NVIDIA DOCA East-West Overlay Encryption

Application

MLNX-15-060467 _v1.3 | May 2022

Table of Contents

Chapter 1. Introduction 1
Chapter 2. System Design 2
Chapter 3. Application Architecture 3
Chapter 4. DOCA Libraries 4
Chapter 5. Configuration Flow 5
5.1. swanctl.conf Files 6
Chapter 6. Running Application 9
6.1. Running strongSwan Example 9
6.2. Building strongSwan 10
6.3. Reverting IPsec Configuration 11
Chapter 7. References 12

Chapter 1. Introduction

IPsec is used to set up encrypted connections between different devices. It helps keep data sent over public networks secure. IPsec is often used to set up VPNs, and it works by encrypting IP packets as well as authenticating the packets' originator.

IPsec contains the following main modules:

- Key exchange - a key is a string of random bytes that can be used for encryption and decryption of messages. IPsec sets up keys with a key exchange between the connected devices, so that each device can decrypt the other device's messages.
- Authentication - IPsec provides authentication for each packet which ensures that they come from a trusted source.
- Encryption - IPsec encrypts the payloads within each packet and possibly, based on the transport mode, the packet's IP header.
- Decryption - at the other end of the communication, packets are decrypted by the IPsec supported node.

IPsec supports two types of headers:

- Authentication header (AH) - AH protocol ensures that packets are from a trusted source. AH does not provide any encryption.
- Encapsulating security protocol (ESP) - ESP encrypts the payload for each packet as well as the IP header depending on the transport mode. ESP adds its own header and a trailer to each data packet.

IPsec support two types of transport mode:

- IPsec tunnel mode - used between two network nodes, each acting as tunnel initiator/terminator on a public network. In this mode, the original IP header and payload are both encrypted. Since the IP header is encrypted, an IP tunnel is added for network forwarding. At each end of the tunnel, the routers decrypt the IP headers to route the packets to their destinations.
- Transport mode - the payload of each packet is encrypted, but the original IP header is not. Intermediary network nodes are therefore able to view the destination of each packet and route the packet, unless a separate tunneling protocol is used.

strongSwan is an open-source IPsec-based VPN solution. For more information, please refer to strongSwan documentation.

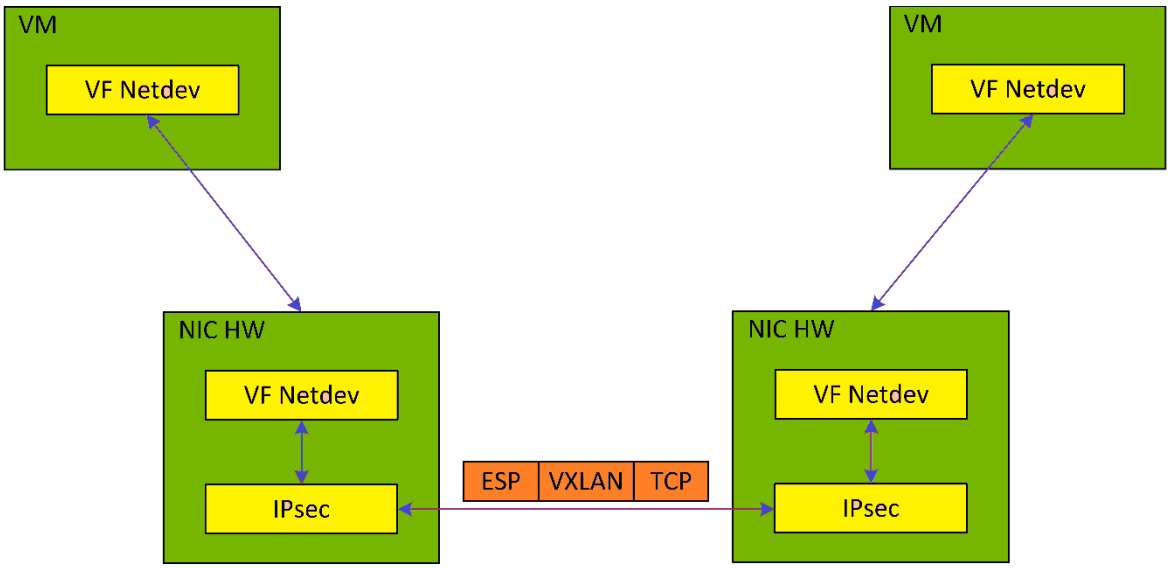
Chapter 2. System Design

IPsec full offload offloads both IPsec crypto (encrypt/decrypt) and IPsec encapsulation to the hardware. IPsec full offload is configured on the Arm via the uplink netdev.

The deployment model allows the IPsec offload to be transparent to the host with the benefits of securing legacy workloads (no dependency on host SW stack) and to zero CPU utilization on host.

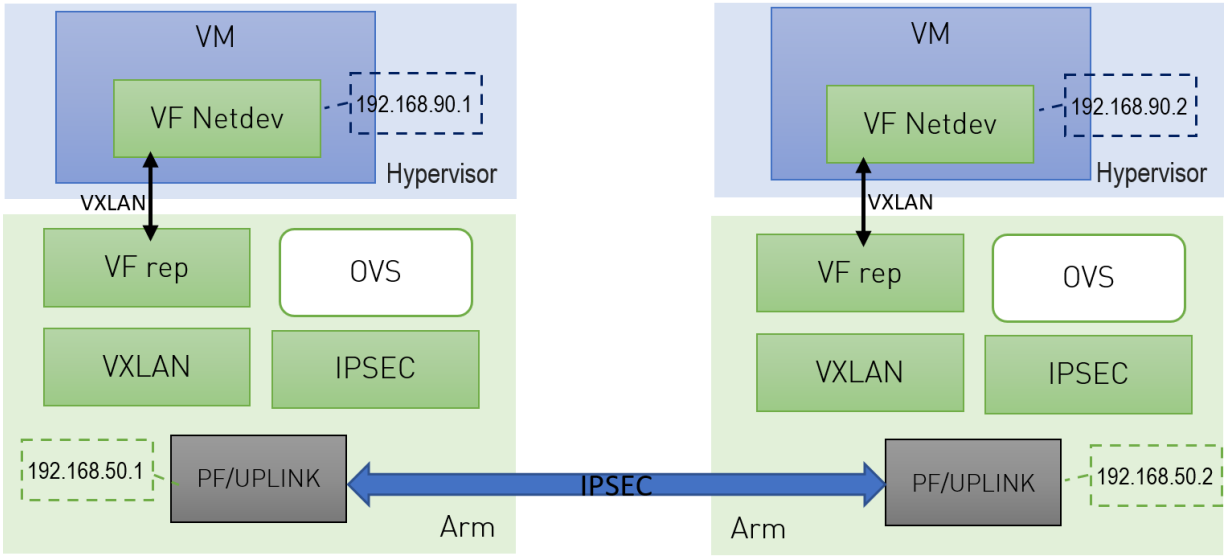
IPsec full offload configuration works with and is transparent to OVS offload. This means all packets from OVS offload are encrypted by IPsec rules.

The following figure illustrates the interaction between IPsec full offload and OVS VXLAN offload.



Note: OVS offload and IPsec IPv6 do not work together.

Chapter 3. Application Architecture



1. Configure strongSwan IPsec offload using swanctl.conf. configuration file.
2. Traffic is sent from the host through BlueField.
3. Using OVS, the packets are encapsulated on ingress using tunnel protocols (VXLAN for example) to match IPsec configuration by strongSwan.
4. Set by strongSwan configuration file, traffic will be encrypted using the hardware offload.
5. Egress flow is decryption first, decapsulation of the tunnel header and forward to the relevant physical function.

Chapter 4. DOCA Libraries

N/A

Chapter 5. Configuration Flow

The following section provides information on manually configuring IPsec full offload in general and on using IPsec with strongSwan specifically.

Note: There is a script, `east_west_overlay_encryption.sh`, which is elaborated on in section Running Application which performs the steps in this section automatically.

If you are working directly with the `ip xfrm` tool, you must use `/opt/mellanox/iproute2/sbin/ip` to benefit from IPsec full offload support.

1. Explicitly enable IPsec full offload on the Arm cores before full offload rule is configured.

   a). Disable `mlx-regex`. Run:
   ```
   systemctl stop mlx-regex
   ```
   b). Set `IPSEC_FULL_OFFLOAD="yes"` in `/etc/mellanox/mlnx-bf.conf`.

   Note: If `IPSEC_FULL_OFFLOAD` does not appear in `/etc/mellanox/mlnx-bf.conf` then you are probably using an old version of BlueField. Check the old way to enable IPsec full offload (in previous DOCA version).

   c). Restart IB driver (rebooting also works). Run:
   ```
   /etc/init.d/openibd restart
   ```
   d). Enable `mlx-regex`. Run:
   ```
   systemctl restart mlx-regex
   ```

   Note: To check if IPsec full offload is indeed enabled, Check that `/sys/class/net/*/compat/devlink/ipsec_mode` is `full`. If not (i.e., it it is `none`), then something is wrong. Retry this procedure from step 1.a. and try rebooting instead of restarting IB driver.

2. Enable TC offloading for the PF and the PF representor (e.g., PF is `p0` and PF representor is `pf0hpf`). Run:
   ```
   ethtool -K $PF0_REP hw-tc-offload on
   ethtool -K $PF0 hw-tc-offload on
   ```
3. Build a VXLAN tunnel over OVS, connect the PF representor to the same OVS bridge, and query OVS VXLAN hw_offload rules. In this case `OUTER_LOCAL_IP` is the IP address of the local machine's `pf0` interface, and `OUTER_REMOTE_IP` is the IP address of the other machine's `pf0` interface. Run:
   ```
   ovs-vsctl add-br vxlan-br
   ovs-vsctl add-port vxlan-br PF0_REP
   ```

```
ovs-vsctl add-port vxlan-br vxlan11 -- set interface vxlan11 type=vxlan
 options:local_ip=OUTER_LOCAL_IP options:remote_ip=OUTER_REMOTE_IP
 options:key=100 options:dst_port=4789
ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
```

If your operating system is Ubuntu, run:

```
service openvswitch-switch start
```

If your operating system is CentOS, run:

```
service openvswitch restart
```

Note: Make sure that the MTU of the PF used by VXLAN is at least 50 bytes larger than the VXLAN-REP MTU, and that the MTU of the Arm's PF used by IPsec is at least 26 bytes larger than the MTU of the Arm's VXLAN-REP.

4. Query OVS VXLAN `hw_offload` rules. Run:

```
ovs-appctl dpctl/dump-flows type=offloaded
```

5. Disable host PF as the port owner from Arm. Run:

```
mlxprivhost -d /dev/mst/mt${pciconf} --disable_port_owner r
```

Note: To get `${pciconf}`, run the following on the DPU:

```
ls --color=never /dev/mst/ | grep --color=never '^m.*f0$' | cut -c 3-
```

For example:

```
mlxprivhost -d /dev/mst/mt41686_pciconf0 --disable_port_owner r
```

6. Configure the `.swanctl.conf` files for each machine. See section swanctl.conf Files.

Note: Each machine should have exactly one `.swanctl.conf` file in `/etc/swanctl/swanctl.conf`.

7. Load the `swanctl.conf` files and initialize strongSwan. Run:

   a). On the receiver's machine, run:

   ```
   systemctl restart strongswan-starter.service
   swanctl --load-all
   ```

   b). On the initiator's machine, run:

   ```
   systemctl restart strongswan-starter.service
   swanctl --load-all
   swanctl -i --child bf
   ```

Now the IPsec connection should be established.

5.1. swanctl.conf Files

strongSwan configures IPSec HW full offload using a new value added to its configuration file `swanctl.conf`. The file should be placed under `sysconfdir` which by default can be found at `/etc/swanctl/swanctl.conf`.

The terms Left (BFL) and Right (BFR), in reference to the illustration under Application Architecture, are used to identify the two nodes (or machines) that communicate.

Note: Either side (BFL or BFR) can fulfill either role (initiator or receiver).

In this example, 192.168.50.1 is used for the left PF uplink and 192.168.50.2 for the right PF uplink.

```
connections {
   BFL-BFR {
      local_addrs  = 192.168.50.1
      remote_addrs = 192.168.50.2

      local {
        auth = psk
        id = host1
      }
      remote {
        auth = psk
        id = host2
      }

      children {
         bf {
            local_ts = 192.168.50.1/24 [udp/4789]
            remote_ts = 192.168.50.2/24 [udp/4789]
            esp_proposals = aes128gcm128-x25519
            mode = transport
            policies_fwd_out = yes
            hw_offload = full
         }
      }
      version = 2
      mobike = no
      reauth_time = 0
      proposals = aes128-sha256-x25519
   }
}

secrets {
   ike-BF {
      id-host1 = host1
      id-host2 = host2
      secret = 0sv+NkxY9LLZvwj4qCC2o/gGrWDF2d21jL
   }
   }
```

The BFB installation will place two example `swanctl.conf` files for BFL and BFR (`BFL.swanctl.conf` and `BFR.swanctl.conf` respectively) in the strongSwan `conf.d` directory. Each node should have only one `swanctl.conf` file in its strongSwan `conf.d` directory.

Note that:

- "`hw_offload = full`" is responsible for configuring IPSec HW full offload
- Full offload support has been added to the existing `hw_offload` field and preserves backward compatibility.

  For your reference:

Value	Description
no	Do not configure HW offload, fail if not supported
yes	Configure crypto HW offload if supported by the kernel, fail if not supported (Existing)

Value	Description
auto	Configure crypto HW offload if supported by the kernel, do not fail (Existing)
full	Configure full HW offload if supported by the kernel, fail if not supported (New)

- Whenever the value of `hw_offload` is changed, strongSwan configuration must be reloaded.
- Switching to crypto HW offload requires setting up `devlink/ipsec_mode` to `none` beforehand.
- Switching to full HW offload requires setting up `devlink/ipsec_mode` to `full` beforehand.
- `[udp/4789]` is crucial for instructing strongSwan to IPSec only VXLAN communication.
- Full HW offload can only be done on what is streamed over VXLAN.

Mind the following limitations:

Fields	Limitation
reauth_time	Ignored if set
rekey_time	Do not use. Ignored if set.
rekey_bytes	Do not use. Not supported and will fail if it is set.
rekey_packets	Use for rekeying

Chapter 6. Running Application

Refer to the following documents:

- NVIDIA DOCA Installation Guide for details on how to install BlueField-related software.
- NVIDIA DOCA Troubleshooting Guide for any issue you may encounter with the installation, compilation, or execution of DOCA applications.

6.1. Running strongSwan Example

Notes:

- IPsec daemons are started by systemd strongswan-starter.service
- Use `systemctl [start | stop | restart]` to control IPsec daemons through strongswan-starter.service. For example, to restart, run:
  ```
  systemctl restart strongswan-starter.service
  ```
  This command effectively does the same thing as `ipsec restart`.

> **Note:** Do not use the `ipsec` (located at `/usr/sbin/ipsec`) script to restart/stop/start.

This subsection explains how to configure and set an IPsec connection using the `ipsec` script. To configure the IPsec connection, you need two DPUs, referred to as the initiator and receiver machines. There are no differences between the two machines except that the initiator is the one that initiates the connection between the two.

The script is located under `/opt/mellanox/doca/applications/east_west_overlay_encryption/bin/east_west_overlay_encryption.sh`.

1. (Optional) Configure the JSON params file, located under `/opt/mellanox/doca/applications/east_west_overlay_encryption/bin/east_west_overlay_encryption_params.json`, for the script.

   > **Note:** You do not need to reconfigure the JSON file, just make sure that the `swanctl.conf` files are located in the correct path. If they cannot be found there, you must create them in each machine. An example of this file can be found in section swanctl.conf Files.

   The file includes the following parameters:

- `initiator_ip_addr` – the IP address of the initiator machine's port interface for the IPsec connection (set by default to 192.168.50.1)
- `receiver_ip_addr` – the IP address of the receiver machine's port interface for the IPsec connection (set by default to 192.168.50.2)
- `port_num` – the number of the port interface for the IPsec connection (0/1)
- `initiator_conf_file_path` – the initiator's `swanctl.conf` file path which must not be changed (default is `/etc/swanctl/conf.d/BFL.swanctl.conf`)
- `receiver_conf_file_path` – the receiver's `swanctl.conf` file path which should not be changed (default is `/etc/swanctl/conf.d/BFR.swanctl.conf`)

2. Run the script on the receiver DPU:
```
/opt/mellanox/doca/applications/east_west_overlay_encryption/bin/
east_west_overlay_encryption.sh -r
```
3. Run the script on the initiator DPU:
```
/opt/mellanox/doca/applications/east_west_overlay_encryption/bin/
east_west_overlay_encryption.sh -i
```
   You may now send encrypted data over the PF interface (192.168.50.[1|2]) configured for VXLAN.

For help and usage, run the script with `--help`/`-h` flag:
```
/opt/mellanox/doca/applications/east_west_overlay_encryption/bin/
east_west_overlay_encryption.sh -h
```

6.2. Building strongSwan

Note: Perform the following only if you want to build your own BFB and would like to rebuild strongSwan.

1. strongSwan IPsec full version can be found here (tag: 5.9.0bf).
2. Install dependencies mentioned here. libgmp-dev is missing from that list, so make sure to install that as well.
3. Git clone https://github.com/Mellanox/strongswan.git.
4. Git checkout BF-5.9.0.
5. Run `autogen.sh` within the strongSwan repo.
6. Run the following:
```
configure --enable-openssl --disable-random --prefix=/usr/local --sysconfdir=/etc
 --enable-systemd
make
make install
```
   Notes:
   - `--enable-systemd` enables the systemd service for strongSwan present inside the GitHub repo (see step 3) at `init/systemd-starter/strongswan-starter.service.in`. This service file is meant for Ubuntu, Debian and Yocto distributions. For CentOS, the contents of the above file must be replaced by the one present in `systemd-conf/strongswan-starter.service.in.centos` (inside the GitHub repo) before running the configure script above.

- When building strongSwan on your own, the `openssl.cnf.mlnx` file, required for PK and RNG HW offload via OpenSSL plugin, is not installed. It must be copied over manually from GitHub repo inside the `openssl-conf` directory. See section "Running Strongswan Example" for important notes.
- The `openssl.cnf.mlnx` file references PKA engine shared objects. libpka (version 1.3 or later) and openssl (version 1.1.1) must be installed for this to work.

6.3. Reverting IPsec Configuration

To destroy IPsec configuration, run the following on both machines:

```
/opt/mellanox/doca/applications/east_west_overlay_encryption/bin/
east_west_overlay_encryption.sh -d
```

Note: If you run this command without initializing the connection first (steps 2 and 3 in section Running strongSwan Example), you will receive errors. These errors have no functional impact and may be safely ignored.

Chapter 7. References

- `/opt/mellanox/doca/applications/east_west_overlay_encryption/bin/east_west_overlay_encryption.sh`
- `/opt/mellanox/doca/applications/east_west_overlay_encryption/bin/east_west_overlay_encryption_params.json`

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation nor any of its direct or indirect subsidiaries and affiliates (collectively: "NVIDIA") make no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assume no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA, the NVIDIA logo, and Mellanox are trademarks and/or registered trademarks of Mellanox Technologies Ltd. and/or NVIDIA Corporation in the U.S. and in other countries. The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a world¬wide basis. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com