Securing Network Traffic Tunneled Over Kernel managed TCP/UDP sockets

Sowmini Varadhan(sowmini.varadhan@oracle.com)
Agenda

• What problem are we trying to solve?
  – Privacy, integrity protection, authentication of traffic that gets tunneled over kernel managed TCP and UDP sockets

• Options at socket layer: TLS, DTLS
  – Pros and cons

• Options at the IP layer: IPsec
  – Pros and cons

• Ongoing and future work
What problem are we trying to solve?

- Security for kernel-managed TCP and UDP sockets
  - VXLAN, GUE, Geneve, and other NVO3 solutions
  - RDS-TCP, KCM: Application traffic sent over PF_RDS/PF_KCM socket, which gets tunnelled over TCP in the kernel
- Security, with reasonable performance
  - Crypto has an unavoidable cost, but the rest of the perf should be streamlined
- Security, without regressing on Failover requirements for Cluster/HA
Typical model for kernel TCP/UDP sockets

- Application data from sender: can come from Virtual Machine, DB application, HTTP/2..
- Typically gets encapsulated in some protocol specific header (VXLAN, GUE, Geneve, RDS) that tracks control plane state (Tenant ID, VNI, OVS state, RDS port numbers..)
- Tunneled in the kernel over a UDP/TCP socket
- Receiver parses control plane header and delivers to the appropriate sender (tenant VM, DB application)
Privacy and Security Concerns

• Traffic tunneled over kernel sockets goes out in the clear today.

• As we scale multi-tenant Cloud environments, we have multiple tenants sharing the same physical infrastructure

• Attack vectors that need to be considered:
  – Protecting tenant payload and tunneling protocol header (privacy, integrity protection, authentication)
  – Protecting the control plane (TCP/IP, for RDS-TCP and KCM)
Privacy for tenant traffic

• Traffic that can traverse long internet paths: attackers should not be able to snoop/impersonate end-points
  - encrypt tenant data using encryption parameters that have been securely installed on both end-points after appropriate authentication.

• Typical solution to provide this is by using TLS/DTLS at the socket layer

• TLS has some attractive properties
  - Per-user authentication
  - Implemented at the application level, not the kernel. So easier support in multiple environments

• But there are some issues with using TLS/DTLS with kernel sockets
Challenges to using TLS/DTLS with RDS-TCP

- Cannot use DTLS/TLS directly on new socket types like PF_RDS and PF_KCM.
- No TLS in the kernel.
- TLS is a complex protocol- handshake and control-plane is complex.
- Can we move the TLS control-plane (including Handshake) to user-space, and just use the TLS negotiated keys for encryption in the kernel?
  - Attempted by Netflix for acceleration of encrypted sendfile()
  - Basis of recent kTLS RFC
Netflix/OCA

- Improve sendfile() throughput of encrypted data for the Netflix OpenConnect Appliance
- Traditional implementation: web-server gets client request for object on disk, retrieves object into a local buffer, encrypts/sends over TLS on network.
- Netflix optimization: when the client request comes in, issue sendfile() call on the file descriptor and socket descriptor: data would then never leave kernel address space.
Netflix/OCA: TLS based encryption in kernel

- Kernel needs to encrypt the data before sending it out on the socket to the network
- Netflix model: TLS session parameters are negotiated in user-space, and pushed down via socket options to the kernel
  - TLS session management in user-space
  - Encryption in kernel
- Primary goal is to provide faster encryption, not full support for a kernel TLS.
Netflix/OCA: TLS customizations

- Netflix/OCA ran into many questions when implementing this proposal
  - Encrypted messages like “Finished” can arrive before CCS has been processed, and keys are in place, so kernel data plane may end up having to buffer a lot of data
  - How will the kernel handle re-keying?
  - “..when you consider .. that messages in the TCP stream may arrive out of order, adding TLS for both sending and receiving adds a lot of complexity to the kernel” [Netflix/OCA]

- Netflix/OCA proposal only implements sender side of TLS, since it is primarily interested in accelerating sendfile()
Netflix/OCA results

- The Netflix/OCA finds that the performance improvements were not that significant for BSD
- Even if a Linux implementation could achieve better perf, the issues identified in the OCA experiment remain
  - Splitting the protocol into a control plane and a data plane is not what TLS intends, and such a split will result in new forms of asynchronicity.
- For securing kernel TCP/UDP sockets, we want a complete security solution.
HA/Failover in the split TLS model

- CCS, Re-keying etc: Control plane changes state, data-plane needs to be correctly synchronized with that change.
- HA/failover: data-plane can restart. Control plane needs to be in tandem with that. Examples:
  - Address/service migration for TCP connection, RDS-TCP module restart
  - RDS resets connection because it detects spurious headers or other compromise.
Protecting from TCP attacks

• TLS only secures the application data.
• TCP connection is still exposed and vulnerable to RST attacks, sequence number attacks.
• Attacks to TCP throw off the state machine in RDS-TCP reassembly.
  – Sender depends on TCP ack# to determine when it can take a dagram off the resend queue. Bogus sequence number reinjection is not acceptable.
• HA: when the RDS-TCP connection breaks, we try to re-connect today. If reconnecting, we should restart authentication, and preferably re-key.
Alternatives to TLS?

- TLS encrypts/authenticates at the socket layer
- Alternative to TLS: IPsec
- IPsec encrypts/authenticates at the IP layer
  - Fully integrated into the Linux kernel
  - Mature implementation; Interfaces between key management and kernel are well-understood.
What is IPsec?

- IP Security
- Suite of protocols for encryption (adding a “ESP” header) and Authentication (adding a “AUTH” header)
- ESP/AH are each applied to a “Security Association” (SA) that is pushed to kernel from user-space.
  - SA is defined by Admin.
  - Parameters: IP endpoint addresses, ports, IP protocol. Ports, protocol can be wild-cards
  - MAY be a TCP/UDP 4-tuple
- IKE (Internet Key Exchange) protocol for establishing keys (using pre-shared key, CA etc) from user-space
IPsec encryption with ESP

- Encrypts data (either TCP/UDP payload for transport mode, or IP packet for tunnel mode)
  - Confidentiality, data-origin authentication, integrity, anti-replay service.
- Adds an ESP header with an “Security Parameter Index” (SPI) and sequence number
  - SPI uniquely identifies a “Security Association” (SA) for which the security parameters (keys, crypto algo etc) are defined. Thus SPI essentially identifies a flow for IPsec
  - Sequence number is used to protect against replay attacks
- Adds an ESP trailer which contains the “original protocol” of the data that was encrypted.
IPsec encryption with ESP

• Encrypts data (either TCP/UDP payload for transport mode, or IP packet for tunnel mode)
  – Confidentiality, data-origin authentication, integrity, anti-replay service.

• Adds an ESP header with an “Security Parameter Index” (SPI) and sequence number
  – SPI uniquely identifies a “Security Association” (SA) for which the security parameters (keys, crypto algo etc) are defined. Thus SPI essentially identifies a flow for IPsec
  – Sequence number is used to protect against replay attacks

• Adds an ESP trailer which contains the “original protocol” of the data that was encrypted.
IPsec Transport vs Tunnel mode

- IPsec Transport mode: ESP/AH transforms apply to L4 (TCP or UDP) header and payload.
  - Protects TCP header
  - L3/routing information is not modified
  - Typically used for host-host IPsec

- IPsec tunnel mode: IP packet is encapsulated inside another IP packet. The IPsec transforms are applied to the inner (original) IP packet.
  - Protects IP and TCP header of the original packet
  - Typically used for VPNs
  - Routing information MAY be modified

- For Cloud/Cluster solutions IPsec Transport mode is sufficient as we do not wish to modify routing information.
Feasibility of using IPsec for kernel TCP/UDP sockets

- IPsec meets the security requirements for kernel TCP/UDP sockets. Linux supports a mature implementation, with all the needed features, and a variety of key distribution functions via the IKEv2 implementation.

- But what is the performance profile?

- We will now look at some performance instrumentation experiments, the findings, and ongoing work to evaluate IPsec impact on performance
IPsec performance evaluation environment

- Evaluate iPerf single-stream throughput and CPU utilization profile for a 10G line on an X5-4, using Intel's ixgbe driver
- Test permutations generated with the following parameters
  - With/Without TSO, GSO, GRO
  - Clear traffic vs IPsec with null encryption (thus no crypto overhead)
  - IPsec with 2 types of encryption:
    - AES-GCM-256 (Galois Counter Mode, keysize 256)
    - AES-CCM-128 (Counter with CBC MAC, keysize 128)
    - GCM: parallelizable in hardware. CCM: smaller gate-count but typically slower implementation.
IPsec test cases used for analysis

- Clear traffic, defaults for TSO, GRO, checksum offload
- Clear traffic, GSO on sender, GRO on receiver, no checksum offload on sender
- Clear traffic, GRO-only: no segmentation or checksum offload on sender, GRO on receiver
- Clear traffic, GSO-only: no TSO on sender, no GRO on receiver
- IPsec with null-encryption, default settings
- IPsec with AES-GCM-256, ICV len 16
- IPsec with AES-CCM-128, ICV len 8
IPsec impact on performance

![Bar chart showing IPsec impact on performance. The chart compares clear throughput (Gbps) and encrypted throughput (Gbps) with peak CPU utilization in percentage. The chart includes different configurations such as default, s/w-offload, gro-only, gso-only, esp-null, aes-gcm-c-256, and aes-ccm-a-128.](chart.png)
Observations:

• Loss of Segmentation/Receive offload (TSO, GSO, GRO) has a severe performance penalty even in the absence of IPsec
• IPsec transforms TCP/UDP payload.
  – MUST be done after segmentation
  – Stack implicitly disables TSO, GSO, GRO today when IPsec is engaged
• Manual Rx side iPerf placement and IRQ balancing was needed for IPsec cases. (loss of RSS/RFS/RPS for IPsec)
• Some inefficiencies in the way IPsec code manages memory
Retaining offload benefits for IPsec

- GSO/GRO are software offload implementations, and can be extended easily to apply IPsec transform after segmentation/receive offload
  - Work with Steffen Klassert for s/w offload
- IPsec transform after GSO segmentation
- Decrypt before GRO coalesce.
IPsec offload to GSO/GRO

- Steffen Klassert is working on patches to offload IPsec to GSO/GRO for Tunnel Mode
- Extended that patch-set to work for Transport Mode

<table>
<thead>
<tr>
<th></th>
<th>Throughput Gbps (peak CPU Utilization %)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ESP-NUL</td>
</tr>
<tr>
<td>Baseline</td>
<td>2.6 Gbps (71%)</td>
</tr>
<tr>
<td>GSO/GRO offload</td>
<td>8 Gbps (95%)</td>
</tr>
</tbody>
</table>
Reducing CPU utilization

- Hardware offload to NIC TSO? Many Intel 10G NICs (Niantic, Twinville, Sageville) already support IPsec offload but Linux stack needs enhancements
Receive side flow hashing

- When IPsec was enabled on the flow, had to manually do IRQ and process-CPU pinning to achieve the best performance
- On the receiver, this is achieved when the IRQs and iPerf process are pinned to separate CPUs.
- For clear traffic, this balancing would have been automatically achieved by RFS/RSS
  - RFS/RSS: Increase performance by steering packets to different queues based on filters applied to packet to determine flows
- RFS/RSS flow determined from TCP/UDP 4-tuple
- TCP/UDP header is encrypted by IPsec so port numbers are not available to RSS/RFS.
RSS/RFS/RPS for IPsec

- Can we use the SPI for flow hashing? Yes.
- SPI identifies the SA (Security Association), i.e., the “flow” for IPsec.
  - Already used by Tx path via `proto_ports_offset()`
- Drivers should be able to return a `rxhash` based on SPI, at least for ESP.
- Need to work the software RSS/RFS to do the same
Ongoing work

- Hardware offload of IPsec
  - Will reduce cpu util
  - NIC has to be updated with SA
  - Microsoft APIs give some clues about what is already available
- Better Rx flow hashing in h/w and s/w
- S/W tweaks to IPsec code paths to keep latency down
- Others? More benchmarks, IPsec offload deployment from within a VM..
Backup Slides
Case study: RDS-TCP Architectural Overview

user

RDS app

RDS app

RDS app

App data

Kernel TCP socket

rds_tcp

TCP

IP

driver

L2

IP

TCP

RDS

App data

TCP

RDS

App data

IP

TCP

RDS

App data

Synchronizing the control and data plane in the split TLS model

- Either client can send a “CCS” (ChangeCipherSpec) mid-stream, and the protocol mandates that both sides MUST start using the new parameters immediately after a CCS.
  - Encrypted data arrives before CCS has been processed
- Re-keying
- HA/failover: data-plane can restart. Control plane needs to be in tandem with that.
What does each transform look like for RDS? Clear vs TLS encrypted packet

Clear (unencrypted packet):

<table>
<thead>
<tr>
<th>Eth header</th>
<th>IP header; proto TCP 10.0.0.1 → 10.0.0.2</th>
<th>TCP hdr</th>
<th>TCP Payload</th>
</tr>
</thead>
</table>

TLS encrypted packet

<table>
<thead>
<tr>
<th>Eth header</th>
<th>IP header; proto TCP 10.0.0.1 → 10.0.0.2</th>
<th>TCP hdr</th>
<th>TCP payload</th>
</tr>
</thead>
</table>

Netdev 1.1 Seville, Spain
# Effect of IPsec transforms

## IPsec transport-mode encaps (ESP only)

<table>
<thead>
<tr>
<th>Eth header</th>
<th>IP header; proto ESP 10.0.0.1 → 10.0.0.2</th>
<th>ESP header SPI, seq#</th>
<th>TCP hdr &amp; payload</th>
<th>ESP trailer proto TCP</th>
</tr>
</thead>
</table>

## IPsec tunnel mode. The outer src/dst are determined by VPN config. They would be the 10.0.0.1 and 10.0.0.2 if no VPN gw is used.

<table>
<thead>
<tr>
<th>Eth hdr</th>
<th>Outer IP header; Proto ESP osrc → odst</th>
<th>ESP header SPI, seq#</th>
<th>Orig TCP/IP packet for 10.0.0.1 → 10.0.0.2, with TCP hdr and payload</th>
<th>ESP trailer Proto (4) IP-in-IP</th>
</tr>
</thead>
</table>
Acronyms: TSO, GSO, GRO

• Segmentation Offload: Split up the TCP packet into segments as late as possible at the sender during TCP transmission for better performance.
• Can be done in the NIC (TSO) or in software (GSO) just before handing off TCP/IP packet to NIC
  – http://www.linuxfoundation.org/collaborate/workgroups/networking/gso
• GRO: Generic Receive Offload: mirrors GSO on the receiver. “Identical” packets that match on constraints applied to the MAC/TCP/IP headers are merged and passed up the stack
  – https://lwn.net/Articles/358910/
Acronyms: RPS/RFS/RSS

• What is RPS/RFS/RSS
  – RPS: Receive Packet Steering, RFS: Receive Flow Steering
  – RSS: Receive Side scaling, hardware equivalent of RPS
  – See Documentation/networking/scaling.txt

• Increase performance by steering packets to different queues based on filters applied to packet to determine flows.

• Flow is typically a hash function applied to IP and/or TCP/UDP headers (port numbers)

• TCP/UDP header is encrypted by IPsec so port numbers are not available to RSS/RFS.