert if they receive the extension with a value they don't understand. This makes it impossible to add new values to the extension without the risk of failed connection attempts.

A server that negotiates "max_fragment_length" is required to echo the value selected by the client. The server cannot request a lower limit than the one the client offered. This is a significant problem if a server is more constrained than the clients it serves.

The "max_fragment_length" extension is also ill-suited to cases where the capabilities of client and server are asymmetric. Constraints on record size are often receiver constraints.

In comparison, an implementation might be able to send data incrementally. Encryption does not have the same atomicity requirement. Some ciphers can be encrypted and sent progressively. Thus, an endpoint might be willing to send records larger than the limit it advertises for records that it receives.

If these disincentives are sufficient to discourage clients from deploying the "max_fragment_length" extension, then constrained servers are unable to limit record sizes.

4. The "record_size_limit" Extension

The ExtensionData of the "record_size_limit" extension is RecordSizeLimit:

```
uint16 RecordSizeLimit;
```

The value of RecordSizeLimit is the maximum size of record in octets that the endpoint is willing to receive. This value is used to limit the size of records that are created when encoding application data and the protected handshake message into records.

When the "record_size_limit" extension is negotiated, an endpoint MUST NOT generate a protected record with plaintext that is larger than the RecordSizeLimit value it receives from its peer.

Unprotected messages are not subject to this limit.

This value is the length of the plaintext of a protected record. The value includes the content type and padding added in TLS 1.3 (that is, the complete length of TLSInnerPlaintext). In TLS 1.2 and earlier, the limit covers all input to compression and encryption (that is, the data that ultimately produces TLSCiphertext.fragment). Padding added as part of encryption, such as that added by a block cipher, is not included in this count (see Section 4.1).

An endpoint that supports all record sizes can include any limit up to the protocol-defined limit for maximum record size. For TLS 1.2 and earlier, that limit is 2^14 octets. TLS 1.3 uses a limit of 2^14+1 octets. Higher values are currently reserved for future versions of the protocol that may allow larger records; an endpoint MUST NOT send a value higher than the protocol-defined maximum record size unless explicitly allowed by such a future version or extension. A server MUST NOT enforce this restriction; a client might advertise a higher limit that is enabled by an extension or version the server

does not understand. A client MAY abort the handshake with an "illegal_parameter" alert if the record_size_limit extension includes a value greater than the maximum record size permitted by the negotiated protocol version and extensions.

Even if a larger record size limit is provided by a peer, an endpoint MUST NOT send records larger than the protocol-defined limit, unless explicitly allowed by a future TLS version or extension.

The record size limit only applies to records sent toward the endpoint that advertises the limit. An endpoint can send records that are larger than the limit it advertises as its own limit. A TLS endpoint that receives a record larger than its advertised limit MUST generate a fatal "record_overflow" alert; a DTLS endpoint that receives a record larger than its advertised limit MAY either generate a fatal "record_overflow" alert or discard the record.

Endpoints SHOULD advertise the "record_size_limit" extension, even if they have no need to limit the size of records. For clients, this allows servers to advertise a limit at their discretion. For servers, this allows clients to know that their limit will be respected. If this extension is not negotiated, endpoints can send

records of any size permitted by the protocol or other negotiated extensions.

Endpoints MUST NOT send a "record_size_limit" extension with a value smaller than 64. An endpoint MUST treat receipt of a smaller value as a fatal error and generate an "illegal_parameter" alert.

In TLS 1.3, the server sends the "record_size_limit" extension in the EncryptedExtensions message.

During renegotiation or resumption, the record size limit is renegotiated. Records are subject to the limits that were set in the handshake that produces the keys that are used to protect those records. This admits the possibility that the extension might not be negotiated when a connection is renegotiated or resumed.

The Path Maximum Transmission Unit (PMTU) in DTLS also limits the size of records. The record size limit does not affect PMTU discovery and SHOULD be set independently. The record size limit is fixed during the handshake and so should be set based on constraints at the endpoint and not based on the current network environment. In comparison, the PMTU is determined by the network path and can change dynamically over time. See [PMTU] and Section 4.1.1.1 of [DTLS] for more detail on PMTU discovery.

PMTU governs the size of UDP datagrams, which limits the size of records, but does not prevent records from being smaller. An endpoint that sends small records is still able to send multiple records in a single UDP datagram.

4.1. Record Expansion Limits

The size limit expressed in the "record_size_limit" extension doesn't account for expansion due to compression or record protection. It is expected that a constrained device will disable compression to avoid unpredictable increases in record size. Stream ciphers and existing AEAD ciphers don't permit variable amounts of expansion, but block ciphers do permit variable expansion.

In TLS 1.2, block ciphers allow from 1 to 256 octets of padding.

When a limit lower than the protocol-defined limit is advertised, a second limit applies to the length of records that use block ciphers. An endpoint MUST NOT add padding to records that would cause the protected record to exceed the size of a protected record that contains the maximum amount of plaintext and the minimum permitted amount of padding.

For example, TLS_RSA_WITH_AES_128_CBC_SHA has 16-octet blocks and a 20-octet MAC. Given a record size limit of 256, a record of that length would require a minimum of 11 octets of padding (for [RFC5246], where the MAC is covered by encryption); or 15 octets if the "encrypt_then_mac" extension [RFC7366] is negotiated. With this limit, a record with 250 octets of plaintext could be padded to the same length by including at most 17 octets of padding, or 21 octets with "encrypt_then_mac".

An implementation that always adds the minimum amount of padding will always comply with this requirement.

5. Deprecating "max_fragment_length"

The "record_size_limit" extension replaces the "max_fragment_length" extension [RFC6066]. A server that supports the "record_size_limit" extension MUST ignore a "max_fragment_length" that appears in a ClientHello if both extensions appear. A client MUST treat receipt of both "max_fragment_length" and "record_size_limit" as a fatal error, and it SHOULD generate an "illegal_parameter" alert.

Clients that depend on having a small record size MAY continue to advertise the "max_fragment_length".

6. Security Considerations

Very small record sizes might generate additional work for senders and receivers, limiting throughput and increasing exposure to denial of service.

7. IANA Considerations

This document registers the "record_size_limit" extension in the "TLS ExtensionType Values" registry established in [RFC5246]. The "record_size_limit" extension has been assigned a code point of 28. The IANA registry [TLS-REGISTRY] lists this extension as "Recommended" (i.e., "Y") and indicates that it may appear in the ClientHello (CH) or EncryptedExtensions (EE) messages in TLS 1.3 [TLS].

In the same registry, the "max_fragment_length" has been changed to not recommended (i.e., "N").

8. References

8.1. Normative References

[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>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[RFC7366] Gutmann, P., "Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7366, DOI 10.17487/RFC7366, September 2014, <https://www.rfc-editor.org/info/rfc7366>.

[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>.

[TLS] Rescorla, E., "The Transport Layer Security (TLS) Protocol

Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

8.2. Informative References

[DTLS] 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>.

[PMTU] McCann, J., Deering, S., Mogul, J., and R. Hinden, Ed., "Path MTU Discovery for IP version 6", STD 87, RFC 8201, DOI 10.17487/RFC8201, July 2017, <https://www.rfc-editor.org/info/rfc8201>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <https://www.rfc-editor.org/info/rfc7925>.

[TLS-REGISTRY] Salowey, J. and S. Turner, "IANA Registry Updates for TLS and DTLS", RFC 8447, DOI 10.17487/RFC8447, August 2018, <https://www.rfc-editor.org/info/rfc8447>.

Acknowledgments

Thomas Pornin and Hannes Tschofenig provided significant input to this document. Alan DeKok identified an issue with the interaction between record size limits and PMTU.

Author's Address

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com