Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 4, 2017

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

October 31, 2016

Interface Extensions for TCP-ENO and tcpcrypt
draft-ietf-tcpinc-api-01

Abstract

TCP-ENO and tcpcrypt perform encryption at the transport layer. They also define a few parameters that are intended to be used or configured by applications. This document specifies operating system interfaces for access to these 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, but the bit is 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 tcpcrypt protocol [I-D.ietf-tcpinc-tcpcrypt] may be negotiated via TCP-ENO, and can operate without configuration. But users may wish to control a few operational details of the protocol:

- Users or system administrators may wish to specify which symmetric ciphers they accept or prefer, or to inspect which cipher has been negotiated for a particular connection. (The key-exchange schemes used by tcpcrypt may be configured via the TCP-ENO API.)

- If connection tampering has been detected via session authentication failure, it may be prudent to purge cached session keys.

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

2. TCP-ENO 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, 0, or 1)
TCP_ENO_SESSID	R	bytes
TCP_ENO_NEGSPEC	R	int (32-127, 160-255)
TCP_ENO_SPECS	RW	bytes
TCP_ENO_SELF_GOPT	RW	int (0-31)
TCP_ENO_PEER_GOPT	R	int (0-31)
TCP_ENO_AA_MANDATORY	RW	int (0 or 1)
TCP_ENO_ROLE	R	int (0 or 1)
TCP_ENO_SELF_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

Unless otherwise specified, all of the read-only options (R) return an error if ENO is disabled ("EINVAL"), the connection is not yet established ("ENOTCONN"), the connection is established and ENO

failed to negotiate encryption ("EINVAL"), or the connection is in raw mode ("EINVAL").

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. When set to -1, uses 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.

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 TEP suboption in the SYN segment sent by host "B".

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 0x20-0x7f (32-127). For future extensibility, the high bit ("v") in these bytes should be set to 0 by applications and ignored by implementations. 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, are between 0x80-0xa0, or are not implemented by the TCP stack.

TCP_ENO_SELF_GOPT

Sets the 5-bit value of the hosts global suboption. The default value should initially be 0. In accordance the ENO specification, regardless of any value set by the application, the least significant bit--termed the _passive role bit_--is forced to 1 when a connection is configured for passive open (i.e., following a "listen" call). Implementations must return an error ("EISCONN") if an application attempts to set this option after a SYN segment has been sent.

TCP_ENO_AA_MANDATORY

If set to 1, enables mandatory application-aware mode in which the local host will disable TCP-ENO unless the remote host has set the application-aware bit (the second-least significant bit in its global suboption). The default value is 0. Implementations must return an error ("EISCONN") if an application attempts to set this option after a SYN segment has been sent.

TCP_ENO_PEER_GOPT

Returns an integer from 0-31 reporting the value of the global suboption in the peer's SYN segment.

TCP_ENO_ROLE

Returns 0 on host "A" and 1 on host "B", according to the roles defined by TCP-ENO. When successful, the value is always equal to the least significant bit of the value returned by TCP_ENO_SELF_GOPT.

TCP_ENO_SELF_NAME

Returns one byte containing the value of TCP_ENO_ROLE (0 or 1) prepended to the TCP_ENO_SESSID, thereby providing a unique name for the local end of the connection.

TCP_ENO_PEER_NAME

Like TCP_ENO_SELF_NAME, but negates the first byte, thereby providing a unique name for the remote end of the connection. (When successful, TCP_ENO_SELF_NAME at one end of a connection should always equal TCP_ENO_PEER_NAME at the other.)

TCP_ENO_RAW

This option is for use by library-level TEP implementations. 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. Unlike any of the other read-only options, this option works in conjunction with TCP_ENO_RAW to allow application-layer encryption to determine what was negotiated.

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. tcpcrypt API extensions

This section recommends further extensions to the API set forth in Section 2 that are specific to the tcpcrypt TEP. Future TEPs may similarly provide TEP-specific options.

3.1. Per-connection options

Option name	RW	Type
TCP_CRYPT_CONF	R	int (0-255)
TCP_CRYPT_CACHE_FLUSH	W	int (1)
TCP_CRYPT_ACONF	RW	bytes
TCP_CRYPT_BCONF	RW	bytes

Table 4: Suggested per-connection tcpcrypt-specific options

Table 4 summarizes the proposed tcpcrypt-specific per-connection options.

TCP_CRYPT_CONF
Returns the one-byte specifier for the authenticated encryption algorithm in use by the connection.

TCP_CRYPT_CACHE_FLUSH
Setting this option to the value 1 wipes cached session keys as specified in section "Session caching" of [I-D.ietf-tcpinc-tcpcrypt]. Useful if application-level authentication discovers a man in the middle attack, to prevent the next connection from using session caching.

TCP_CRYPT_ACONF
Set of allowed symmetric ciphers (AEAD algorithms) this host advertises in "Init1" messages. These bytes are encoded exactly

as the bytes "sym-cipher0 ... sym-cipherK" in section "Key exchange messages" of [I-D.ietf-tcpinc-tcpcrypt]; that is, each is one of the "sym-cipher" bytes from the table of AEAD algorithms. The order of these bytes is immaterial.

TCP_CRYPT_BCONF

Order of preference of symmetric ciphers. These bytes are encoded in the same way as for "TCP_CRYPT_ACONF" above, except they indicate the decreasing order of preference used to determine which "sym-cipher" value to choose when sending an "Init2" message.

3.2. Global options

Name	Type
crypt_aconf	bytes
crypt_bconf	bytes

Table 5: Suggested tcrypt-specific global parameters

System administrators should also be able to set defaults for the per-socket connection parameters. Table 5 lists the system-wide parameters for doing so, which can exist along side the general ENO parameters described in Table 3.

4. Example API mappings

The previous sections 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.

4.1. 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.

4.2. 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".

5. 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 bit. To signal it has been upgraded to support application-level authentication, an application should set the second-least significant bit of "TCP_ENO_SELF_GOPT" before opening a connection. An application should then check that "TCP_ENO_PEER_GOPT" has this bit set before attempting to send authenticators that would otherwise be misinterpreted as application data.

5.1. 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 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.

5.2. 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.

6. Security considerations

The TCP-ENO specification [I-D.ietf-tcpinc-tcpeno] 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 as suggested in this document for socket options when TCP-ENO is not enabled. An example of an API design with potentially catastrophic consequences would be to attempt to communicate TCP-ENO failure by successfully returning a zero-length or zero-valued session ID. 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 ID authentication fails.

7. 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, Kyle Rose, and Joe Touch. This work was partially funded by DARPA CRASH and the Stanford Secure Internet of Things Project.

8. References

8.1. Normative References

[I-D.ietf-tcpinc-tcpcrypt]
Bittau, A., Boneh, D., Giffin, D., Hamburg, M., Handley, M., Mazieres, D., Slack, Q., and E. Smith, "Cryptographic protection of TCP Streams (tcpcrypt)", draft-ietf-tcpinc-tcpcrypt-03 (work in progress), October 2016.

[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-06 (work in progress), October 2016.

8.2. 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
Google
345 Spear Street
San Francisco, CA 94105
US

Email: bittau@google.com

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