NAME

ktls - kernel Transport Layer Security

SYNOPSIS

options KERN_TLS

DESCRIPTION

The **ktls** facility allows the kernel to perform Transport Layer Security (TLS) framing on TCP sockets. With **ktls**, the initial handshake for a socket using TLS is performed in userland. Once the session keys are negotiated, they are provided to the kernel via the TCP_TXTLS_ENABLE and TCP_RXTLS_ENABLE socket options. Both socket options accept a *struct tls_enable* structure as their argument. The members of this structure describe the cipher suite used for the TLS session and provide the session keys used for the respective direction.

ktls only permits the session keys to be set once in each direction. As a result, applications must disable rekeying when using **ktls**.

Modes

ktls can operate in different modes. A given socket may use different modes for transmit and receive, or a socket may only offload a single direction. The available modes are:

TCP_TLS_MODE_NONE	**ktls** is not enabled.
TCP_TLS_MODE_SW	TLS records are encrypted or decrypted in the kernel in the socket layer via crypto(9). Typically the encryption or decryption is performed in software, but it may also be performed by co-processors.
TCP_TLS_MODE_IFNET	TLS records are encrypted or decrypted by the network interface card (NIC). In this mode, the network stack does not work with encrypted data. Instead, the NIC encrypts TLS records as they are being transmitted, or decrypts received TLS records before providing them to the host. Network interfaces which support this feature will advertise the TXTLS4 (for IPv4) and/or TXTLS6 (for IPv6) capabilities as reported by ifconfig(8). These capabilities can also be controlled by ifconfig(8). If a network interface supports rate limiting (also known as packet pacing) for TLS offload, the interface will advertise the TXTLS_RTLMT capability.

TCP_TLS_MODE_TOE	TLS records are encrypted by the NIC using a TCP offload engine (TOE). This is similar to TCP_TLS_MODE_IFNET in that the network stack does not work with encrypted data. However, this mode works in tandem with a TOE to handle interactions between TCP and TLS.

Transmit

Once TLS transmit is enabled by a successful set of the TCP_TXTLS_ENABLE socket option, all data written on the socket is stored in TLS records and encrypted. Most data is transmitted in application layer TLS records, and the kernel chooses how to partition data among TLS records. Individual TLS records with a fixed length and record type can be sent by sendmsg(2) with the TLS record type set in a TLS_SET_RECORD_TYPE control message. The payload of this control message is a single byte holding the desired TLS record type. This can be used to send TLS records with a type other than application data (for example, handshake messages) or to send application data records with specific contents (for example, empty fragments).

The current TLS transmit mode of a socket can be queried via the TCP_TXTLS_MODE socket option. A socket using TLS transmit offload can also set the TCP_TXTLS_MODE socket option to toggle between TCP_TLS_MODE_SW and TCP_TLS_MODE_IFNET.

Receive

Once TLS receive is enabled by a successful set of the TCP_RXTLS_ENABLE socket option, all data read from the socket is returned as decrypted TLS records. Each received TLS record must be read from the socket using recvmsg(2). Each received TLS record will contain a TLS_GET_RECORD control message along with the decrypted payload. The control message contains a *struct tls_get_record* which includes fields from the TLS record header. If an invalid or corrupted TLS record is received, recvmsg(2) will fail with one of the following errors:

[EINVAL]	The version fields in a TLS record’s header did not match the version required by the *struct tls_enable* structure used to enable in-kernel TLS.
[EMSGSIZE]	A TLS record’s length was either too small or too large.
[EMSGSIZE]	The connection was closed after sending a truncated TLS record.
[EBADMSG]	The TLS record failed to match the included authentication tag.

The current TLS receive mode of a socket can be queried via the TCP_RXTLS_MODE socket option. At present, the mode cannot be changed.

Sysctl Nodes

ktls uses several sysctl nodes under the *kern.ipc.tls* node. A few of them are described below:

kern.ipc.tls.enable	Determines if new kernel TLS sessions can be created.
kern.ipc.tls.cbc_enable	Determines if new kernel TLS sessions with a cipher suite using AES-CBC can be created.
kern.ipc.tls.sw	A tree of nodes containing statistics for TLS sessions using TCP_TLS_MODE_SW.
kern.ipc.tls.ifnet	A tree of nodes containing statistics for TLS sessions using TCP_TLS_MODE_IFNET.
kern.ipc.tls.toe	A tree of nodes containing statistics for TLS sessions using TCP_TLS_MODE_TOE.
kern.ipc.tls.stats	A tree of nodes containing various kernel TLS statistics.

The *kern.ipc.mb_use_ext_pgs* sysctl controls whether the kernel may use unmapped mbufs. They are required for TLS transmit.

Supported Hardware

The cxgbe(4) and mlx5en(4) drivers include support for the TCP_TLS_MODE_IFNET mode.

The cxgbe(4) driver includes support for the TCP_TLS_MODE_TOE mode.

Supported Libraries

OpenSSL 3.0 and later include support for **ktls**. The *security/openssl-devel* port may also be built with support for **ktls** by enabling the KTLS option. OpenSSL in the base system includes KTLS support when built with WITH_OPENSSL_KTLS.

Applications using a supported library should generally work with **ktls** without any changes provided they use standard interfaces such as SSL_read(3) and SSL_write(3). Additional performance may be gained by the use of SSL_sendfile(3).

IMPLEMENTATION NOTES

ktls assumes the presence of a direct map of physical memory when performing software encryption and decryption. As a result, it is only supported on architectures with a direct map.

SEE ALSO

cxgbe(4), mlx5en(4), tcp(4), src.conf(5), ifconfig(8), sysctl(8), crypto(9)

HISTORY

Kernel TLS first appeared in FreeBSD 13.0.