XFRM device - offloading the IPsec computations

Shannon Nelson <shannon.nelson@oracle.com>

Overview

IPsec is a useful feature for securing network traffic, but the computational cost is high: a 10Gbps link can easily be brought down to under 1Gbps, depending on the traffic and link configuration. Luckily, there are NICs that offer a hardware based IPsec offload which can radically increase throughput and decrease CPU utilization. The XFRM Device interface allows NIC drivers to offer to the stack access to the hardware offload.

Userland access to the offload is typically through a system such as libreswan or KAME/raccoon, but the iproute2 'ip xfrm' command set can be handy when experimenting. An example command might look something like this:

```
ip x s add proto esp dst 14.0.0.70 src 14.0.0.52 spi 0x07 mode transport \
   reqid 0x07 replay-window 32 \
   aead 'rfc4106(gcm(aes))' 0x44434241343332312423222114131211f4f3f2f1 128 \
   sel src 14.0.0.52/24 dst 14.0.0.70/24 proto tcp \
   offload dev eth4 dir in
```

Yes, that's ugly, but that's what shell scripts and/or libreswan are for.

Callbacks to implement

```
/* from include/linux/netdevice.h */
struct xfrmdev_ops {
      int     (*xdo_dev_state_add) (struct xfrm_state *x);
      void    (*xdo_dev_state_delete) (struct xfrm_state *x);
      void    (*xdo_dev_state_free) (struct xfrm_state *x);
      bool    (*xdo_dev_offload_ok) (struct sk_buff *skb,
                                     struct xfrm_state *x);
      void    (*xdo_dev_state_advance_esn) (struct xfrm_state *x);
};
```

The NIC driver offering ipsec offload will need to implement these callbacks to make the offload available to the network stack's XFRM subsystem. Additionally, the feature bits NETIF_F_HW_ESP and NETIF_F_HW_ESP_TX_CSUM will signal the availability of the offload.

Flow

At probe time and before the call to register_netdev(), the driver should set up local data structures and XFRM callbacks, and set the feature bits. The XFRM code's listener will finish the setup on NETDEV_REGISTER.

```
adapter->netdev->xfrmdev_ops = &ixgbe_xfrmdev_ops;
adapter->netdev->features |= NETIF_F_HW_ESP;
adapter->netdev->hw_enc_features |= NETIF_F_HW_ESP;
```

When new SAs are set up with a request for "offload" feature, the driver's xdo_dev_state_add() will be given the new SA to be offloaded and an indication of whether it is for Rx or Tx. The driver should

- verify the algorithm is supported for offloads
- store the SA information (key, salt, target-ip, protocol, etc)
- enable the HW offload of the SA
- return status value:

0	success
-EOPNETSUPP	offload not supported, try SW IPsec
other	fail the request

The driver can also set an offload_handle in the SA, an opaque void pointer that can be used to convey context into the fast-path offload requests:

```
xs->xso.offload_handle = context;
```

When the network stack is preparing an IPsec packet for an SA that has been setup for offload, it first calls into xdo_dev_offload_ok() with the skb and the intended offload state to ask the driver if the offload will serviceable. This can check the packet information to be sure the offload can be supported (e.g. IPv4 or IPv6, no IPv4 options, etc) and return true of false to signify its support.

When ready to send, the driver needs to inspect the Tx packet for the offload information, including the opaque context, and set up the packet send accordingly:

```
xs = xfrm_input_state(skb);
context = xs->xso.offload_handle;
set up HW for send
```

The stack has already inserted the appropriate IPsec headers in the packet data, the offload just needs to do the encryption and fix up the header values.

When a packet is received and the HW has indicated that it offloaded a decryption, the driver needs to add a reference to the decoded SA into the packet's skb. At this point the data should be decrypted but the IPsec headers are still in the packet data; they are removed later up the stack in xfrm_input().

find and hold the SA that was used to the Rx skb:

```
get spi, protocol, and destination IP from packet headers
xs = find xs from (spi, protocol, dest_IP)
xfrm_state_hold(xs);
```

store the state information into the skb:

```
sp = secpath_set(skb);
if (!sp) return;
sp->xvec[sp->len++] = xs;
sp->olen++;
```

indicate the success and/or error status of the offload:

```
xo = xfrm_offload(skb);
xo->flags = CRYPTO_DONE;
xo->status = crypto_status;
```

hand the packet to napi_gro_receive() as usual

In ESN mode, xdo_dev_state_advance_esn() is called from xfrm_replay_advance_esn(). Driver will check packet seq number and update HW ESN state machine if needed.

When the SA is removed by the user, the driver's xdo_dev_state_delete() is asked to disable the offload. Later, xdo_dev_state_free() is called from a garbage collection routine after all reference counts to the state have been removed and any remaining resources can be cleared for the offload state. How these are used by the driver will depend on specific hardware needs.

As a netdev is set to DOWN the XFRM stack's netdev listener will call xdo_dev_state_delete() and xdo_dev_state_free() on any remaining offloaded states.