TLS
Internet-Draft
Intended status: Standards Track
Expires: June 12, 2018

A. Ghedini
Cloudflare, Inc.
V. Vasiliev
Google

December 09, 2017

Transport Layer Security (TLS) Certificate Compression
draft-ietf-tls-certificate-compression-01

Abstract

In Transport Layer Security (TLS) handshakes, certificate chains often take up the majority of the bytes transmitted.

This document describes how certificate chains can be compressed to reduce the amount of data transmitted and avoid some round trips.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 12, 2018.

Table of Contents

1. Introduction

In order to reduce latency and improve performance it can be useful to reduce the amount of data exchanged during a Transport Layer Security (TLS) handshake.

[RFC7924] describes a mechanism that allows a client and a server to avoid transmitting certificates already shared in an earlier handshake, but it doesn't help when the client connects to a server for the first time and doesn't already have knowledge of the server's certificate chain.

This document describes a mechanism that would allow certificates to be compressed during full handshakes.

2. Notational Conventions

The words "MUST", "MUST NOT", "SHALL", "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. Negotiating Certificate Compression

This document defines a new extension type (compress_certificates(TBD)), which is used by the client and the

server to negotiate the use of compression for their certificate chains, as well as the choice of the compression algorithm.

By sending the compress_certificates extension, the client indicates to the server the certificate compression algorithms it supports.

The "extension_data" field of this extension in the ClientHello SHALL contain a CertificateCompressionAlgorithms value:

```
enum {
    zlib(0),
    brotli(1),
    (255)
} CertificateCompressionAlgorithm;

struct {
    CertificateCompressionAlgorithm algorithms<1..2^8-1>;
} CertificateCompressionAlgorithms;
```

If the server supports any of the algorithms offered in the ClientHello, it MAY respond with an extension indicating which compression algorithm it chose. In that case, the "extension_data" SHALL be a CertificateCompressionAlgorithm value corresponding to the chosen algorithm. If the server has chosen to not use any compression, it MUST NOT send the compress_certificates extension.

4. Compressed Certificate Message

If a compression algorithm has been negotiated, server and client MAY compress their corresponding Certificate messages and send them in the form of the CompressedCertificate message (replacing the Certificate message).

The CompressedCertificate message is formed as follows:

```
struct {
     uint24 uncompressed_length;
     opaque compressed_certificate_message<1..2^24-1>;
} CompressedCertificate;
```

uncompressed_length The length of the Certificate message once it is uncompressed. If after decompression the specified length does

not match the actual length, the party receiving the invalid message MUST abort the connection with the "bad_certificate" alert.

compressed_certificate_message The compressed body of the Certificate message, in the same format as it would normally be expressed in. The compression algorithm defines how the bytes in the compressed_certificate_message field are converted into the Certificate message.

A peer is not required to compress their own Certificate messages even if the compress_certficates extension has been negotiated, but MUST be able to decompress a received CompressedCertificate message.

If the negotiated compression algorithm is zlib, then the Certificate message MUST be compressed with the ZLIB compression algorithm, as defined in [RFC1950]. If the negotiated compression algorithm is brotli, the Certificate message MUST be compressed with the Brotli compression algorithm as defined in [RFC7932].

If the received CompressedCertificate message cannot be decompressed, the connection MUST be tore down with the "bad_certificate" alert.

If the format of the Certificate message is altered using the server_certificate_type extension [RFC7250], the resulting altered message is compressed instead.

If the server chooses to use the cached_info extension [RFC7924] to replace the Certificate message with a hash, it MUST NOT send the compress_certificates extension.

5. Security Considerations

After decompression, the Certificate message MUST be processed as if it were encoded without being compressed. This way, the parsing and the verification have the same security properties as they would have in TLS normally.

Since certificate chains are typically presented on a per-server name

or per-user basis, the attacker does not have control over any individual fragments in the Certificate message, meaning that they cannot leak information about the certificate by modifying the plaintext.

The implementations SHOULD bound the memory usage when decompressing the CompressedCertificate message.

The implementations MUST limit the size of the resulting decompressed chain to the specified uncompressed length, and they MUST abort the connection if the size exceeds that limit. TLS framing imposes 16777216 byte limit on the certificate message size, and the implementations MAY impose a limit that is lower than that; in both cases, they MUST apply the same limit as if no compression were used.

6. Middlebox Compatibility

It's been observed that a significant number of middleboxes intercept and try to validate the Certificate message exchanged during a TLS handshake. This means that middleboxes that don't understand the CompressedCertificate message might misbehave and drop connections that adopt certificate compression.

However this is not a problem when using TLS version 1.3 [draft-ietf-tls-tls13] and higher, due to the fact that the Certificate (and thus the CompressedCertificate) message is encrypted, preventing middleboxes from intercepting it.

7. IANA Considerations

7.1. Update of the TLS ExtensionType Registry

Create an entry, compress_certificates(TBD), in the existing registry for ExtensionType (defined in [RFC5246]).

7.2. Update of the TLS HandshakeType Registry

Create an entry, compressed_certificate(TBD), in the existing registry for HandshakeType (defined in [RFC5246]).

7.3. Registry for Compression Algorithms

This document establishes a registry of compression algorithms supported for compressing the Certificate message, titled "Certificate Compression Algorithm IDs", under the existing "Transport Layer Security (TLS) Extensions" heading.

The entries in the registry are:

Algorithm Number	Description
0	zlib
1	brotli
224 to 255	Reserved for Private Use

The values in this registry shall be allocated under "IETF Review" policy for values strictly smaller than 64, and under "Specification Required" policy otherwise (see [RFC5226] for the definition of relevant policies).

8. Normative References

[RFC1950] Deutsch, P. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, DOI 10.17487/RFC1950, May 1996, <https://www.rfc-editor.org/info/rfc1950>.

[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>.

[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>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

[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>.

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <https://www.rfc-editor.org/info/rfc7250>.

[RFC7924] Santesson, S. and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension", RFC 7924, DOI 10.17487/RFC7924, July 2016, <https://www.rfc-editor.org/info/rfc7924>.

[RFC7932] Alakuijala, J. and Z. Szabadka, "Brotli Compressed Data Format", RFC 7932, DOI 10.17487/RFC7932, July 2016, <https://www.rfc-editor.org/info/rfc7932>.

Authors' Addresses

Alessandro Ghedini
Cloudflare, Inc.

Email: alessandro@cloudflare.com

Victor Vasiliev
Google

Email: vasilvv@google.com