In-Kernel TLS Framing and Encryption for FreeBSD

John Baldwin

vBSDCon

September 6, 2019

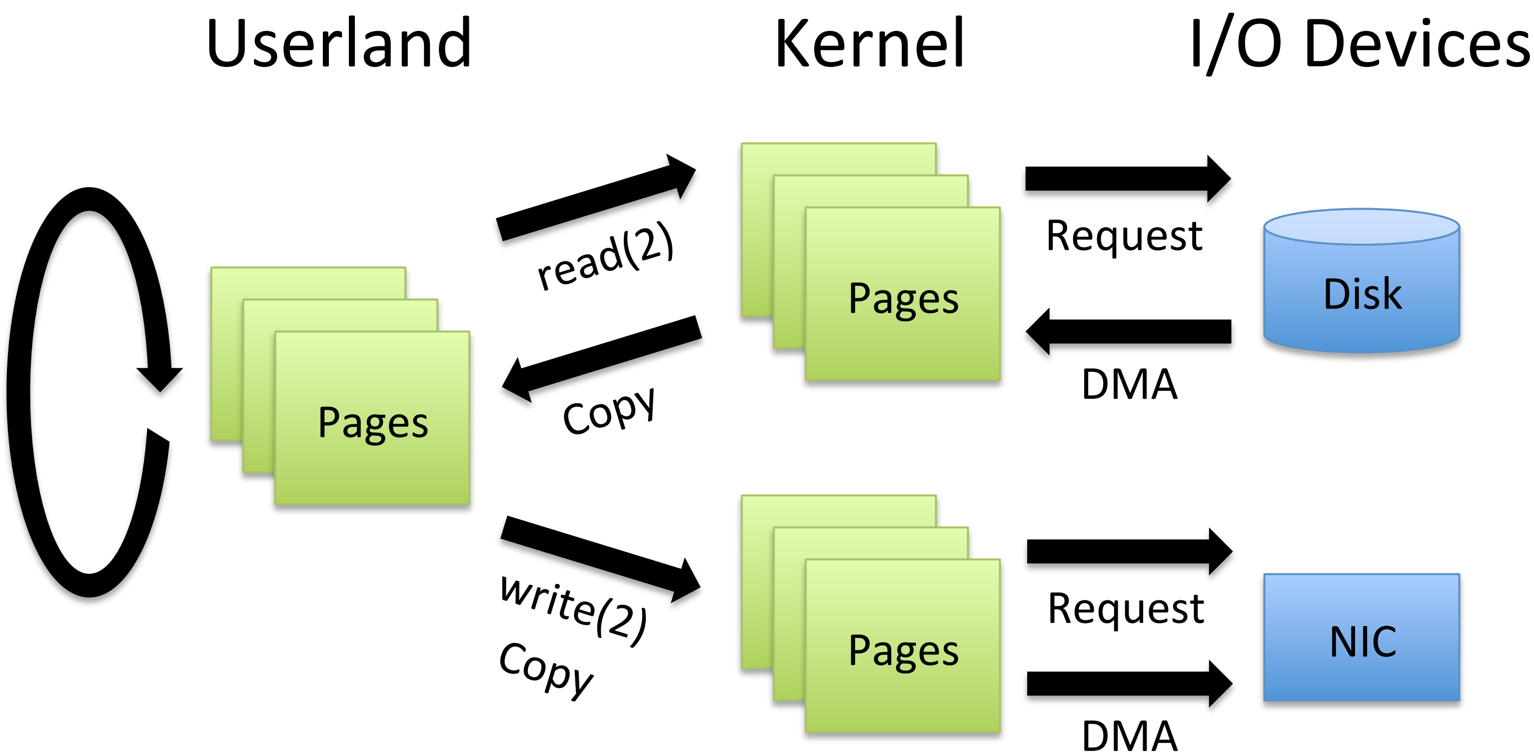
Overview

- Motivation
- Kernel TLS
- Software TLS
- NIC TLS
- Numbers

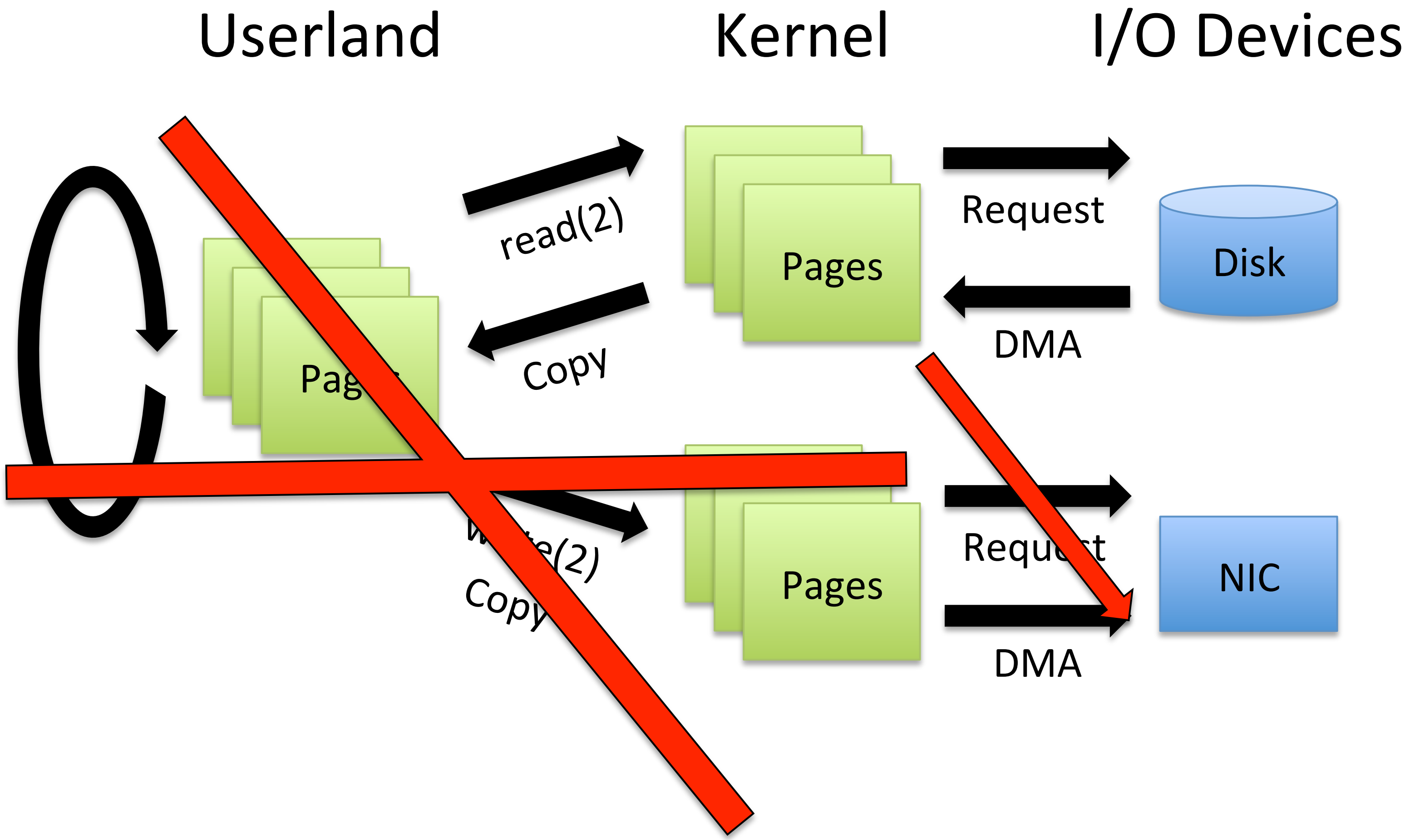
Why KTLS?

- The story of KTLS is really a repeat of the story of `sendfile(2)`
- So let’s start with that…

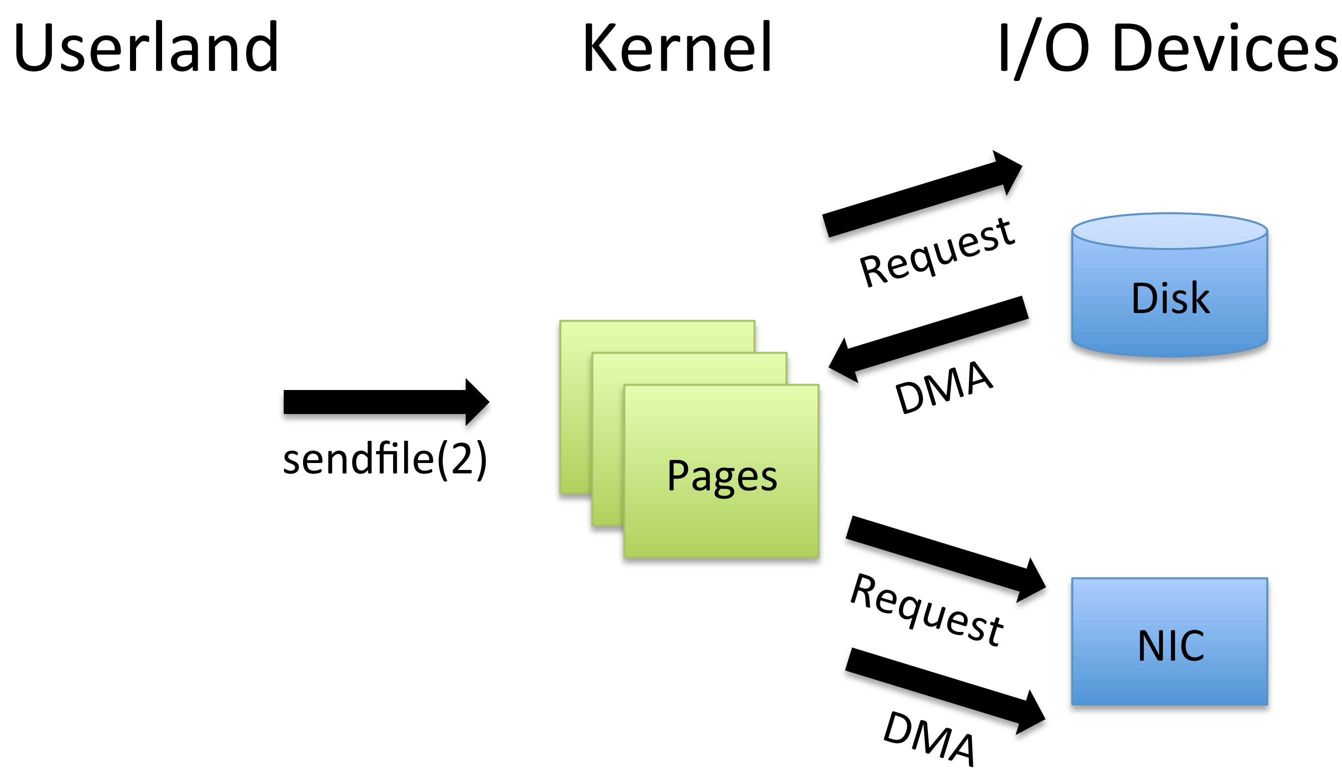
Pre-sendfile(2) HTTP/FTP Workflow



Pre-sendfile(2) HTTP/FTP Workflow

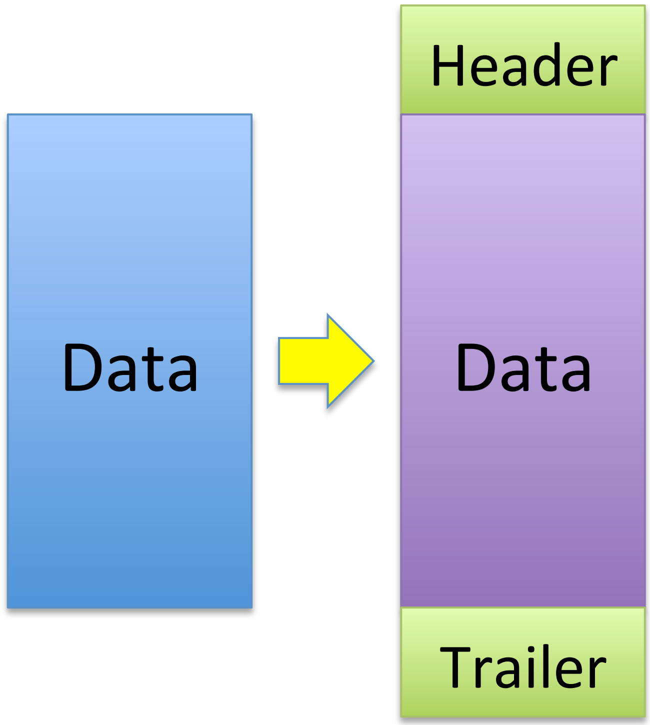


sendfile(2) HTTP/FTP Workflow

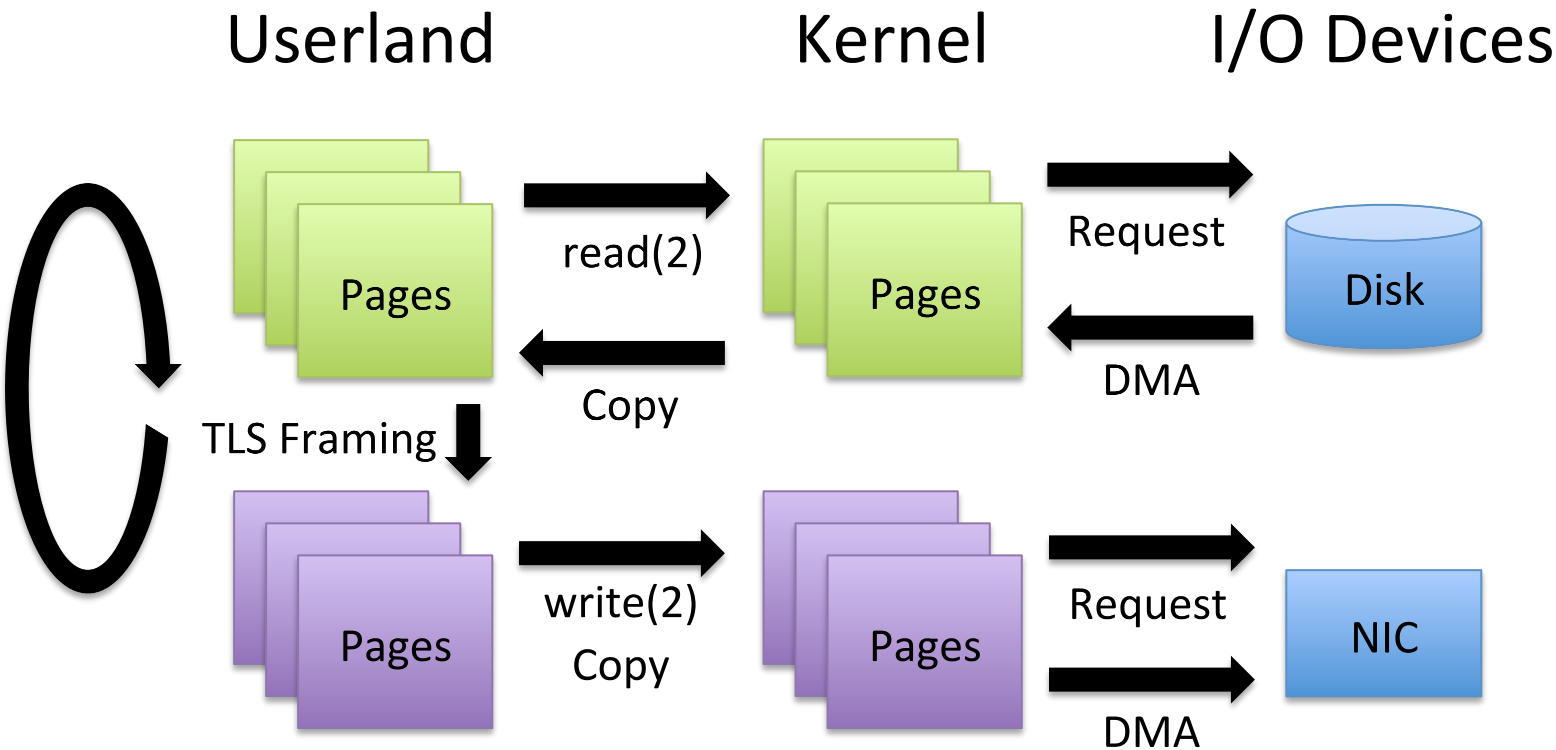


Back to TLS

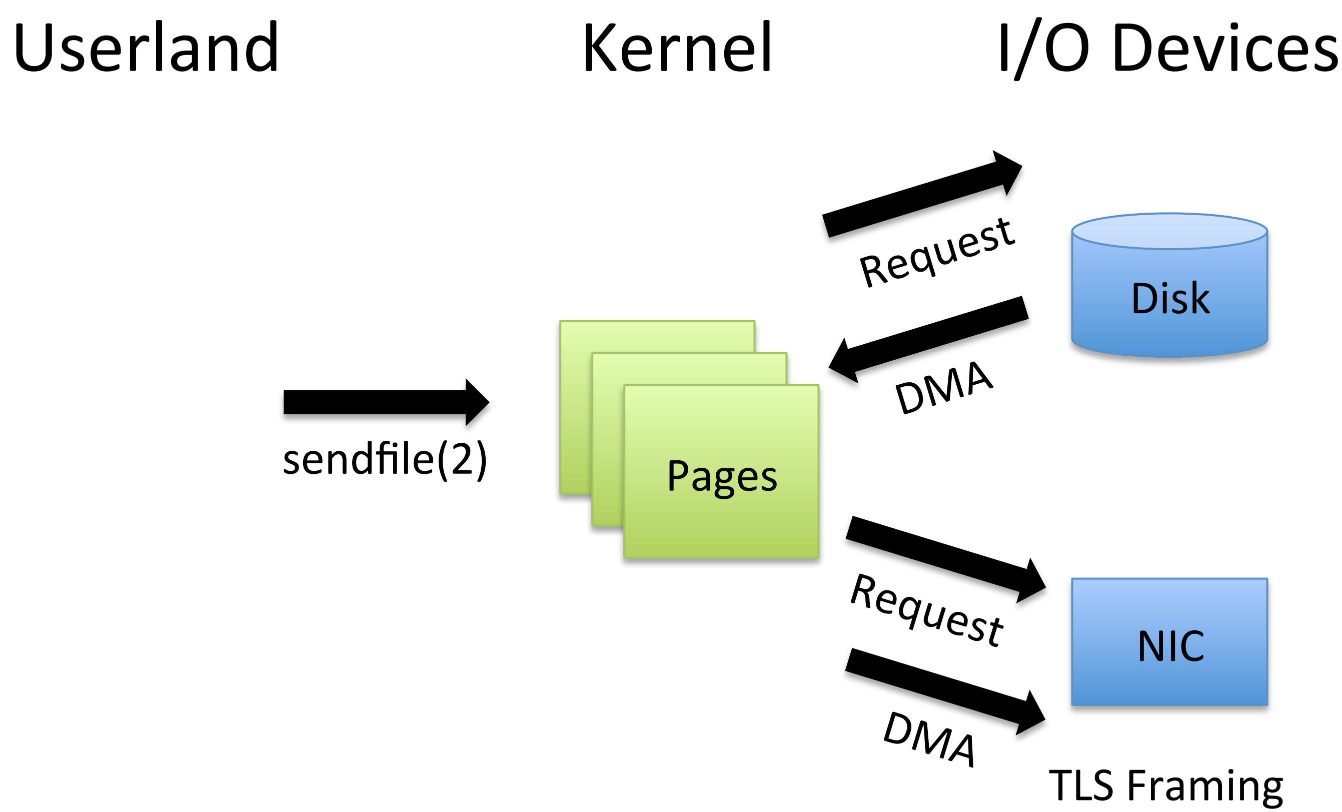
- TLS stores data in TLS records / frames
- Each frame contains
  - Header
  - Encrypted Payload
  - Trailer
- This framing is all currently done in userland (OpenSSL, etc.)



Current HTTPS Workflow



Ideal HTTPS Workflow



KTLS: Towards an Ideal Workflow

- Goal: Use `sendfile(2)` with HTTPS

What is Required?

- Raw file data has to be framed into TLS records in the kernel
- Session parameters (e.g. keys) required for framing
- Ability to send non-application data TLS records (e.g. Alerts)
- Framing overhead included in TCP's sequence space

What is not Required?

- Initial handshake and key negotiation
  - This can be handled in userland as it is now before the bulk data transfer
- Receive Offload
  - For transmit-heavy workloads such as Netflix's, once the handshake is complete, the only receive data is TCP ACKs

KTLS Components

- TLS session objects
- Storing TLS frames in mbufs
- Framing written data
- Software TLS
- NIC TLS

TLS Session Objects

- Holds ciphers used and session keys for those ciphers
- Created in response to `TCP_TXTLS_ENABLE` socket option
- Socket send buffer holds a reference to current TLS session

Storing TLS Frames in mbufs

- Netflix added a new external mbuf type (`EXT_PGS`) to more efficiently handle `sendfile(2)` requests (r349529)
- Each TLS frame is stored in a single `EXT_PGS` mbuf
- KTLS extends `struct mbuf_ext_pgs`
  - Reference to TLS session object
  - TLS header and trailer
  - `m_len` accounts for header and trailer

Framing Written Data

- Once KTLS is enabled, all data written to a socket is stored in TLS frames
- Data is always stored in EXT_PGS mbufs
- mbufs are passed to `ktls_frame()` before being inserted into the socket buffer

Framing Written Data

- Most system calls (`write(2)`, `send(2)`, and `sendfile(2)`) store data in Application Data frames
- sendmsg(2) can send individual TLS records with a different record type
  - Entire buffer is sent as a single TLS record
  - Record type set via `TLS_SET_RECORD_TYPE` control message

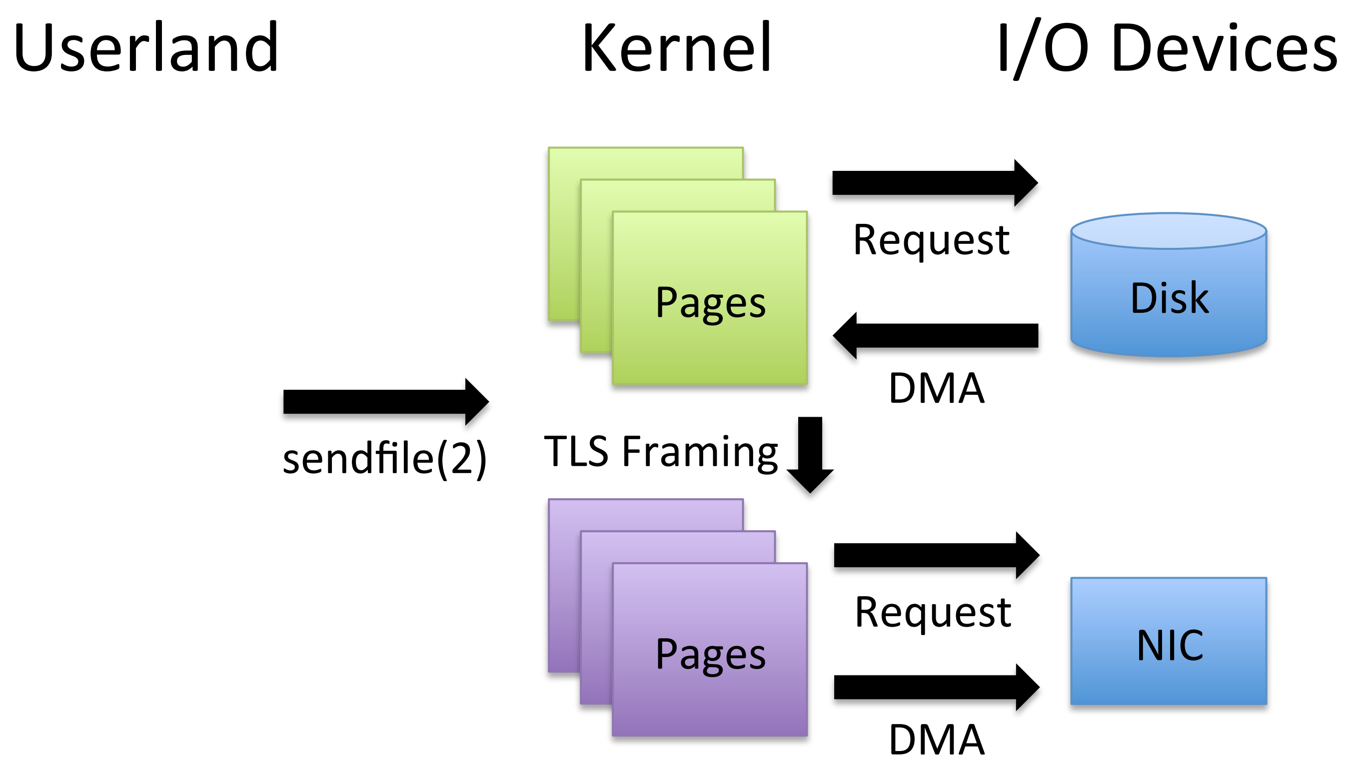
ktls_frame()

- Uses socket send buffer's TLS session reference
- Adds TLS session reference to each mbuf
- Calculates header and trailer lengths and sets `m_len` to length of full frame
  - Includes variable-length padding for AES-CBC
- Populates TLS header including explicit IV

Software TLS

- TLS session object is associated with an encryption backend
- Data is encrypted once while it is in the socket buffer
- Once encrypted, TCP transmits data from socket buffer just like regular data
  - TLS session object reference dropped after encryption

Software TLS Workflow



Software TLS with sendfile(2)

- sendfile(2) allocates EXT_PGS mbufs to hold file data pages
- sendfile_iodone() callback schedules mbufs for encryption instead of marking mbufs ready
- KTLS worker thread allocates pages to hold encrypted copy of data and invokes encryption backend
- Encrypted mbufs marked ready

Software TLS with write(2)

- write(2) allocates EXT_PGS mbufs to hold copy of user's data
- mbufs marked M_NOTREADY and queued for encryption
- KTLS worker thread invokes encryption backend to encrypt in place
- Encrypted mbufs marked ready

Software TLS

- Software TLS avoids kernel <-> userland transitions and reduces number of copies
- CPU is still touching the data
- For sendfile(2), copy into per-socket pages still required

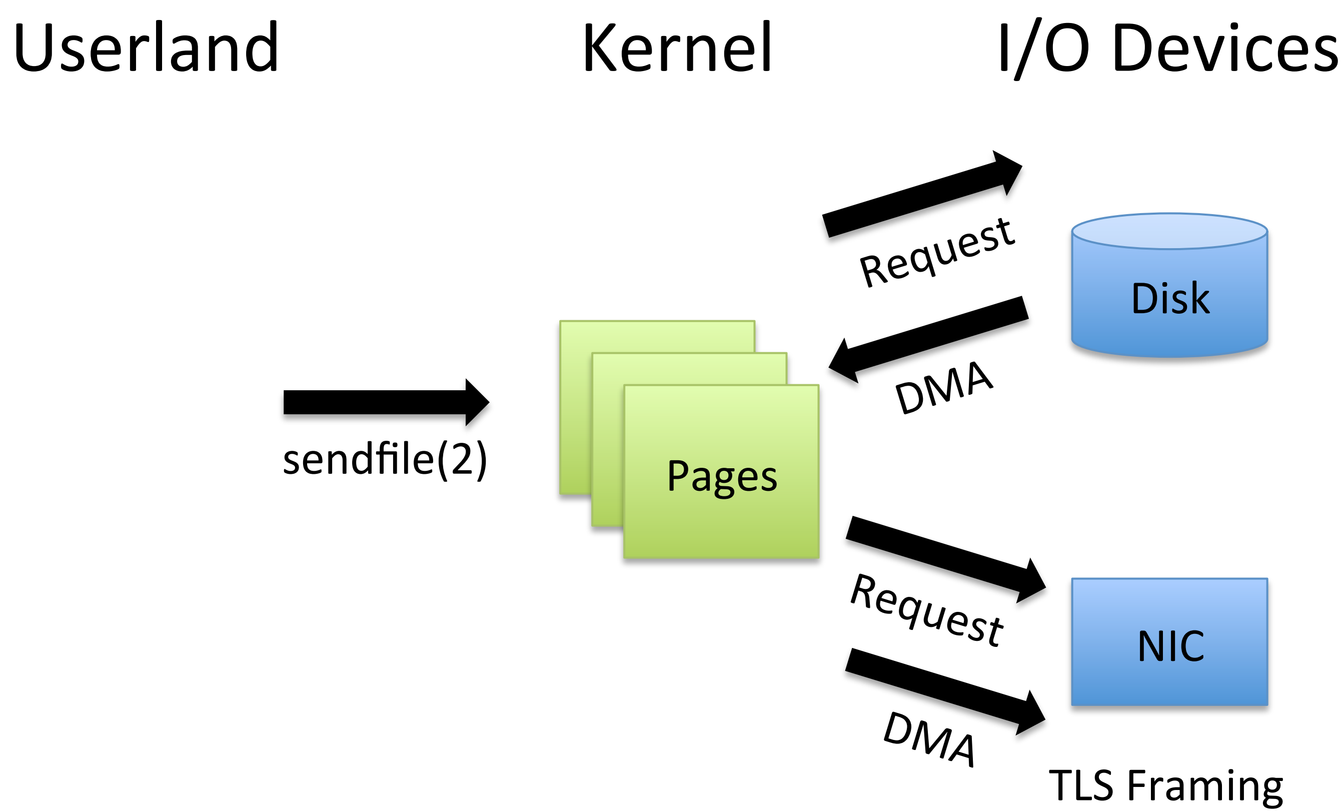
NIC TLS

- TLS sessions allocate a send tag on the associated NIC
  - Send tag holds driver-specific TLS session data
- Socket layer passes unencrypted mbufs to TCP
  - TLS session object reference held until data is ACKed and mbuf is dropped from socket buffer

NIC TLS

- IP output verifies TLS send tag matches NIC
  - Avoids leaking unencrypted data due to route change
  - Builds on r348254
- NIC encrypts TLS frames and splits into TCP segments

NIC TLS Workflow



NIC TLS

- Avoids copies from Software TLS
- CPU no longer touches the data
- Similar workflow to sendfile(2) without TLS

Benchmarking Setup

- Two identical 4-core Intel E5-1620 v3 systems with HTT and Chelsio T6 100 Gbps NICs connected back-to-back
- 16 openssl s_time instances using Chelsio TOE TLS with RX + TX offload on receiver
- nginx 1.14.2 with KTLS patches on server using patched OpenSSL 1.1.1
- AES256-GCM used as the cipher

HTTPS Bandwidth (Gbps)

Mode	1 worker	4 workers
Plain (userland) TLS	7.9	30
KTLS with cryptosoft0	2.9	2.8
KTLS with aesni0	36	36
KTLS with ccr0	36	35
KTLS with Intel ISA-L	48	48
KTLS with Chelsio T6	72	64

Netflix Benchmarks

System	Mode	CPU Usage	Bandwidth (Gbps)
Late 2018 12-core Xeon-D	T6 NIC TLS	62%	90
Late 2018 8-core Xeon-D	T6 NIC TLS	80%	80
2016 16-core Xeon E5v4	T6 NIC TLS	35%	90
2016 16-core Xeon E5v4	ISA-L SW TLS	68%	90

System	Mode	Memory Bandwidth (GB/s)
2016 16-core Xeon E5v4	T6 NIC TLS	30
2016 16-core Xeon E5v4	ISA-L SW TLS	55

Supported Ciphers

- TLS 1.0 – 1.2
- AES-CBC with SHA1 and SHA2-256 HMAC
- AES-GCM
- Backends and NIC drivers might only support a subset
  - ktls_ocf only supports AES-GCM
  - Chelsio T6 NIC TLS supports AES-CBC and AES-GCM, but not TLS 1.0

Where are the bits

- Kernel Framework: r351522
- T6 NIC TLS
  - https://github.com/bsdjhb/freebsd/tree/kern_tls_t6
- Intel ISA-L software backend
  - https://reviews.freebsd.org/D21446

Where are the bits

- OpenSSL patches
  - https://github.com/bsdjhb/openssl
  - 1.1.1 => kern_tls_1_1_1 branch
  - master => ktls_master branch
- nginx patches
  - https://github.com/bsdjhb/nginx
  - OpenSSL 1.1.1 => ktls-1.14 branch
  - OpenSSL master => ktls-1.14-openssl-master

Future Work

- Merging OpenSSL changes upstream
- Updating TOE TLS to use KTLS framework
- TLS RX offload
- TLS 1.2 Encrypt-then-Mac
- TLS 1.3
  - Drew has an initial version

Acknowledgments

- Scott Long and Randall Stewart
  - Initial software TLS work
- Drew Gallatin
  - EXT_PGS mbufs for sendfile
  - Software TLS backend framework
- Myself
  - NIC TLS framework
  - T6 NIC TLS
- Funded by Netflix and Chelsio