TCPING
Internet-Draft
Intended status: Standards Track
Expires: November 12, 2015

E. Rescorla
Mozilla

May 11, 2015

TCP Use TLS Option
draft-rescorla-tcpinc-tls-option-02

Abstract

This document defines a TCP option (TCP-TLS) to indicate that TLS should be negotiated on a given TCP connection.

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 November 12, 2015.

Table of Contents

1. Introduction 2
2. Overview 3
3. Extension Definition 3
4. Transport Integrity 5
5. Implementation Options 5
6. TLS Profile 6
7. Channel Bindings 6
8. NAT/Firewall considerations 6
9. IANA Considerations 6
10. Security Considerations 7
11. References 7
11.1. Normative References 7
11.2. Informative References 8

1. Introduction

RFC EDITOR: PLEASE REMOVE THE FOLLOWING PARAGRAPH The source for this draft is maintained in GitHub. Suggested changes should be submitted as pull requests at https://github.com/ekr/tcpinc-tls. Instructions are on that page as well.

The TCPINC WG is chartered to define protocols to provide ubiquitous, transparent security for TCP connections.

While TLS [RFC5246] is by far the most popular mechanism for securing TCP data, adding it to a given protocol requires some sort of coordination; if a client just tries to initiate TLS with a non-TLS server, the server will most likely reject the protocol messages because they do not conform to its expectations for the application layer protocol. This coordination can take a number of forms, including:

- An external signal in the URL that the client should do TLS (e.g., "https:")

- Using a separate port for the secure and non-secure versions of the protocol.

- An extension to the application protocol to negotiate use or non-use of TLS ("STARTTLS")

While mechanisms of this type are in wide use, they all require modifications to the application layer and thus do not meet the goals of TCPINC. This document describes a TCP option which allows a pair of communicating TCP endpoints to negotiate TLS use automatically

without modifying the application layer protocols, thus allowing for transparent deployment.

2. Overview

The basic idea behind the TCP-TLS option is simple. The SYN and SYN/ACK messages carry TCP options indicating the willingness to do TLS and some basic information about the expected TLS modes. If both sides want to do TLS and have compatible modes, then the application data is automatically TLS protected prior to being sent over TCP. Otherwise, the application data is sent as usual.

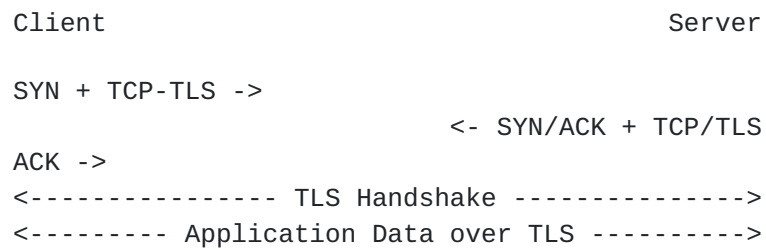


Figure 1: Negotiating TLS with TCP-TLS

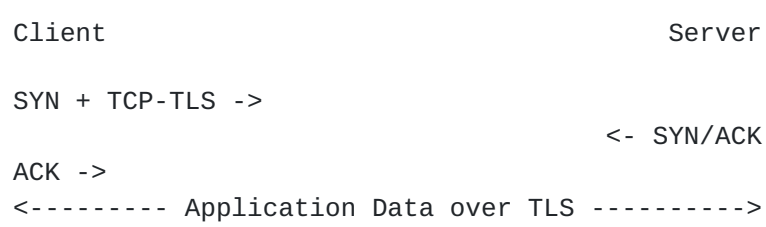
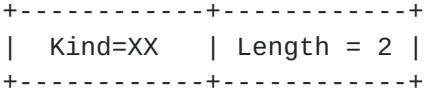


Figure 2: Fall back to TCP

If use of TLS is negotiated, the data sent over TCP simply is TLS data in compliance with {{RFC5246}.

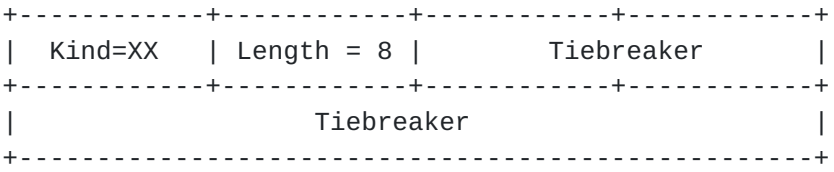
3. Extension Definition

The TCP-TLS option is very simple. For the normal case where each side knows who is the passive and who is the active opener, the option is empty. I.e.



In this case, the active opener MUST take on the role of TLS Client.

In the abnormal case of simultaneous open, the option includes a tiebreaker value.



The tiebreaker field is a 48-bit value which is used to determine the TLS roles, with the highest value being the TLS client and the lowest value being the TLS server. Applications MUST generate the tiebreaker randomly. If both sides generate the same tiebreaker value, then TCP-TLS MUST NOT be used (this has a vanishing probability of happening by accident.)

The default mode of operation MUST be the ordinary client/server mode and implementations MUST only use the simultaneous open mode if instructed by an application. If an implementation in simultaneous open mode receives an option without a tiebreaker, it MUST treat that tiebreaker as 0. If simultaneous open mode is not in use, and implementations detect a simultaneous open, then they MUST NOT try to negotiate TLS, regardless of the presence of this option.

If an endpoint sends the TCP-TLS option and correctly receives it from the other side it SHALL immediately negotiate TLS, taking on the role described above.

Once the TLS handshake has completed, all application data SHALL be sent over that negotiated TLS channel. Application data MUST NOT be sent prior to the TLS handshake.

If the TLS handshake fails for non-cryptographic reasons such as failure to negotiate a compatible cipher or the like, endpoints SHOULD behave as if the the TCP-TLS option was not present. This is obviously not the conventional behavior for TLS failure, but as the entire idea here is to be opportunistic and the attacker can simply suppress the TCP-TLS option entirely, this provides the maximum robustness against broken intermediaries. If the TLS handshake fails for cryptographic reasons that indicate damage to the datastream (e.g., a decryption failure or a Finished failure) then the endpoints SHOULD signal a connection failure, as this suggests that there is a middlebox modifying the data and there is a reasonable chance that the state is now corrupted.

4. Transport Integrity

The basic operational mode defined by TCP-TLS protects only the application layer content, but not the TCP segment metadata. Upon receiving a packet, implementations MUST first check the TCP checksum and discard corrupt packets without presenting them to TLS. If the TCP checksum passes but TLS integrity fails, the connection MUST be torn down.

Thus, TCP-TLS provides automatic security for the content, but not protection against DoS-style attacks. For instance, attackers will be able to inject RST packets, bogus application segments, etc., regardless of whether TLS authentication is used. Because the application data is TLS protected, this will not result in the application receiving bogus data, but it will constitute a DoS on the connection.

This attack can be countered by using TCP-TLS in combination with TCP-AO [RFC5925], as follows:

1. The TLS connection is negotiated using the "tcpao" ALPN [I-D.ietf-tls-applayerprotoneg] indicator.

2. Upon TLS handshake completion, a TLS Exporter [RFC5705] is used to generate keying material of appropriate length using exporter label TBD.

3. Further packets are protected using TCP-AO with the generated keys.

The Finished messages MUST NOT be protected with AO. The first application data afterwards MUST be protected with AO. Note that because of retransmission, non-AO packets may be received after AO has been engaged; they MUST be ignored.

[[OPEN ISSUE: How do we negotiate the parameters? Do we need a use_ao option like with RFC 5764? Is ALPN really what we want here?]]

[[TODO: verify that the state machine matches up here.]]

5. Implementation Options

There are two primary implementation options for TCP-TLS:

o Implement all of TCP-TLS in the operating system kernel.

- Implement just the TCP-TLS negotiation option in the operating system kernel with an interface to tell the application that TCP-TLS has been negotiated and therefore that the application must negotiate TLS.

The former option obviously achieves easier deployment for applications, which don't have to do anything, but is more effort for kernel developers and requires a wider interface to the kernel to configure the TLS stack. The latter option is inherently more flexible but does not provide as immediate transparent deployment. It is also possible for systems to offer both options.

6. TLS Profile

Implementations of this specification MUST at minimum support TLS 1.2 [RFC5246] and MUST support cipher suite XXX. Implementations MUST NOT negotiate versions of TLS prior to TLS 1.2. Implementations MUST NOT negotiate non-AEAD cipher suites and MUST use only PFS cipher suites with a key of at least 2048 bits (finite field) or 256 bites (elliptic curve).

[[OPEN ISSUE: What cipher suites? Presumably we require one authenticated and one anonymous cipher suite, all with GCM.]] [[OPEN ISSUE: If TLS 1.3 is ready, we may want to require that.]]

7. Channel Bindings

This specification is compatible with external authentication via TLS Channel Bindings [RFC5929]. If Channel Bindings are to be used, the TLS Extended Master Secret Extension [I-D.ietf-tls-session-hash]

8. NAT/Firewall considerations

If use of TLS is negotiated, the data sent over TCP simply is TLS data in compliance with {{RFC5246}. Thus it is extremely likely to pass through NATs, firewalls, etc. The only kind of middlebox that is likely to cause a problem is one which does protocol enforcement that blocks TLS on arbitrary (non-443) ports but _also_ passes unknown TCP options. Although no doubt such devices do exist, because this is a common scenario, a client machine should be able to probe to determine if it is behind such a device relatively readily.

9. IANA Considerations

IANA [shall register/has registered] the TCP option XX for TCP-TLS.

IANA [shall register/has registered] the ALPN code point "tcpao" to indicate the use of TCP-TLS with TCP-AO.

10. Security Considerations

The mechanisms in this document are inherently vulnerable to active attack because an attacker can remove the TCP-TLS option, thus downgrading you to ordinary TCP. Even when TCP-AO is used, all that is being provided is continuity of authentication from the initial handshake. If some sort of external authentication mechanism was provided or certificates are used, then you might get some protection against active attack.

Once the TCP-TLS option has been negotiated, then the connection is resistant to active data injection attacks. If TCP-AO is not used, then injected packets appear as bogus data at the TLS layer and will result in MAC errors followed by a fatal alert. The result is that while data integrity is provided, the connection is not resistant to DoS attacks intended to terminate it.

If TCP-AO is used, then any bogus packets injected by an attacker will be rejected by the TCP-AO integrity check and therefore will never reach the TLS layer. Thus, in this case, the connection is also resistant to DoS attacks, provided that endpoints require integrity protection for RST packets. If endpoints accept unauthenticated RST, then no DoS protection is provided.

11. References

11.1. Normative References

[I-D.ietf-tls-applayerprotoneg]
Friedl, S., Popov, A., Langley, A., and S. Emile, "Transport Layer Security (TLS) Application Layer Protocol Negotiation Extension", draft-ietf-tls-applayerprotoneg-05 (work in progress), March 2014.

[I-D.ietf-tls-session-hash]
Bhargavan, K., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", draft-ietf-tls-session-hash-05 (work in progress), April 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, March 2010.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, June 2010.

11.2. Informative References

[I-D.bittau-tcp-crypt]
Bittau, A., Boneh, D., Hamburg, M., Handley, M., Mazieres, D., and Q. Slack, "Cryptographic protection of TCP Streams (tcpcrypt)", draft-bittau-tcp-crypt-04 (work in progress), February 2014.

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, July 2010.

Author's Address

Eric Rescorla
Mozilla

EMail: ekr@rtfm.com