Kernel TLS and hardware TLS offload in FreeBSD 13

by

Mellanox, Chelsio and Netflix
Why crypto?

- Bob and Alice and the secret message
- Mathematical dependance on a relatively small pre-shared key
- When used right:
  - Prevents eavesdropping
  - Prevents data tampering
- When used wrong:
  - Makes denial of service easier
What is TLS?

- Transport Layer Security, TLS
- Used behind https:// (TCP port 443)
- Supports multiple crypto codecs among others
  - AES 128B / 256B
- Supports multiple key exchange protocols
  - DiffieHellman, DH
  - Ron Rivest, Adi Shamir, Leonard Adleman, RSA
- Most recent version is v1.3
TLS v1.2

- Layout of a TLS record
- More detailed information at: [https://tls.ulfheim.net/](https://tls.ulfheim.net/)

<table>
<thead>
<tr>
<th>TLS REC(s)</th>
<th>tcp_hdr</th>
<th>ipv4/ipv6 hdr</th>
<th>eth_hdr</th>
<th>uint8_t</th>
<th>tls_type (data, handshake, alert)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>uint8_t</td>
<td>tls_vmajor (3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>tls_vminor (3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint16_t</td>
<td>tls_length (0..16K)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>tls_nonce[]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>tls_data[]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TLS v1.3

- Layout of a TLS record
- More detailed information at: https://tls.ulfheim.net/

<table>
<thead>
<tr>
<th>TLS REC(s)</th>
<th>TCP HDR</th>
<th>IPv4/IPv6 HDR</th>
<th>ETH HDR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>uint8_t</td>
<td>tls_type (data=23)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>tls_vmajor (3)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>tls_vminor (3)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint16_t</td>
<td>tls_length (0..16K)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>uint8_t</td>
<td>tls_data[ ]</td>
<td></td>
</tr>
</tbody>
</table>
Advanced Encryption Standard, AES

- See: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

- A 16-byte block cipher

- The stream version can stop and resume encryption at any arbitrary point in the TLS record
  - Supports the concept of a crypto cursor

- FreeBSD also supports CBC
Current FreeBSD alternatives (OpenSSL based)
- Generic user-space, AES-NI
- SW kernel TLS, AES-NI
- Open Crypto Framework kernel backend
- TCP Offload Engine for TLS
- NIC kernel TLS
A look inside OpenSSL

- Datapath is oriented around:
  - `typedef struct bio_st BIO;`
  - `BIO_read()`
  - `BIO_write()`
- All data must have a pointer in user-space in order to be encrypted
- Based on the source and sink methodology
- Refer to the `bio(3)` manual page
OpenSSL and kTLS

- 16 patches have been submitted by Boris Pismenny <borisp@mellanox.com>
- FreeBSD userspace APIs:
  - `#include <sys/ktls.h>`
  - `setsockopt(TCP_TXTLS_ENABLE)`
  - `setsockopt(TCP_TXTLS_MODE)`
- FreeBSD kernel support added in r351522:
  - [https://svnweb.freebsd.org/changeset/base/351522](https://svnweb.freebsd.org/changeset/base/351522)
Netflix kTLS

• Kernel TLS Motivation
  ○ Handle 100Gb/s of TLS with nginx
  ○ Retain performance advantages of async sendfile(9) (fewer context switches, no nginx thread pool, no extra memory copy)
  ○ Eliminate any possible inefficiency
New mbuf technologies

- Not ready flag
- Unmapped mbufs
- Send Tags
● mbuf flag `M_NOTREADY` tell socket buffers if mbufs are ready for transmission or not.
● Added to support async sendfile in r275329
● Sendfile(9) adds mbuf to socket buffer marked `M_NOTREADY`
  ○ Until `M_NOTREADY` is cleared, tcp cannot send it
● disk reads are issued into those mbufs
● `M_NOTREADY` cleared and tcp_usr_ready() routine called after disk read is complete
● Allows a simple mbuf filter routine, like TLS encryption, to process the mbufs before they are submitted to the network driver via the TCP stack.
Netflix “unmapped” mbufs

- Called “unmapped” because they carry an array of pointers to unmapped physical addresses.
- Initially envisioned for sendfile, not TLS
- Dramatically reduces the length of socket buffer mbuf chains, thus reducing cache misses. For a 16K TLS record, it compresses chains by about 6:1 (TLS hdr, trailer and 4 buffers). For unencrypted sendfile, it can compress mbuf chains up to 19:1
  - 5-20% CPU reduction in Netflix unencrypted workloads
- Describes a TLS record entirely, including TLS header, trailer, message data, and pointers to kernel TLS session state in a single mbuf
- A single reference counted entity per TLS record is key for NIC TLS offload to be able to easily handle TCP retransmissions.
Software Kernel TLS Implementation, TLS 1.0 -> TLS 1.3

- Plaintext data passed to kernel via sendfile() or sosend().
- The kernel frames TLS records into M_NOMAP mbufs at sendfile() or sosend() time and places them into socket buffers.
- Mbuf chains are marked with M_NOTREADY
- Framed records are queued for encryption when they would previously be marked “ready”
- Encryption is done by a pool of kernel threads (1 per core)
- Once encrypted, mbufs are marked “ready” & sent to TCP
mbuf send tags

- A property of mbufs which tell the underlying network interface about dedicated packet processing and queues.
- A quick and efficient way to demultiplex data traffic.
- Allows for traversal through VLAN and LAGG (Link Aggregation).
- Safe against route changes.
mbuf send tag APIs

- Control path methods:
  - `struct mbuf_snd_tag *mst;`
  - `struct ifnet *ifp;`
  - `Allocate(ifp, &mst)`
  - `Modify(mst, arg)`
  - `Query(mst, arg)`
  - `Free(ifp, mst)`
mbuf send tags

- From Network Stack, NS, perspective:
  - struct mbuf *mb;
  - struct ifnet *ifp;
  - m_pkthdr.snd_tag = mst;
  - m_pkthdr.csum_flag |= CSUM_SND_TAG;
  - ifp->if_output(mb);
mbuf send tags

- From Network Driver, ND, perspective:
  - struct mbuf *mb;
  - struct xxx_send_tag *st;
  - st = container_of(m_pkthdr.snd_tag, ...)
  - select queue by st->queue;
Dataflow overview
Sendfile dataflow overview

Using sendfile and software kTLS, data is encrypted by the host CPU. This increases our bandwidth requirements by 25GB/s to roughly 55GB/s.
Using sendfile and inline kTLS, data is encrypted by the NIC.

This reduces our bandwidth requirements by 25GB/s to roughly the same as no TLS.
TLS before and after
NIC kTLS offload challenges

- Minor OSI model violation.
- Packets are sent containing full headers, except for un-encrypted payload.
- Prior to retransmission, crypto cursor needs update by re-transmitting off-the-wire parts of the TLS record, if any.
Netflix Video Serving with TLS

Kernel TLS Performance: 90Gb/s, 68% CPU (SW), 35% CPU (T6 NIC kTLS)

- Original (~2016) Netflix 100G NVME flash appliance
  - E5-2697A v4 @ 2.60GHz (16 core / 32 HTT), 128GB DDR4 2400MT/s, 1x100GbE, 4xNVME

**kTLS vs Userspace**

- Bandwidth
- %CPU

<table>
<thead>
<tr>
<th></th>
<th>Bandwidth</th>
<th>%CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO KTLS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SW KTLS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NIC KTLS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Mellanox NIC TLS

Iperf with TLS support

CPU utilization vs. # of parallel client threads

- Plain text
- SW kTLS
- NIC TLS
**Mellanox NIC TLS support**

- **ConnectX-6 DX (coming October 2019)**
  - [http://www.mellanox.com/page/ethernet_cards_overview](http://www.mellanox.com/page/ethernet_cards_overview)
  - 16 000 000 simultaneous TLS connections (25, 50, 100 and 200 Gbit/s)
Chelsio HW TLS support

- T6 NIC TLS supports TLS v1.1 and v1.2 using both AES-CBC and AES-GCM.
- TOE TLS support for kTLS is in progress.
- ccr(4) can be used for AES-GCM via the OCF backend.
Questions and Answers

Q/A