TLS
Internet-Draft
Updates: 6066 (if approved)
Intended status: Standards Track
Expires: September 28, 2017

M. Thomson
Mozilla

March 27, 2017

# Record Size Limit Extension for Transport Layer Security (TLS)
# draft-thomson-tls-record-limit-00

## 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.

## Status of This Memo

## Copyright Notice

Table of Contents

## 1. Introduction

Implementing Transport Layer Security (TLS) [I-D.ietf-tls-tls13] 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 and earlier [RFC5246] 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.

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 that replaces "max_fragment_length" (see Section 4). This extension is valid in all versions of TLS.

## 2. Conventions and Definitions

The words "MUST", "MUST NOT", "SHOULD", and "MAY" are used in this document. It's not shouting; when they are capitalized, they have the special meaning defined in [RFC2119].

## 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 [TARREAU]. 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 risking connection attempts failing.

The "max_fragment_length" extension is also ill-suited to cases where the capabilities of client and server are asymmetric. The server is required to select a fragment length that is as small or smaller than the client offers and both endpoints need to comply with this smaller limit.

Constraints on record size are often receiver constraints. In particular, an Authentication Encryption with Additional Data (AEAD) ciphers (see [RFC5116]) API requires that an entire record be present to decrypt and authenticate it. Some implementations choose not to implement an AEAD interface in this way to avoid this problem, but that exposes them to risks that an AEAD is intended to protect against.

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 more than its receive limit.

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 that the endpoint is willing to receive. 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 - handshake messages in particular - are not subject to this limit.

The size limit value governs 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). Padding added as part of encryption, such as that added by a block cipher, is not included in this count.

An endpoint that supports all record sizes can include any limit up to the protocol-defined limit for maximum record size. For TLS 1.3 and earlier, that limit is 2^14 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.

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 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 and know - and account for - the maximum expansion possible due to record protection based on the cipher suites it offers or selects. Note that up to 256 octets of padding and padding length can be added to block ciphers.

The record size limit only applies to protected records that are sent toward a peer. An endpoint MAY send records that are larger than the limit it advertises.

Clients SHOULD advertise the "record_size_limit" extension, even if they have no need to limit the size of records. This allows servers to apply a limit at their discretion. 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.

## 5. Deprecating "max_fragment_length"

The "record_size_limit" extension replaces the "max_fragment_length" extension. A server that supports the "record_size_limit" extension MUST ignore and "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 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 TBD; it is recommended and marked as "Encrypted" in TLS 1.3.

## 8. References

### 8.1. Normative References

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-19 (work in progress), March 2017.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://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, <http://www.rfc-editor.org/info/rfc5246>.

## 8.2. Informative References

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

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

[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, <http://www.rfc-editor.org/info/rfc7925>.

[TARREAU] Tarreau, W., "Re: Stuck in a train -- reading HTTP/2 draft.", n.d., <https://lists.w3.org/Archives/Public/ietf-http-wg/2014AprJun/1591.html>.

Author's Address

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com