Network Working Group
Internet-Draft
Intended status: Informational
Expires: February 11, 2016

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

August 10, 2015

Interface Extensions for TCPINC
draft-bittau-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 for 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

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 February 11, 2016.

Table of Contents

1. Introduction

The TCP Encryption Negotiation Option (TCP-ENO) [I-D.bittau-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:

- 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 a two-bit "application-aware" status for

this purpose, but this status is not accessible through existing interfaces.

- Applications employing TCP's simultaneous open feature need a way to supply a symmetry-breaking "tie-breaker" bit to 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

Application should access TCP-ENO options through the same mechanism they use to access other TCP configuration options, such as "TCP_NODELAY" [RFC0896]. With the popular sockets API, this mechanism consists of two socket options, "getsockopt" and "setsockopt", shown in Figure 1. Socket-based TCP-ENO implementations should define a set of new "option_name" values accessible at "level" "IPPROTO_TCP" (generally defined as 6, to match the IP protocol field).

```
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

Table 1 summarizes the new "option_name" arguments that TCP-ENO introduces to the socket option (or equivalent) system calls. 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
TCPENO_ENABLED	RW	int (-1 - 1)
TCPENO_SESSID	R	bytes
TCPENO_NEGSPEC	R	int (32 - 255)
TCPENO_SPECS	RW	bytes
TCPENO_SELF_AWARE	RW	int (0 - 3)
TCPENO_PEER_AWARE	R	int (0 - 3)
TCPENO_TIEBREAKER	RW	int (0 - 1)
TCPENO_ROLE	R	int (0 - 1)
TCPENO_RAW	RW	bytes
TCPENO_TRANSCRIPT	R	bytes

Table 1: Suggested new IPPROTO_TCP socket options

TCPENO_ENABLED When set to 0, completely disables TCP-ENO regardless of any other socket option settings except "TCPENO_RAW". When set to 1, enables TCP-ENO. When set to -1 (which should be the default), uses a system default value to determine whether or not to enable TCP-ENO. This option must return an error after a SYN segment has already been sent.

TCPENO_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, the connection is not yet established, or the transport layer does not implement the negotiated spec.

TCPENO_NEGSPEC Returns the negotiated encryption spec for the current connection. As defined by TCP-ENO, the negotiated spec is the first valid suboption in the "B" host's SYN segment (without any suboption data for variable-length suboptions). This option must return an error if encryption is disabled or the connection is not yet established.

TCPENO_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 order of the list matters only for the host playing the "B" role. Implementations must return an error if an application attempts to set this option after the SYN segment has been sent. Implementations should return an error if any of the bytes are below 0x20 or are not implemented by the TCP stack.

TCPENO_SELF_AWARE The value is an integer from 0-3, allowing applications to specify the "aa" bits in the general suboption sent by the host. When listening on a socket, the value of this option applies to each accepted connection. Implementations must return an error if an application attempts to set this option after a SYN segment has been sent.

TCPENO_PEER_AWARE The value is an integer from 0-3 reporting the "aa" bits in the general suboption of the peer's segment. Implementations must return an error if an application attempts to read this value before the connection is established.

TCPENO_TIEBREAKER The value is a bit (0 or 1), indicating the value of the "b" bit to set in the host's general suboption. The "b" bit breaks the symmetry of simultaneous open to assign a unique role "A" or "B" to each end of the connection. The host that sets the "b" bit assumes the "B" role (which in non-simultaneous open is assigned to the passive opener). Implementations must return an error for this options after the SYN segment has already been sent.

TCPENO_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 tie-breaker 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 or if TCP-ENO has failed.

TCPENO_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 TCPENO_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.

TCPENO_TRANSCRIPT Returns the negotiation transcript as specified by TCP-ENO. Implementations must return an error if negotiation failed or has not yet completed.

In addition to these per-socket options, implementations should use "sysctl" or an equivalent mechanism to allow administrators to configure system-wide defaults for "TCPENO_ENABLED" and "TCPENO_SPECS". These parameters should be named "eno_enabled" and "eno_specs" and placed alongside most TCP parameters. For example, on BSD derived systems a suitable name would be "net.inet.tcp.eno_enabled" and "net.inet.tcp.eno_specs", while on Linux more appropriate names would be "net.ipv4.tcp_eno_enabled" and "net.ipv4.tcp_eno_specs".

Because initial deployment may run into issues with middleboxes or incur slowdown for unnecessary double-encryption, implementations should also allow ENO to be blacklisted for particular local and remote ports, via sysctl on "net.inet.tcp.eno_bad_localport" and "net.inet.tcp.eno_bad_remoteport" (or the equivalent under "net.ipv4" for linux), both of which consist of a list of TCP port numbers on which to disable TCP-ENO by default. For example the following command:

```
sysctl net.inet.tcp.eno_bad_remoteport=443,993
```

would disable ENO encryption on outgoing connections to ports 443 and 993 (which use application-layer encryption for TLS and IMAP, respectively).

The per-socket "TCPENO_ENABLED" option, if not -1, should override both the "eno_enabled" and port-range sysctls.

3. Automatic configuration protocol

TCP-ENO is designed to fail by reverting to unencrypted TCP. Such behavior is necessary for incremental deployment, and is no worse than the status quo in which there is no TCP-layer encryption. However, one outcome worse than the status quo would be to for TCP-ENO connections to fail completely where unenecrypted connections would work. Fortunately, if TCP-ENO is not supported by both endpoints, or if middleboxes strip the ENO option from packets, then implementations simply revert to unencrypted TCP upon receiving a SYN or initial ACK segment without an ENO option. This fallback approach also applies to interception proxies [RFC3040], which typically

terminate TCP connections and hence will not include ENO in their SYN segments if they do not know about it.

However, given that the goal of TCP-ENO is to encrypt previously plaintext traffic, there is always the possibility that a middlebox performing deep packet inspection could shut down a connection because the ciphertext does not resemble an expected higher-level application protocol such as HTTP. Such middleboxes would cause TCP-ENO connections to fail. Systems may wish to probe the network so as to enable TCP-ENO only in places where middleboxes do not induce such failures.

A precedent for probing middlebox behavior is the STUN protocol [RFC5389], which applications use to characterize NAT. STUN relies on having a known, publicly-accessible server beyond any locally administered middleboxes. STUN is typically invoked by applications that require peer-to-peer communication to decide whether they can accept incoming connections. For TCP-ENO, which affects all TCP connections, it makes more sense to probe for network compatibility at the time network interfaces are configured by DHCP [RFC2131], stateless address autoconfiguration [RFC4862], or other mechanisms. Many DHCP implementation already provide hooks through which such probes can be configured.

Like STUN, TCP-ENO probing requires a known external server running an agreed upon protocol. We suggests using HTTP as the protocol, and responding to the path "/tcp-eno/session-id" with a response of type "text/plain". Upon successful TCP-ENO negotiation, servers should reply with the string "encrypted " followed by a lower-case hexadecimal encoding of the tcpcrypt session ID followed by a newline. For connection on which TCP-ENO fails, the same path should return the string "unencrypted\n" (with no session ID). If such a request works with TCP-ENO disabled but hangs or resets with TCP-ENO enabled, then TCP-ENO should be disabled for the host. Otherwise, if probes succeed, even if they return "unencrypted", TCP-ENO should be enabled (for the possible benefit of local connections), as middleboxes may simply be stripping off the option.

Hosts should perform the above probe twice, using both port 80 and a different port, we suggest 8080, on the same server. Given the prevalence of interception proxies on port 80, port 80 may experience entirely different failure modes from other ports. If the port 80 probe fails, TCP-ENO should be disabled for port 80. If the other probe fails, TCP-ENO should be disabled entirely.

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, applications should set "TCPENO_SELF_AWARE" to 1 before opening a connection. An application should then check that "TCPENO_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 computes and sends the following value to the other side:

```
authenticator = PRF(cookie, role || session-ID)
```

Here "PRF" is a psueo-random function such as HMAC-SHA-256 [RFC6234]. "role" is the byte 0 or 1, as returned by the "TCPENO_ROLE" socket options. "session-ID" is the session ID returned by the "TCPENO_SESSID" session ID. The symbol "||" denotes concatenation. Each side must verify that the other side's authenticator is correct. 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 should be employed, such as prefixing "role || session-ID" 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, role || session-ID)
```

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 "role || session-ID" 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 fallback 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.bittau-tcpinc-tcpeno]
Bittau, A., Boneh, D., Giffin, D., Handley, M., Mazieres, D., and E. Smith, "TCP-ENO: Encryption Negotiation Option", draft-bittau-tcpinc-tcpeno-01 (work in progress), August 2015.

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>.

[RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, DOI 10.17487/RFC2131, March 1997, <http://www.rfc-editor.org/info/rfc2131>.

[RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web Replication and Caching Taxonomy", RFC 3040, DOI 10.17487/RFC3040, January 2001, <http://www.rfc-editor.org/info/rfc3040>.

[RFC4862] Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, DOI 10.17487/RFC4862, September 2007, <http://www.rfc-editor.org/info/rfc4862>.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <http://www.rfc-editor.org/info/rfc5389>.

[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