In-kernel TLS Framing and Encryption for FreeBSD

John Baldwin
BSDCan
June 2020
Overview

- What is KTLS?
- TLS Transmit
- TLS Receive
- Current Status
What is TLS?

• Transport Layer Security (TLS) is an application layer protocol
• Provides authentication and privacy
• Structured as a stream of records, or frames, sent and received over a transport protocol
• Includes handshake messages to negotiate session keys and application data messages to tunnel application data
What is KTLS?

• In-kernel TLS (KTLS) handles TLS framing and encryption/decryption in the kernel

• KTLS does not handle session key negotiation
  – Userland library such as OpenSSL supplies session keys to kernel after handshake
Why KTLS?

Two reasons to handle TLS in the kernel

1. Enable zero-copy send over TLS via sendfile()
2. Support TLS offload in NICs
TLS Sessions

• TLS Sessions describe session keys
  – Ciphersuite (AES-GCM, AES-CBC with HMAC)
  – Cipher and MAC keys

• SSL library provides session keys via setsockopt()

• TLS Sessions are associated with socket buffers
  – Separate sessions for transmit and receive
TLS Transmit

• All data written on a socket using KTLS transmit is encrypted by the kernel
• Userland can send individual TLS records with a specific record type and length via sendmsg() – TLS_SET_RECORD_TYPE control message
• Kernel chooses framing and uses “application data” record type for all other data
TLS Transmit

• TLS records stored in a special type of mbuf
  – TLS header and trailer stored inline in mbuf
  – Payload data referenced via physical address pointers

• Not-yet-encrypted TLS record mbufs hold a reference to a TLS session
  – Session reference inherited from socket buffer
TLS Transmit: SW KTLS

Userland

write()

Application Data

Kernel

Socket Buffer

Unencrypted (M_NOTREADY)

Encrypted

TCP Packet

NIC
TLS Transmit: SW KTLS

Userland
(sendfile())

Kernel

Socket Buffer

Disk

TCP Packet

NIC

Per-socket Copies
TLS Transmit: NIC/TOE KTLS

Userland

Kernel

write() -> Application Data

Socket Buffer

Unencrypted

TCP Packet

NIC
TLS Transmit: NIC/TOE KTLS

Userland

sendfile()

Kernel

Socket Buffer

Disk

No Copies!

TCP Packet

NIC
TLS Receive

- All data received on a socket using KTLS receive is decrypted by the kernel
- Userland receives individual TLS records via recvmsg()
  – TLS_GET_RECORD control message
- Socket buffer holds a list of TLS records like a datagram socket even though TCP is a stream socket
TLS Receive: SW KTLS

Userland

recvmsg()

Application Data

Kernel

Socket Buffer

Decrypted

Encrypted (M_NOTREADY)

TCP Packet

Encrypted

NIC
TLS & Socket Send Buffers

• TLS uses send socket buffer in the “usual” way. It is a single “record” holding a stream of TLS mbufs.
  – Each mbuf describes a single TLS record
  – Unencrypted records are marked as M_NOTREADY
  – Both unencrypted and encrypted TLS records live in the same stream
  – To mark a record as encrypted, clear M_NOTREADY
TLS & Socket Receive Buffers

• TLS uses receive socket buffer differently
  – Decrypted TLS records are stored as “records” in socket buffer consisting of control message mbuf holding TLS header followed by decrypted data in “normal” mbufs. No trailer.
  – Encrypted TLS records received from TCP are just “normal” mbufs with TLS header and trailer data in the mbuf payload
  – Can’t simply flip M_NOTREADY bit to convert from encrypted to decrypted
Decryption TLS Records

- Wait for full TLS record to be received
- Decrypt TLS record payload
- Allocate control message and copy TLS header into message
- Discard TLS header and trailer from “normal” mbufs holding TLS record
- Ensure the mbufs holding TLS record aren’t freed out from under decryption handler via sbcut(), sbdrop(), or sbflush()
- Ensure socket buffer accounting is accurate
Splitting the Receive Buffer

Socket Buffer → sb_mtls → sb_mb
Decrypting a TLS Record

sb_mtls

sb_mtls

sb_mb

sb_mb
TLS Receive: NIC TLS (Sketch)

Decrypted TLS Record

Out of order encrypted data
Current Status: Transmit

• KTLS Transmit for TLS 1.0-1.3 merged to FreeBSD 13.0-CURRENT
  – Includes SW TLS, NIC TLS, TOE TLS
  – ktls_ocf.ko and security/ktls_isa-l_crypto-kmod port/package
• KTLS Transmit for TLS 1.0-1.2 merged to OpenSSL master (will ship in 3.0)
• TLS 1.3 for OpenSSL pending review
  – https://github.com.openssl/openssl/pull/10626
Current Status: Receive

- KTLS Receive for TLS 1.1-1.2 via TOE merged to FreeBSD 13.0-CURRENT
- KTLS Receive for TLS 1.1-1.2 via SW in progress
  - [https://reviews.freebsd.org/D24628](https://reviews.freebsd.org/D24628)
- KTLS Receive for TLS 1.1-1.2 for OpenSSL pending review
  - [https://github.com/openssl/openssl/pull/11679](https://github.com/openssl/openssl/pull/11679)
Current Status: nginx

- nginx patches to support SSL\_sendfile()
  - https://github.com/nginx/nginx/compare/branches/stable-1.16...bsdjhb:ktls-1.16
Further WIP

• Improving KTLS performance using OCF
  – Goal is to bring aesni.ko and ktls_ocf.ko on par with security/ktls_isa-l_crypto-kmod

• Adding support for TLS 1.1 (and maybe 1.0) transmit to SW KTLS via OCF

• Adding support for TLS 1.3 receive to SW KTLS

• Making OpenSSL KTLS available via base or ports

• Adding SSL_sendfile() support to nginx port