Network Working Group
Internet-Draft
Intended status: Experimental
Expires: January 1, 2017

A. Bittau
D. Boneh
D. Giffin
Stanford University
M. Handley
University College London
D. Mazieres
Stanford University
E. Smith
Kestrel Institute

June 30, 2016

TCP-ENO: Encryption Negotiation Option
draft-ietf-tcpinc-tcpeno-02

Abstract

Despite growing adoption of TLS [RFC5246], a significant fraction of TCP traffic on the Internet remains unencrypted. The persistence of unencrypted traffic can be attributed to at least two factors. First, some legacy protocols lack a signaling mechanism (such as a "STARTTLS" command) by which to convey support for encryption, making incremental deployment impossible. Second, legacy applications themselves cannot always be upgraded, requiring a way to implement encryption transparently entirely within the transport layer. The TCP Encryption Negotiation Option (TCP-ENO) addresses both of these problems through a new TCP option kind providing out-of-band, fully backward-compatible negotiation of encryption.

Status of This Memo

Copyright Notice

Table of Contents

1. Requirements language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Introduction

Many applications and protocols running on top of TCP today do not encrypt traffic. This failure to encrypt lowers the bar for certain attacks, harming both user privacy and system security. Counteracting the problem demands a minimally intrusive, backward-compatible mechanism for incrementally deploying encryption. The TCP Encryption Negotiation Option (TCP-ENO) specified in this document provides such a mechanism.

Introducing TCP options, extending operating system interfaces to support TCP-level encryption, and extending applications to take advantage of TCP-level encryption all require effort. To the greatest extent possible, the effort invested in realizing TCP-level encryption today needs to remain applicable in the future should the need arise to change encryption strategies. To this end, it is useful to consider two questions separately:

1. How to negotiate the use of encryption at the TCP layer, and

2. How to perform encryption at the TCP layer.

This document addresses question 1 with a new TCP option, ENO. TCP-ENO provides a framework in which two endpoints can agree on one among multiple possible TCP encryption _specs_. For future compatibility, encryption specs can vary widely in terms of wire format, use of TCP option space, and integration with the TCP header and segmentation. However, ENO abstracts these differences to ensure the introduction of new encryption specs can be transparent to applications taking advantage of TCP-level encryption.

Question 2 is addressed by one or more companion documents describing encryption specs. While current specs enable TCP-level traffic encryption today, TCP-ENO ensures that the effort invested to deploy today's specs will additionally benefit future specs.

2.1. Design goals

TCP-ENO was designed to achieve the following goals:

1. Enable endpoints to negotiate the use of a separately specified TCP encryption _spec_.

2. Transparently fall back to unencrypted TCP when not supported by both endpoints.

3. Provide out-of-band signaling through which applications can better take advantage of TCP-level encryption (for instance, by improving authentication mechanisms in the presence of TCP-level encryption).

4. Provide a standard negotiation transcript through which specs can defend against tampering with TCP-ENO.

5. Make parsimonious use of TCP option space.

6. Define roles for the two ends of a TCP connection, so as to name each end of a connection for encryption or authentication purposes even following a symmetric simultaneous open.

3. Terminology

We define the following terms, which are used throughout this document:

SYN segment
A TCP segment in which the SYN flag is set

ACK segment
A TCP segment in which the ACK flag is set (which includes most segments other than an initial SYN segment)

non-SYN segment
A TCP segment in which the SYN flag is clear

SYN-only segment
A TCP segment in which the SYN flag is set but the ACK flag is clear

SYN-ACK segment
A TCP segment in which the SYN and ACK flags are both set

Active opener
A host that sends a SYN-only segment. With the BSD socket API, this occurs when an application calls "connect". In client-server configurations, active openers are typically clients.

Passive opener
A host that does not send a SYN-only segment (only a SYN-ACK segment). With the BSD socket API, this occurs following a call

to "listen". In client-server configurations, passive openers are typically servers.

Simultaneous open
The act of symmetrically establishing a TCP connection between two active openers (both of which call "connect" with BSD sockets). Each host of a simultaneous open sends both a SYN-only and a SYN-ACK segment. Simultaneous open is less common than asymmetric open, but can be used for NAT traversal by peer-to-peer applications [RFC5382].

Encryption spec
A separate document specifying an approach to encrypting TCP traffic in conjunction with TCP-ENO.

Spec identifier
A unique 7-bit value in the range 0x20-0x7f that IANA has assigned to an encryption spec.

Negotiated [encryption] spec
The single encryption spec governing a TCP connection, as determined by the protocol specified in this document.

4. TCP-ENO specification

TCP-ENO extends TCP connection establishment to enable encryption opportunistically. It uses a new TCP option kind to negotiate one among multiple possible encryption specs--separate documents describing how to do actual traffic encryption. The negotiation involves hosts exchanging sets of supported specs, where each spec is represented by a _suboption_ within a larger TCP option in the offering host's SYN segment.

If TCP-ENO succeeds, it yields the following information:

- A negotiated encryption spec, represented by a unique 7-bit spec identifier,

- A few extra bytes of suboption data from each host, if needed by the spec,

- A negotiation transcript that the negotiated spec must cryptograhpically authenticate,

- Role assignments designating one endpoint "host A" and the other endpoint "host B", and

- A few bits indicating whether or not the application at each end knows it is using TCP-ENO.

If TCP-ENO fails, encryption is disabled and the connection falls back to traditional unencrypted TCP.

The remainder of this section provides the normative description of the TCP ENO option and handshake protocol.

4.1. ENO option

TCP-ENO employs an option in the TCP header [RFC0793]. There are two equivalent kinds of ENO option, shown in Figure 1. Section 10 specifies which of the two kinds is permissible and/or preferred.

```
   byte    0     1     2             N+1   (N+2 bytes total)
        +-----+-----+-----+--....--+-----+
        |Kind=|Len= |                    |
        | 69  | N+2 | contents (N bytes) |
        +-----+-----+-----+--....--+-----+

   byte    0     1     2     3     4             N+3   (N+4 bytes total)
        +-----+-----+-----+-----+-----+--....--+-----+
        |Kind=|Len= |   ExID    |                    |
        | 253 | N+4 | 69  | 78  | contents (N bytes) |
        +-----+-----+-----+-----+-----+--....--+-----+
```

Figure 1: Two equivalent kinds of TCP-ENO option

The contents of an ENO option can take one of two forms. A SYN form, illustrated in Figure 2, appears only in SYN segments. A non-SYN form, illustrated in Figure 3, appears only in non-SYN segments. The SYN form of ENO acts as a container for one or more suboptions, labeled "Opt_0", "Opt_1", ... in Figure 2. The non-SYN form, by its presence, acts as a one-bit acknowledgment, with the actual contents ignored by ENO. Particular encryption specs MAY assign additional meaning to the contents of non-SYN ENO options. When a negotiated spec does not assign such meaning, the contents of a non-SYN ENO option SHOULD be zero bytes.

```
   byte    0     1     2     3                     ... N+1
        +-----+-----+-----+-----+--...--+-----+----...----+
        |Kind=|Len= |Opt_0|Opt_1|       |Opt_i|   Opt_i   |
        | 69  | N+2 |     |     |       |     |   data    |
        +-----+-----+-----+-----+--...--+-----+----...----+

   byte    0     1     2     3     4     5                     ... N+3
        +-----+-----+-----+-----+-----+-----+--...--+-----+----...----+
        |Kind=|Len= |   ExID    |Opt_0|Opt_1|       |Opt_i|   Opt_i   |
        | 253 | N+4 | 69  | 78  |     |     |       |     |   data    |
        +-----+-----+-----+-----+-----+-----+--...--+-----+----...----+
```

Figure 2: SYN form of ENO

```
                byte   0     1     2     N+1
                    +-----+-----+-----...----+
                    |Kind=|Len= |  ignored   |
                    | 69  | N+2 | by TCP-ENO |
                    +-----+-----+-----...----+

                byte   0     1     2     3     4     N+3
                    +-----+-----+-----+-----+-----...----+
                    |Kind=|Len= |   ExID    |  ignored   |
                    | 253 | N+4 | 69  | 78  | by TCP-ENO |
                    +-----+-----+-----+-----+-----...----+
```

Figure 3: Non-SYN form of ENO, where N MAY be 0

Every suboption starts with a byte of the form illustrated in Figure 4. The high bit "v", when set, introduces suboptions with variable-length data. When "v = 0", the byte itself constitutes the entirety of the suboption. The 7-bit value "cs" expresses one of:

- Global configuration data (discussed in Section 4.2),

- Suboption data length for the next suboption (discussed in Section 4.4), or

- An offer to use a particular encryption spec detailed in a separate document.

```
bit   7   6   5   4   3   2   1   0
    +---+---+---+---+---+---+---+---+
    | v |            cs             |
    +---+---+---+---+---+---+---+---+

    v  - non-zero for use with variable-length suboption data
    cs - global configuration option or encryption spec identifier
```

Figure 4: Format of initial suboption byte

Table 1 summarizes the meaning of initial suboption bytes. Values of "cs" greater than or equal to 0x20 are spec identifiers, while those below 0x20 are shared between general suboptions and length bytes. When "v = 0", the initial suboption byte constitutes the entirety of the suboption and all information is expressed by the 7-bit value "cs", which can be a spec identifier or general suboption. When "v = 1", it indicates a suboption with one or more bytes of suboption data. Only spec identifiers may have suboption data, not general suboptions. Hence, bytes with "v = 1" and "cs < 0x20" are not general suboptions but rather length bytes governing the length of the next suboption. In the absence of a length byte, a spec identifier suboption with "v = 1" has suboption data extending to the end of the TCP option.

cs	v	Meaning
0x00-0x1f	0	General suboption (Section 4.2)
0x00-0x1f	1	Length byte (Section 4.4)
0x20-0x7f	0	Encryption spec without suboption data
0x20-0x7f	1	Encryption spec followed by suboption data

Table 1: Initial suboption byte values

A SYN segment MUST contain at most one ENO TCP option. If a SYN segment contains more than one ENO option, the receiver MUST behave as though the segment contained no ENO options and disable encryption. An encryption spec MAY define the use of multiple ENO options in a non-SYN segment. For non-SYN segments, ENO itself only distinguishes between the presence or absence of ENO options; multiple ENO options are interpreted the same as one.

4.2. General suboptions

Suboptions 0x00-0x1f are used for general conditions that apply regardless of the negotiated encryption spec. A TCP SYN segment MUST include at most one ENO suboption in this range. A receiver MUST

ignore all but the first suboption in this range so as to anticipate future revisions of ENO that assign new meaning to bits in subsequent general suboptions. The value of a general suboption byte is interpreted as a bitmask, illustrated in Figure 5.

```
            bit   7   6   5   4   3   2   1   0
                +---+---+---+-------+---+---+---+
                | 0 | 0 | 0 |  zz   | m | a | b |
                +---+---+---+-------+---+---+---+

                b   - Passive role bit
                a   - Application-aware bit
                m   - Middleware signaling bit
                zz  - Zero bits (reserved for future use)
```

Figure 5: Format of the general option byte

The fields of the bitmask are interpreted as follows:

b

The passive role bit MUST be 1 for all passive openers. For active openers, it MUST default to 0, but implementations SHOULD provide an API through which an application can set "b = 1" before calling "connect". (Manual configuration of "b" is necessary for simultaneous open.)

a

The application-aware bit "a" is an out-of-band signal indicating that the application on the sending host is aware of TCP-ENO and has been extended to alter its behavior in the presence of encrypted TCP. Implementations MUST set this bit to 0 by default, and SHOULD provide an API through which applications can change the value of the bit as well as examine the value of the bit sent by the remote host. Implementations SHOULD furthermore support a _mandatory_ application-aware mode in which TCP-ENO is automatically disabled if the remote host does not set "a = 1".

m

The middleware bit "m" functions similarly to the application-aware bit "a". It is available for middleware such as shared libraries needing out-of-band signaling to improve the security of legacy applications. Implementations MUST set this bit to 0 by default and SHOULD provide an API through which software can change the value. Unlike the application-aware "a" bit, no mandatory mode is needed for the middleware bit.

zz

The "zz" bits are reserved for future revisions of TCP-ENO. They MUST be set to zero in sent segments and MUST be ignored in received segments.

A SYN segment without an explicit general suboption has an implicit general suboption of 0x00. Because passive openers MUST always set "b = 1", they cannot rely on this implicit 0x00 byte and MUST include an explicit general suboption in the ENO options of their SYN-ACK segments.

4.3. TCP-ENO roles

TCP-ENO uses abstract roles to distinguish the two ends of a TCP connection. These roles are determined by the "b" bit in the general suboption. The host that sent an implicit or explicit suboption with "b = 0" plays the "A" role. The host that sent "b = 1" plays the "B" role.

If both sides of a connection set "b = 1" (which can happen if the active opener misconfigures "b" before calling "connect"), or both sides set "b = 0" (which can happen with simultaneous open), then TCP-ENO MUST be disabled and the connection MUST fall back to unencrypted TCP.

Encryption specs SHOULD refer to TCP-ENO's A and B roles to specify asymmetric behavior by the two hosts. For the remainder of this document, we will use the terms "host A" and "host B" to designate the hosts with A and B roles, respectively, in a connection.

4.4. Specifying suboption data length

An encryption spec MAY optionally specify the use of one or more bytes of suboption data. The presence of such data is indicated by setting "v = 1" in the initial suboption byte (see Figure 4). By default, suboption data extends to the end of the TCP option. Hence, if only one suboption requires data, the most compact way to encode it is to place it last in the ENO option, after all one-byte suboptions. As an example, in Figure 2, the last suboption, "Opt_i", has suboption data and thus requires "v = 1"; however, the suboption data length can be implicit in the total length of the TCP option.

When a suboption with data is not last in an ENO option, the sender MUST explicitly specify the suboption data length for the receiver to know where the next suboption starts. The sender does so by preceding the suboption with a length byte.

Figure 6 shows the format of a length byte. It encodes a 5-bit value "nnnnn". Adding one to "nnnnn" yields the length of the suboption

data not including the length byte and initial spec identifier byte. Hence, a length byte can designate a suboption carrying anywhere from 1 to 32 bytes of suboption data (inclusive).

```
               bit   7   6   5   4   3   2   1   0
                   +---+---+---+-------------------+
                   | 1   0   0         nnnnn       |
                   +---+---+---+-------------------+

                   nnnnn - 5-bit value encoding (length - 1)
```

Figure 6: Format of a length byte

A suboption preceded by a length byte MUST be a spec identifier ("cs >= 0x20") and MUST have "v = 1". Figure 7 shows an example of such a suboption.

```
       byte    0      1       2      nnnnn+2  (nnnnn+3 bytes total)
            +------+------+-------...-------+
            |length| spec | suboption data  |
            | byte |ident.| (nnnnn+1 bytes) |
            +------+------+-------...-------+

            length byte     - specifies nnnnn
            spec identifier - MUST have v = 1 and cs >= 0x20
            suboption data  - length specified by nnnnn+1
```

Figure 7: Suboption with length byte

If the octet following a length byte has the high bit clear (meaning "v = 0"), then the length byte and following octet together are interpreted as a length word, as shown in Figure 8. The length word encodes an 8-bit value corresponding to one less than the suboption data length. As with length bytes, spec identifiers MUST have "v = 1".

```
   bit   15  14  13  12  11  10  9   8   7   6   5   4   3   2   1   0
       +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
       | 1   0   0       zzzz      | m | 0 |            nnnnnnn        |
       +---+---+---+-------------------+---+---------------------------+

       nnnnnnn - 7 least significant bits of 8-bit value (length - 1)
       m       - Most significant bit of 8-bit value (length - 1)
       zzzz    - Bits that MUST be zero (reserved for future use)
```

Figure 8: Format of a length word

The "zzzz" bits in a length word MUST be set to 0 by a sender. If a host receives a length word in which the "zzzz" bits are not all 0, it MUST ignore the entire received ENO option and disable encryption. Similarly, if a length byte or word in a received SYN segment indicates that a TCP-ENO suboption would extend beyond the end of the ENO TCP option, the receiver MUST behave as though the received SYN segment contained no TCP-ENO option and disable encryption.

4.5. The negotiated spec

A spec identifier "cs" is _valid_ for a connection when:

1. Each side has sent a suboption for "cs" in its SYN-form ENO option,

2. Any suboption data in these "cs" suboptions is valid according to the spec and satisfies any runtime constraints, and

3. If one host sends multiple suboptions with "cs", then such repetition is well-defined by the encryption spec.

The _negotiated encryption spec_ is the last valid spec identifier in host B's SYN-form ENO option. This means host B specifies suboptions in order of increasing priority, while host A does not influence spec priority.

A passive opener (which is always host B) sees the remote host's SYN segment before constructing its own SYN-ACK. Hence, a passive opener SHOULD include only one spec identifier in SYN-ACK segments and SHOULD ensure this spec identifier is valid. However, simultaneous open or implementation considerations can prevent host B from offering only one encryption spec.

4.6. TCP-ENO handshake

A host employing TCP-ENO for a connection MUST include an ENO option in every TCP segment sent until either encryption is disabled or the host receives a non-SYN segment.

A host MUST disable encryption, refrain from sending any further ENO options, and fall back to unencrypted TCP if any of the following occurs:

1. Any segment it receives up to and including the first received ACK segment does not contain an ENO option,

2. The SYN segment it receives does not contain a valid spec identifier, or

3. It receives a SYN segment with an incompatible general suboption. (Specifically, incompatible means the two hosts set the same "b" value or the connection is in mandatory application-aware mode and the remote host set "a = 0".)

Hosts MUST NOT alter SYN-form ENO options in retransmitted segments, or between the SYN and SYN-ACK segments of a simultaneous open, with two exceptions for an active opener. First, an active opener MAY unilaterally disable ENO (and thus remove the ENO option) between retransmissions of a SYN-only segment. (Such removal could be useful if middleboxes are dropping segments with the ENO option.) Second, an active opener performing simultaneous open MAY include no TCP-ENO option in its SYN-ACK if the received SYN caused it to disable encryption according to the above rules (for instance because role negotiation failed).

Once a host has both sent and received an ACK segment containing an ENO option, encryption MUST be enabled. Once encryption is enabled, hosts MUST follow the encryption protocol of the negotiated spec and MUST NOT present raw TCP payload data to the application. In particular, data segments MUST contain ciphertext or key agreement messages as determined by the negotiated spec, and MUST NOT contain plaintext application data.

4.7. Negotiation transcript

To defend against attacks on encryption negotiation itself, encryption specs need a way to reference a transcript of TCP-ENO's negotiation. In particular, an encryption spec MUST fail with high probability if its selection resulted from tampering with or forging initial SYN segments.

TCP-ENO defines its negotiation transcript as a packed data structure consisting of two TCP-ENO options exactly as they appeared in the TCP header (including the TCP option kind, TCP option length byte, and, for option kind 253, the bytes 69 and 78 as illustrated in Figure 1). The transcript is constructed from the following, in order:

1. The TCP-ENO option in host A's SYN segment, including the kind and length bytes.

2. The TCP-ENO option in host B's SYN segment, including the kind and length bytes.

Note that because the ENO options in the transcript contain length bytes as specified by TCP, the transcript unambiguously delimits A's and B's ENO options.

5. Requirements for encryption specs

Though TCP-ENO affords spec authors a large amount of design flexibility, to abstract spec differences away from applications requires fitting them all into a coherent framework. As such, any encryption spec claiming an ENO spec identifier MUST satisfy the following normative list of properties.

- Specs MUST protect TCP data streams with authenticated encryption.

- Specs MUST define a session ID whose value identifies the TCP connection and, with overwhelming probability, is unique over all time if either host correctly obeys the spec. Section 5.1 describes the requirements of the session ID in more detail.

- Specs MUST NOT permit the negotiation of any encryption algorithms with significantly less than 128-bit security.

- Specs MUST NOT allow the negotiation of null cipher suites, even for debugging purposes. (Implementations MAY support debugging modes that allow applications to extract their own session keys.)

- Specs MUST NOT depend on long-lived secrets for data confidentiality, as implementations SHOULD provide forward secrecy some bounded, short time after the close of a TCP connection.

- Specs MUST protect and authenticate the end-of-file marker traditionally conveyed by TCP's FIN flag when the remote application calls "close" or "shutdown". However, end-of-file MAY be conveyed though a mechanism other than TCP FIN. Moreover, specs MAY permit attacks that cause TCP connections to abort, but such an abort MUST raise an error that is distinct from an end-of-file condition.

- Specs MAY disallow the use of TCP urgent data by applications, but MUST NOT allow attackers to manipulate the URG flag and urgent pointer in ways that are visible to applications.

5.1. Session IDs

Each spec MUST define a session ID that uniquely identifies each encrypted TCP connection. Implementations SHOULD expose the session ID to applications via an API extension. Applications that are aware of TCP-ENO SHOULD incorporate the session ID value and TCP-ENO role (A or B) into any authentication mechanisms layered over TCP encryption so as to authenticate actual TCP endpoints.

In order to avoid replay attacks and prevent authenticated session IDs from being used out of context, session IDs MUST be unique over all time with high probability. This uniqueness property MUST hold even if one end of a connection maliciously manipulates the protocol in an effort to create duplicate session IDs. In other words, it MUST be infeasible for a host, even by deviating from the encryption spec, to establish two TCP connections with the same session ID to remote hosts obeying the spec.

To prevent session IDs from being confused across specs, all session IDs begin with the negotiated spec identifier--that is, the last valid spec identifier in host B's SYN segment. If the "v" bit was 1 in host B's SYN segment, then it is also 1 in the session ID. However, only the first byte is included, not the suboption data. Figure 9 shows the resulting format. This format is designed for spec authors to compute unique identifiers; it is not intended for application authors to pick apart session IDs. Applications SHOULD treat session IDs as monolithic opaque values and SHOULD NOT discard the first byte to shorten identifiers.

```
              byte    0     1     2        N-1    N
                   +-----+------------...------------+
                   | sub-| collision-resistant hash  |
                   | opt | of connection information |
                   +-----+------------...------------+
```

Figure 9: Format of a session ID

Though specs retain considerable flexibility in their definitions of the session ID, all session IDs MUST meet certain minimum requirements. In particular:

- The session ID MUST be at least 33 bytes (including the one-byte suboption), though specs may choose longer session IDs.

- The session ID MUST depend in a collision-resistant way on fresh data contributed by both sides of the connection.

- The session ID MUST depend in a collision-resistant way on any public keys, public Diffie-Hellman parameters, or other public asymmetric cryptographic parameters that are employed by the encryption spec and have corresponding private data that is known by only one side of the connection.

- Unless and until applications disclose information about the session ID, all but the first byte MUST be computationally indistinguishable from random bytes to a network eavesdropper.

- Applications MAY chose to make session IDs public. Therefore, specs MUST NOT place any confidential data in the session ID (such as data permitting the derivation of session keys).

- The session ID MUST depend on the negotiation transcript specified in Section 4.7 in a collision-resistant way.

6. Examples

This subsection illustrates the TCP-ENO handshake with a few non-normative examples.

```
(1) A -> B:  SYN      ENO<X,Y>
(2) B -> A:  SYN-ACK  ENO<b=1,Y>
(3) A -> B:  ACK      ENO<>
[rest of connection encrypted according to spec for Y]
```

Figure 10: Three-way handshake with successful TCP-ENO negotiation

Figure 10 shows a three-way handshake with a successful TCP-ENO negotiation. The two sides agree to follow the encryption spec identified by suboption Y.

```
(1) A -> B:  SYN      ENO<X,Y>
(2) B -> A:  SYN-ACK
(3) A -> B:  ACK
[rest of connection unencrypted legacy TCP]
```

Figure 11: Three-way handshake with failed TCP-ENO negotiation

Figure 11 shows a failed TCP-ENO negotiation. The active opener (A) indicates support for specs corresponding to suboptions X and Y. Unfortunately, at this point one of several things occurs:

1. The passive opener (B) does not support TCP-ENO,

2. B supports TCP-ENO, but supports neither of specs X and Y, and so does not reply with an ENO option,

3. B supports TCP-ENO, but has the connection configured in mandatory application-aware mode and thus disables ENO because A's SYN segment does not set the application-aware bit, or

4. The network stripped the ENO option out of A's SYN segment, so B did not receive it.

Whichever of the above applies, the connection transparently falls back to unencrypted TCP.

```
(1) A -> B:  SYN      ENO<X,Y>
(2) B -> A:  SYN-ACK  ENO<b=1,X> [ENO stripped by middlebox]
(3) A -> B:  ACK
[rest of connection unencrypted legacy TCP]
```

Figure 12: Failed TCP-ENO negotiation because of network filtering

Figure 12 Shows another handshake with a failed encryption negotiation. In this case, the passive opener B receives an ENO option from A and replies. However, the reverse network path from B to A strips ENO options. Hence, A does not receive an ENO option from B, disables ENO, and does not include a non-SYN form ENO option when ACKing the other host's SYN segment. The lack of ENO in A's ACK segment signals to B that the connection will not be encrypted. At this point, the two hosts proceed with an unencrypted TCP connection.

```
(1) A -> B:  SYN      ENO<Y,X>
(2) B -> A:  SYN      ENO<b=1,X,Y,Z>
(3) A -> B:  SYN-ACK  ENO<Y,X>
(4) B -> A:  SYN-ACK  ENO<b=1,X,Y,Z>
[rest of connection encrypted according to spec for Y]
```

Figure 13: Simultaneous open with successful TCP-ENO negotiation

Figure 13 shows a successful TCP-ENO negotiation with simultaneous open. Here the first four segments MUST contain a SYN-form ENO option, as each side sends both a SYN-only and a SYN-ACK segment. The ENO option in each host's SYN-ACK is identical to the ENO option in its SYN-only segment, as otherwise connection establishment could not recover from the loss of a SYN segment. The last valid spec in host B's ENO option is Y, so Y is the negotiated spec.

7. Design rationale

This section describes some of the design rationale behind TCP-ENO.

7.1. Future developments

TCP-ENO is designed to capitalize on future developments that could alter trade-offs and change the best approach to TCP-level encryption (beyond introducing new cipher suites). By way of example, we discuss a few such possible developments.

Various proposals exist to increase option space in TCP [I-D.ietf-tcpm-tcp-edo][I-D.briscoe-tcpm-inspace-mode-tcpbis][I-D.touch-tcpm-tcp-syn-ext-opt]. If SYN segments gain large options, it becomes possible to fit public keys or Diffie-Hellman parameters into SYN segments. Future encryption specs can take advantage of this by performing key

agreement directly within suboption data, both simplifying protocols and reducing the number of round trips required for connection setup.

New revisions to socket interfaces [RFC3493] could involve library calls that simultaneously have access to hostname information and an underlying TCP connection. Such an API enables the possibility of authenticating servers transparently to the application, particularly in conjunction with technologies such as DANE [RFC6394]. The middleware bit "m" in general suboptions enables a middleware library to indicate to the peer that it wishes to engage in an authentication protocol before turning the TCP connection over to the application. Different middleware authentication protocols can employ unique identifiers to multiplex the "m" bit.

TLS can currently only be added to legacy applications whose protocols accommodate a STARTTLS command or equivalent. TCP-ENO, because it provides out-of-band signaling, opens the possibility of future TLS revisions being generically applicable to any TCP application.

7.2. Handshake robustness

Incremental deployment of TCP-ENO depends critically on failure cases devolving to unencrypted TCP rather than causing the entire TCP connection to fail.

Because some network paths drop ENO options in one direction only, a host must know not just that the peer supports encryption, but that the peer has received an ENO option. To this end, ENO disables encryption unless it receives an ACK segment bearing an ENO option. To stay robust in the face of dropped segments, hosts must continue to include non-SYN form ENO options in segments until such point as they have received a non-SYN segment from the other side.

One particularly pernicious middlebox behavior found in the wild is load balancers that echo unknown TCP options found in SYN segments back to an active opener. The passive role bit "b" in general suboptions ensures encryption will always be disabled under such circumstances, as sending back a verbatim copy of an active opener's SYN-form ENO option always causes role negotiation to fail.

7.3. Suboption data

Encryption specs can employ suboption data for session caching, cipher suite negotiation, or other purposes. However, TCP currently limits total option space consumed by all options to only 40 bytes, making it impractical to have many suboptions with data. For this reason, ENO optimizes the case of a single suboption with data by

inferring the length of the last suboption from the TCP option length. Doing so saves one byte.

7.4. Passive role bit

TCP-ENO, associated encryption specs, and applications all have asymmetries that require an unambiguous way to identify one of the two connection endpoints. As an example, Section 4.7 specifies that host A's ENO option comes before host B's in the negotiation transcript. As another example, an application might need to authenticate one end of a TCP connection with a digital signature. To ensure the signed message cannot not be interpreted out of context to authenticate the other end, the signed message would need to include both the session ID and the local role, A or B.

A normal TCP three-way handshake involves one active and one passive opener. This asymmetry is captured by the default configuration of the "b" bit in the general suboption. With simultaneous open, both hosts are active openers, so TCP-ENO requires that one host manually configure "b = 1". An alternate design might automatically break the symmetry to avoid this need for manual configuration. However, all such designs we considered either lacked robustness or consumed precious bytes of SYN option space even in the absence of simultaneous open. (One complicating factor is that TCP does not know it is participating in a simultaneous open until after it has sent a SYN segment. Moreover, with packet loss, one host might never learn it has participated in a simultaneous open.)

7.5. Option kind sharing

This draft does not specify the use of ENO options beyond the first few segments of a connection. Moreover, it does not specify the content of ENO options in non-SYN segments, only their presence. As a result, any use of option kind 69 (or option kind 253 with ExID 0x454E) after the SYN exchange does not conflict with this document. Because in addition ENO guarantees at most one negotiated spec per connection, encryption specs will not conflict with one another or ENO if they use ENO's option kind for out-of-band signaling in non-SYN segments.

8. Experiments

This document has experimental status because TCP-ENO's viability depends on middlebox behavior that can only be determined _a posteriori_. Specifically, we must determine to what extent middleboxes will permit the use of TCP-ENO. Once TCP-ENO is deployed, we will be in a better position to gather data on two types of failure:

1. Middleboxes downgrading TCP-ENO connections to unencrypted TCP. This can happen if middleboxes strip unknown TCP options or if they terminate TCP connections and relay data back and forth.

2. Middleboxes causing TCP-ENO connections to fail completely. This can happen if applications perform deep packet inspection and start dropping segments that unexpectedly contain ciphertext.

The first type of failure is tolerable since TCP-ENO is designed for incremental deployment anyway. The second type of failure is more problematic, and, if prevalent, will require the development of techniques to avoid and recover from such failures.

9. Security considerations

An obvious use case for TCP-ENO is opportunistic encryption--that is, encrypting some connections, but only where supported and without any kind of endpoint authentication. Opportunistic encryption protects against undetectable large-scale eavesdropping. However, it does not protect against detectable large-scale eavesdropping (for instance, if ISPs terminate and proxy TCP connections or simply downgrade them to unencrypted). Moreover, it emphatically does not protect against targeted attacks that employ trivial spoofing to redirect a specific high-value connection to a man-in-the-middle attacker.

Achieving stronger security with TCP-ENO requires verifying session IDs. Any application relying on ENO for communications security MUST incorporate session IDs into its endpoint authentication. By way of example, an authentication mechanism based on keyed digests (such Digest Access Authentication [RFC7616]) can be extended to include the role and session ID in the input of the keyed digest. Where necessary for backwards compatibility, applications SHOULD use the application-aware bit to negotiate the inclusion of session IDs in authentication.

Because TCP-ENO enables multiple different encryption specs to coexist, security could potentially be only as strong as the weakest available spec. In particular, if session IDs do not depend on the TCP-ENO transcript in a strong way, an attacker can undetectably tamper with ENO options to force negotiation of a deprecated and vulnerable spec. To avoid such problems, specs SHOULD compute session IDs using only well-studied and conservative hash functions. That way, even if other parts of a spec are vulnerable, it is still intractable for an attacker to induce identical session IDs at both ends after tampering with ENO contents in SYN segments.

Implementations MUST NOT send ENO options unless they have access to an adequate source of randomness [RFC4086]. Without secret

unpredictable data at both ends of a connection, it is impossible for encryption specs to achieve confidentiality and forward secrecy. Because systems typically have very little entropy on bootup, implementations might need to disable TCP-ENO until after system initialization.

With a regular three-way handshake (meaning no simultaneous open), the non-SYN form ENO option in an active opener's first ACK segment MAY contain N > 0 bytes of spec-specific data, as shown in Figure 3. Such data is not part of the TCP-ENO negotiation transcript, and hence MUST be separately authenticated by the encryption spec.

10. IANA Considerations

[This section is a placeholder, as David Black has offered to help rewrite it.]

Implementations MUST NOT use option kind 69 unless and until it is assigned to TCP-ENO by IANA. In the meantime, implementations MUST use experimental option 253 [RFC6994], to which IANA has assigned ExID 0x454E (encoded by decimal bytes 69, 78 in Figure 1). Conversely, after IANA assigns a dedicated option kind to TCP-ENO, the use of option 253 is deprecated.

IANA will need to maintain an ENO suboption registry mapping suboption "cs" values to encryption specs.

11. Acknowledgments

We are grateful for contributions, help, discussions, and feedback from the TCPINC working group, including Marcelo Bagnulo, David Black, Bob Briscoe, Jana Iyengar, Tero Kivinen, Mirja Kuhlewind, Yoav Nir, Christoph Paasch, Eric Rescorla, and Kyle Rose. This work was funded by DARPA CRASH and the Stanford Secure Internet of Things Project.

12. References

12.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[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>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <http://www.rfc-editor.org/info/rfc4086>.

[RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <http://www.rfc-editor.org/info/rfc6994>.

12.2. Informative References

[I-D.briscoe-tcpm-inspace-mode-tcpbis] Briscoe, B., "Inner Space for all TCP Options (Kitchen Sink Draft - to be Split Up)", draft-briscoe-tcpm-inspace-mode-tcpbis-00 (work in progress), March 2015.

[I-D.ietf-tcpm-tcp-edo] Touch, J. and W. Eddy, "TCP Extended Data Offset Option", draft-ietf-tcpm-tcp-edo-06 (work in progress), June 2016.

[I-D.touch-tcpm-tcp-syn-ext-opt] Touch, J. and T. Faber, "TCP SYN Extended Option Space Using an Out-of-Band Segment", draft-touch-tcpm-tcp-syn-ext-opt-04 (work in progress), April 2016.

[RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, DOI 10.17487/RFC3493, February 2003, <http://www.rfc-editor.org/info/rfc3493>.

[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>.

[RFC5382] Guha, S., Ed., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", BCP 142, RFC 5382, DOI 10.17487/RFC5382, October 2008, <http://www.rfc-editor.org/info/rfc5382>.

[RFC6394] Barnes, R., "Use Cases and Requirements for DNS-Based Authentication of Named Entities (DANE)", RFC 6394, DOI 10.17487/RFC6394, October 2011, <http://www.rfc-editor.org/info/rfc6394>.

[RFC7616] Shekh-Yusef, R., Ed., Ahrens, D., and S. Bremer, "HTTP Digest Access Authentication", RFC 7616, DOI 10.17487/RFC7616, September 2015, <http://www.rfc-editor.org/info/rfc7616>.

Authors' Addresses

Andrea Bittau
Stanford University
353 Serra Mall, Room 288
Stanford, CA 94305
US

Email: bittau@cs.stanford.edu

Dan Boneh
Stanford University
353 Serra Mall, Room 475
Stanford, CA 94305
US

Email: dabo@cs.stanford.edu

Daniel B. Giffin
Stanford University
353 Serra Mall, Room 288
Stanford, CA 94305
US

Email: dbg@scs.stanford.edu

Mark Handley
University College London
Gower St.
London WC1E 6BT
UK

Email: M.Handley@cs.ucl.ac.uk

David Mazieres
Stanford University
353 Serra Mall, Room 290
Stanford, CA 94305
US

Email: dm@uun.org

Eric W. Smith
Kestrel Institute
3260 Hillview Avenue
Palo Alto, CA 94304
US

Email: eric.smith@kestrel.edu