TLS Working Group
Internet-Draft
Intended status: Standards Track
Expires: 28 December 2020

V. Vasiliev
Google

26 June 2020

TLS Application-Layer Protocol Settings Extension
draft-vvv-tls-alps-00

Abstract

This document describes a Transport Layer Security (TLS) extension for negotiating application-layer protocol settings (ALPS) within the TLS handshake. Any application-layer protocol operating over TLS can use this mechanism to indicate its settings to the peer in parallel with the TLS handshake completion.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the TLS Working Group mailing list (tls@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/tls/ (https://mailarchive.ietf.org/arch/browse/tls/).

Source for this draft and an issue tracker can be found at https://github.com/vasilvv/tls-alps (https://github.com/vasilvv/tls-alps).

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 https://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 28 December 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

An application-layer protocol often starts with both parties negotiating parameters under which the protocol operates; for instance, HTTP/2 [RFC7540] uses a SETTINGS frame to exchange the list of protocol parameters supported by each endpoint. This is usually achieved by waiting for TLS handshake [RFC8446] to complete and then performing the application-layer handshake within the application protocol itself. This approach, despite its apparent simplicity at first, has multiple drawbacks:

1. While the server is technically capable of sending configuration to the peer as soon as it sends its Finished message, most TLS implementations do not allow any application data to be sent until the Finished message is received from the client. This

adds an extra round-trip to the time of when the server settings are available to the client.

2. In QUIC, any settings delivered within the application layer can arrive after other application data; thus, the application has to operate under the assumption that peer's settings are not always available.

3. If the application needs to be aware of the server settings in order to send 0-RTT data, the application has to manually integrate with the TLS stack to associate the settings with TLS session tickets.

This document introduces a new TLS extension, "application_settings", that allows applications to exchange settings within the TLS handshake. Through doing that, the settings can be made available to the application as soon as the handshake completes, and can be associated with TLS session tickets automatically at the TLS layer. This approach allows the application protocol to be designed with the assumption that it has access to the peer's settings whenever it is able to send data.

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. Semantics

Settings are defined to be an opaque blob that is specified by the application when initiating a TLS connection. The settings are meant to be a _declaration_ of the protocol parameters supported by the sender. While in this version of the extension the server settings are always sent first, this may change in future versions; thus, the application MUST NOT vary client settings based on the ones received from the server.

ALPS is _not_ a negotiation mechanism: there is no notion of rejecting peer's settings, and the settings are not responses to one another. Nevertheless, it is possible for parties to coordinate behavior by, for instance, requiring a certain parameter to be present in both client and server settings. This makes ALPS mechanism similar to QUIC transport parameters [I-D.ietf-quic-transport] or HTTP/2 SETTINGS frame [RFC7540], but puts it in contrast to similar mechanisms in TLS.

Settings are exchanged as a part of the TLS handshake that is encrypted with the handshake keys. When the server settings are sent, the identity of the client has not been yet established; therefore, an application MUST NOT use ALPS if it requires the settings to be available only to the authenticated clients.

The ALPS model provides applications with a guarantee that the settings are available before any application data can be written. Note that this implies that when the full handshake is performed, the server can no longer send data immediately after sending its Finished message; it has to wait for the client to respond with its settings. This may negatively impact the latency of the protocols where the server sends the first message, however it should be noted that sending application data before receiving has not been widely supported by TLS implementations, nor has it been allowed in situations when establishing client identity through TLS is required.

ALPS can only be used in conjunction with Application-Layer Protocol Negotiation: the client MUST offer ALPN [RFC7301] if advertising ALPS support, and the server MUST NOT reply with ALPS unless it is also negotiating ALPN. The ALPS payload is protocol-dependent, and as such it MUST be specified with respect to a selected ALPN.

For application protocols that support 0-RTT data, both the client and the server have to remember the settings provided by the both sides during the original connection. If the client sends 0-RTT data and the server accepts it, the ALPS values SHALL be the same values as were during the original connection. In all other cases (including session resumption that does not result in server accepting early data), new ALPS values SHALL be negotiated.

If the client wishes to send different client settings for the 0-RTT session, it MUST NOT offer 0-RTT. Conversely, if the server would send different server settings, it MUST reject 0-RTT. Note that the ALPN itself is similarly required to match the one in the original connection, thus the settings only need to be remembered or checked for a single application protocol.

4. Wire protocol

ALPS is only supported in TLS version 1.3 or later, as the earlier versions do not provide any confidentiality protections for the handshake data. The exchange is performed in three steps:

1. The client sends an extension in ClientHello that enumerates all ALPN values for which ALPS is supported.

2. The server sends an encrypted extension containing the server settings.

3. The client sends a new handshake message containing the client settings.

```
       Client                                           Server

       ClientHello
       + alpn
       + alps                  -------->

                                                   ServerHello
                                         {EncryptedExtensions}
                                                      + {alpn}
                                                      + {alps}
                                                           ...
                               <--------            {Finished}

       {ClientApplicationSettings}
       {Certificate*}
       {CertificateVerify*}
```

```
        {Finished}                 -------->

                           +  Indicates extensions sent in the
                              previously noted message.

                           {} Indicates messages protected using
                              the handshake keys.

                           *  Indicates optional messages that are
                              not related to ALPS.
```

Figure 1: ALPS exchange in a full TLS handshake

A TLS client can enable ALPS by specifying an "application_settings" extension. The value of the "extension_data" field for the ALPS extension SHALL be a ApplicationSettingsSupport struct:

```
struct {
    ProtocolName supported_protocols<2..2^16-1>;
} ApplicationSettingsSupport;
```

Here, the "supported_protocols" field indicates the names of the protocols (as defined in [RFC7301]) for which ALPS exchange is supported; this is necessary for the situations when the client offers multiple ALPN values but only supports ALPS in some of them.

If the server chooses an ALPN value for which the client has offered ALPS support, the server MAY send an "application_settings" extension in the EncryptedExtensions. The value of the "extension_data" field in that case SHALL be an opaque blob containing the server settings as specified by the application protocol.

If the client receives an EncryptedExtensions message containing an "application_settings" extension from the server, after receiving server's Finished message it MUST send a ClientApplicationSettings handshake message before sending the Finished message:

```
enum {
    client_application_settings(TBD), (255)
} HandshakeType;
```

```
struct {
    opaque application_settings<0..2^16-1>;
} ClientApplicationSettings;
```

The value of the "application_settings" field SHALL be an opaque blob containing the client settings as specified by the application protocol. If the client is providing a client certificate, the ClientApplicationSettings message MUST precede the Certificate message sent by the client.

If the ClientApplicationSettings message is sent or received during the handshake, it SHALL be appended to the end of client's Handshake Context context as defined in Section 4.4 of [RFC8446]. In addition, for Post-Handshake Handshake Context, it SHALL be appended after the client Finished message.

When performing session resumption with 0-RTT data, the settings are carried over from the original connection. The server SHALL send an empty "application_settings" extension if it accepts 0-RTT, and the client SHALL NOT send a ClientApplicationSettings message.

5. Security Considerations

ALPS is protected using the handshake keys, which are the secret keys derived as a result of (EC)DHE between the client and the server.

In order to ensure that the ALPS values are authenticated, the TLS implementation MUST NOT reveal the contents of peer's ALPS until peer's Finished message is received, with exception of cases where the ALPS has been carried over from the previous connection.

6. IANA Considerations

IANA will update the "TLS ExtensionType Values" registry to include "application_settings" with the value of TBD; the list of messages in which this extension may appear is "CH, SH".

IANA will also update the "TLS HandshakeType" registry to include "client_application_settings" message with value TBD, and "DTLS-OK" set to "Y".

7. References

7.1. 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>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/info/rfc7301>.

[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>.

[RFC8446] 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>.

7.2. Informative References

[I-D.ietf-quic-transport]
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-29, 9 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-29.txt>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

Acknowledgments

This document has benefited from contributions and suggestions from David Benjamin, Nick Harper, David Schinazi, Renjie Tang and many others.

Author's Address

Victor Vasiliev
Google

Email: vasilvv@google.com