Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 9, 2017

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

July 8, 2016

# Interface Extensions for TCP-ENO
**draft-ietf-tcpinc-api-00**

## Abstract

TCP-ENO negotiates encryption at the transport layer. It also defines a few parameters that are intended to be used or configured by applications. This document specifies operating system interfaces for access to these TCP-ENO parameters. We describe the interfaces in terms of socket options, the de facto standard API for adjusting per-connection behavior in TCP/IP, and sysctl, a popular mechanism for setting global defaults. Operating systems that lack socket or sysctl functionality can implement similar interfaces in their native frameworks, but should ideally adapt their interfaces from those presented in this document.

## Status of This Memo

Copyright Notice



Table of Contents



## 1. Introduction

The TCP Encryption Negotiation Option (TCP-ENO) [I-D.ietf-tcpinc-tcpeno] permits hosts to negotiate encryption of a TCP connection. One of TCP-ENO's use cases is to encrypt traffic transparently, unbeknownst to legacy applications. Transparent encryption requires no changes to existing APIs. However, other use cases require applications to interact with TCP-ENO. In particular:

- o Transparent encryption protects only against passive eavesdroppers. Stronger security requires applications to authenticate a _Session ID_ value associated with each encrypted connection.

- Applications that have been updated to authenticate Session IDs must somehow advertise this fact to peers in a backward-compatible way. TCP-ENO carries an "application-aware" bit for this purpose, as well as a "middleware" bit, but these bits are not accessible through existing interfaces.

- Applications employing TCP's simultaneous open feature need a way to configure a passive-role bit to break symmetry for TCP-ENO.

- System administrators and applications may wish to set and examine negotiation preferences, such as which encryption schemes (and perhaps versions) to enable and disable.

- Applications that perform their own encryption may wish to disable TCP-ENO entirely.

The remainder of this document describes an API through which systems can meet the above needs. The API extensions relate back to quantities defined by TCP-ENO.

## 2. API extensions

This section describes an API for per-connection options, followed by a discussion of system-wide configuration options.

### 2.1. Per-connection options

Table 1 summarizes a set of options that TCP-ENO implementations should provide on a per-socket basis. For each option, the table lists whether it is read-only (R) or read-write (RW), as well as the type of the option's value. Read-write options, when read, always return the previously successfully written value or the default if they have not been written. Options of type "bytes" consist of a variable-length array of bytes, while options of type "int" consist of a small integer with the exact range indicated in parentheses. We discuss each option in more detail below.

| Option name | RW | Type |
| --- | --- | --- |
| TCP_ENO_ENABLED | RW | int (-1 - 1) |
| TCP_ENO_SESSID | R | bytes |
| TCP_ENO_NEGSPEC | R | int (32 - 127) |
| TCP_ENO_SPECS | RW | bytes |
| TCP_ENO_SELF_GOPT | RW | int (0 - 31) |
| TCP_ENO_PEER_GOPT | R | int (0 - 31) |
| TCP_ENO_ROLE | R | int (0 - 1) |
| TCP_ENO_LOCAL_NAME | R | bytes |
| TCP_ENO_PEER_NAME | R | bytes |
| TCP_ENO_RAW | RW | bytes |
| TCP_ENO_TRANSCRIPT | R | bytes |

Table 1: Suggested per-connection options

The socket options must return errors under certain circumstances. These errors are mapped to three suggested error codes shown in Table 2. Systems based on sockets already have constants for these errors. Non-socket systems should use existing error codes corresponding to the same conditions. "EINVAL" is the existing error returned when attempting to set options or otherwise operate on a closed socket. "EISCONN" corresponds to calling connect a second time, while "ENOTCONN" corresponds to requesting the peer address of an unconnected socket.

| Symbol | Description |
| --- | --- |
| EINVAL | General error signifying bad parameters |
| EISCONN | Option no longer valid because connection established |
| ENOTCONN | Option not (yet) valid because no connection established |

Table 2: Suggested error codes

TCP_ENO_ENABLED

When set to 0, completely disables TCP-ENO regardless of any other socket option settings except "TCP_ENO_RAW". When set to 1, enables TCP-ENO. If set to -1, use a system-wide default determined at the time of an "accept" or "connect" system call, as described in Section 2.2. This option must return an error ("EISCONN") after a SYN segment has already been sent.

TCP_ENO_SESSID

Returns the session ID of the connection, as defined by the encryption spec in use. This option must return an error if encryption is disabled ("EINVAL"), the connection is not yet established ("ENOTCONN"), or the transport layer does not implement the negotiated spec ("EINVAL").

TCP_ENO_NEGSPEC
Returns a byte in which the lower 7 bits correspond to the spec identifier of the negotiated encryption spec for the current connection, and the high bit is 1 if there was suboption data present. As defined by TCP-ENO, the negotiated spec is the last valid suboption in host B's SYN segment. This option must return an error if encryption is disabled ("EINVAL") or the connection is not yet established ("ENOTCONN").

TCP_ENO_SPECS
Allows the application to specify an ordered list of encryption specs different from the system default list. If the list is empty, TCP-ENO is disabled for the connection. Each byte in the list specifies one suboption type from 0x20-0xff. The list contains no suboption data for variable-length suboptions, only the one-byte spec identifier. The high bit ("v") in these bytes is ignored unless future implementations of encryption specs assign it special meaning. The order of the list matters only for the host playing the "B" role. Implementations must return an error ("EISCONN") if an application attempts to set this option after the SYN segment has been sent. Implementations should return an error ("EINVAL") if any of the bytes are below 0x20 or are not implemented by the TCP stack.

TCP_ENO_SELF_GOPT
The value is an integer from 0-31 specifying the 5-bit value of the general suboption. The default value should initially be 0, but the low bit must be forced to 1 if the application configures the connection for a passive open.

TCP_ENO_PEER_GOPT
The value is an integer from 0-31 reporting the value of the general suboption in the peer's connection.

TCP_ENO_ROLE
The value is a bit (0 or 1). TCP-ENO defines two roles, "A" and "B", for the two ends of a connection. After a normal three-way handshake, the active opener is "A" and the passive opener is "B". Simultaneous open uses the role-override bit to assign unique roles. This option returns 0 when the local host has the "A" role, and 1 when the local host has the "B" role. This call must

return an error before the connection is established ("ENOTCONN") or if TCP-ENO has failed ("EINVAL").

TCP_ENO_LOCAL_NAME
Returns the concatenation of the TCP_ENO_ROLE byte and the TCP_ENO_SESSID. This provides a unique name for the local end of the connection.

TCP_ENO_PEER_NAME
Returns the concatenation of the negation of the TCP_ENO_ROLE byte and the TCP_ENO_SESSID. This is the same value as returned by TCP_ENO_LOCAL_NAME on the other host, and hence provides a unique name for the remote end of the connection.

TCP_ENO_RAW
This option is for use by library-level implementations of encryption specs. It allows applications to make use of the TCP-ENO option, potentially including encryption specs not supported by the transport layer, and then entirely bypass any TCP-level encryption so as to encrypt above the transport layer. The default value of this option is a 0-byte vector, which disables RAW mode. If the option is set to any other value, it disables all other socket options described in this section except for TCP_ENO_TRANSCRIPT.

The value of the option is a raw ENO option contents (without the kind and length) to be included in the host's SYN segment. In raw mode, the TCP layer considers negotiation successful when the two SYN segments both contain a suboption with the same encryption spec value "cs" >= 0x20. For an active opener in raw mode, the TCP layer automatically sends a two-byte minimal ENO option when negotiation is successful. Note that raw mode performs no sanity checking on the "v" bits or any suboption data, and hence provides slightly less flexibility than a true TCP-level implementation.

TCP_ENO_TRANSCRIPT
Returns the negotiation transcript as specified by TCP-ENO. Implementations must return an error if negotiation failed ("EINVAL") or has not yet completed ("ENOTCONN").

## 2.2. Global options

In addition to these per-socket options, implementations should use a global configuration mechanism to allow administrators to configure a default value for "TCP_ENO_SPECS", as well as default behavior for when "TCP_ENO_ENABLED" is -1. These defaults can be system-wide, or per network namespace on systems that provide network namespaces.

Table 3 provides a table of suggested parameters. The type "words" corresponds to a list of 16-bit unsigned words representing TCP port numbers (similar to the "baddynamic" sysctls that, on some operating systems, blacklist automatic assignment of particular port numbers).

| Name | Type |
|---|---|
| eno_specs | bytes |
| eno_enable_connect | int (0 - 1) |
| eno_enable_listen | int (0 - 1) |
| eno_bad_connect_ports | words |
| eno_bad_listen_ports | words |

Table 3: Suggested global parameters

"eno_specs" is simply a string of bytes, and provides the default value for the "TCP_ENO_SPECS" socket option. If "TCP_ENO_SPECS" is non-empty, the remaining sysctls determine whether to attempt TCP-ENO negotiation when the "TCP_ENO_ENABLED" option is -1 (the default), using the following rules.

- On active openers: If "eno_enable_connect" is 0, then TCP-ENO is disabled. If the remote port number is in "eno_bad_connect_ports", then TCP-ENO is disabled. Otherwise, the host attempts to use TCP-ENO.

- On passive openers: If "eno_enable_listen" is 0, then TCP-ENO is disabled. Otherwise, if the local port is in "eno_bad_listen_ports", then TCP-ENO is disabled. Otherwise, if the host receives an SYN segment with an ENO option containing compatible encryption specs, it attempts negotiation.

Because initial deployment may run into issues with middleboxes or incur slowdown for unnecessary double-encryption, sites may wish to blacklist particular ports. For example setting "eno_bad_connect_ports" to 443,993 would disable ENO encryption on outgoing connections to ports 443 and 993 (which use application-layer encryption for HTTP and IMAP, respectively). If the per-socket "TCP_ENO_ENABLED" is not -1, it overrides the sysctl values.

Similarly, on a server, setting "eno_bad_listen_ports" to 443 makes it possible to disable TCP-ENO for incoming HTTPS connection without modifying the web server to set "TCP_ENO_ENABLED" to 0.

## 3. Example API mappings

Section 2 presented abstract APIs for per-connection and global options. One implementation strategy would be to map these APIs to existing per-socket and global configuration mechanisms. By way of example, this section describes a way to map the per-connection settings to BSD sockets options and global configuration to the Unix "sysctl" interface.

### 3.1. Socket options for per-connection settings

Systems with sockets can allow applications to configure TCP-ENO through the same mechanism they use for other TCP connection configuration such as "TCP_NODELAY" [RFC0896], namely the "getsockopt" and "setsockopt" system calls shown in Figure 1.

```
int getsockopt(int socket, int level, int option_name,
               void *option_value, socklen_t *option_len);

int setsockopt(int socket, int level, int option_name,
               const void *option_value, socklen_t option_len);
```

Figure 1: Socket option API

Socket-based TCP-ENO implementations can define a set of new "option_name" values accessible at "level" "IPPROTO_TCP" (generally defined as 6, to match the IP protocol field), where each entry in Table 1 corresponds to a unique "option_name" constant.

### 3.2. Setting System-wide options with sysctl

User-level implementations of TCP-ENO can use a configuration file to set global options. However, such an approach may be awkward for kernel-based implementations. Instead, kernel-level implementations can use the "sysctl" configuration tool. With this approach, TCP-ENO parameters should be placed alongside most TCP parameters. For example, on BSD derived systems a suitable name would be "net.inet.tcp.eno.specs", while on Linux a more appropriate name would be "net.ipv4.tcp_eno_specs".

## 4. Examples

This section provides examples of how applications might authenticate session IDs. Authentication requires exchanging messages over the TCP connection, and hence is not backwards compatible with existing application protocols. To fall back to opportunistic encryption in the event that both applications have not been updated to authenticate the session ID, TCP-ENO provides the application-aware

bits. To signal it has been upgraded to support application-level authentication, an application should set "TCP_ENO_SELF_AWARE" to 1 before opening a connection. An application should then check that "TCP_ENO_PEER_AWARE" is non-zero before attempting to send authenticators that would otherwise be misinterpreted as application data.

### 4.1. Cookie-based authentication

In cookie-based authentication, a client and server both share a cryptographically strong random or pseudo-random secret known as a "cookie". Such a cookie is preferably at least 128 bits long. To authenticate a session ID using a cookie, each host computes and sends the following value to the other side:

```
authenticator = PRF(cookie, local-name)
```

Here "PRF" is a pseudo-random function such as HMAC-SHA-256 [RFC6234]. "local-name" is the result of the "TCP_ENO_LOCAL_NAME" socket option. Each side must verify that the other side's authenticator is correct. To do so, software obtains the remote host's local name via the "TCP_ENO_PEER_NAME" socket option. Assuming the authenticators are correct, applications can rely on the TCP-layer encryption for resistance against active network attackers.

Note that if the same cookie is used in other contexts besides session ID authentication, appropriate domain separation must be employed, such as prefixing "local-name" with a unique prefix to ensure "authenticator" cannot be used out of context.

### 4.2. Signature-based authentication

In signature-based authentication, one or both endpoints of a connection possess a private signature key the public half of which is known to or verifiable by the other endpoint. To authenticate itself, the host with a private key computes the following signature:

```
authenticator = Sign(PrivKey, local-name)
```

The other end verifies this value using the corresponding public key. Whichever side validates an authenticator in this way knows that the other side belongs to a host that possesses the appropriate signature key.

Once again, if the same signature key is used in other contexts besides session ID authentication, appropriate domain separation should be employed, such as prefixing "local-name" with a unique prefix to ensure "authenticator" cannot be used out of context.

## 5. Security considerations

The TCP-ENO specification discusses several important security considerations that this document incorporates by reference. The most important one, which bears reiterating, is that until and unless a session ID has been authenticated, TCP-ENO is vulnerable to an active network attacker, through either a downgrade or active man-in-the-middle attack.

Because of this vulnerability to active network attackers, it is critical that implementations return appropriate errors for socket options when TCP-ENO is not enabled. Equally critical is that applications must never use these socket options without checking for errors.

Applications with high security requirements that rely on TCP-ENO for security must either fail or fall back to application-layer encryption if TCP-ENO fails or session IDs authentication fails.

## 6. Acknowledgments

This work was funded by DARPA CRASH under contract #N66001-10-2-4088.

## 7. References

### 7.1. Normative References

[I-D.ietf-tcpinc-tcpeno]
Bittau, A., Boneh, D., Giffin, D., Handley, M., Mazieres, D., and E. Smith, "TCP-ENO: Encryption Negotiation Option", draft-ietf-tcpinc-tcpeno-01 (work in progress), February 2016.

### 7.2. Informative References

[RFC0896] Nagle, J., "Congestion Control in IP/TCP Internetworks", RFC 896, DOI 10.17487/RFC0896, January 1984, <http://www.rfc-editor.org/info/rfc896>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <http://www.rfc-editor.org/info/rfc6234>.

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