NVIDIA DOCA App Shield Agent

Application Guide

MLNX-15-060536 _v1.4 | September 2022

Table of Contents

Chapter 1. Introduction

App Shield Agent monitors a process in the host system using the DOCA App Shield library (`doca-apsh`).

This security capability helps identify corruption of core processes in the system from an independent and trusted DPU. This is a major and innovate intrusion detection system (IDS) ability since it cannot be provided from inside the host.

The DOCA App Shield library gives the capability to read, analyze, and authenticate the host (bare metal/VM) memory directly from the DPU.

Using the library, this application hashes the un-writeable memory pages (also unloaded pages) of a specific process and its libraries. Then, at regularly occurring intervals the app authenticates the loaded pages.

The app reports pass/fail after every iteration until the first attestation failure. The reports are both printed to the console and exported to the DOCA telemetry service (DTS) using inter-process communication (IPC).

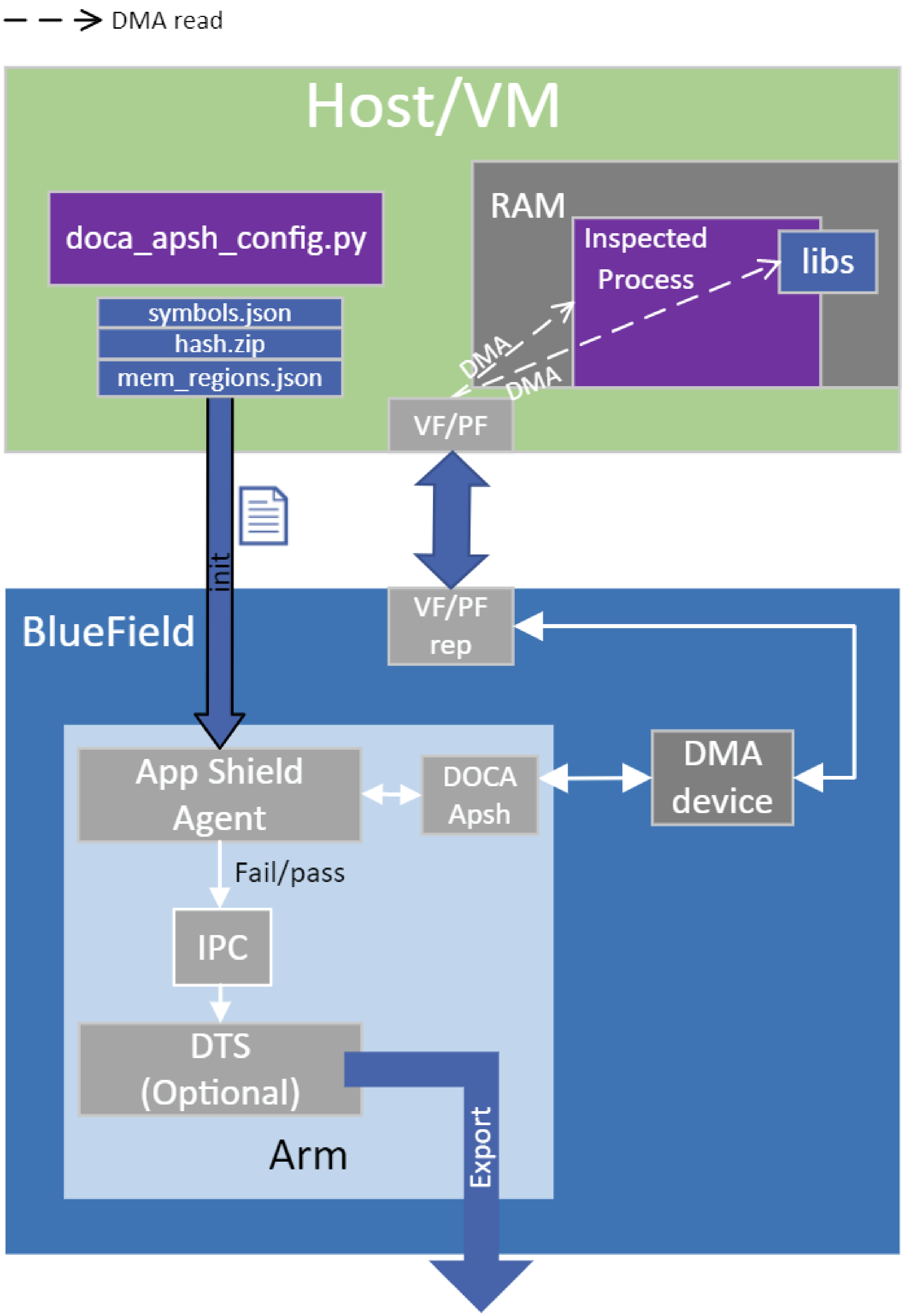
This document describes how to build secure process monitoring using the DOCA App Shield library, which leverages the DPU's advantages such as hardware-based DMA, integrity, and more.

Chapter 2. System Design

The App Shield agent is designed to run independently on the DPU's Arm without hindering the host.

The host's involvement is limited to configuring monitoring of a new process when there is a need to generate the needed ZIP and JSON files to pass to the DPU. This is done at inception ("time 0") which is when the host is still in a "safe" state.

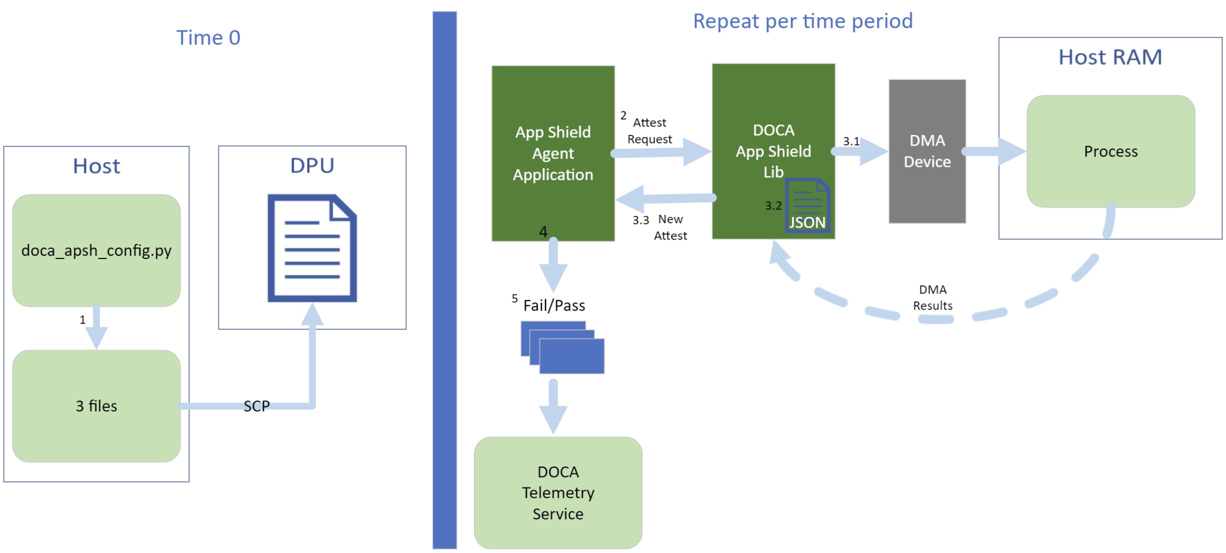
Generating the needed files can be done by running DOCA App Shield's `doca_apsh_config.py` tool on the host. See NVIDIA DOCA App Shield Programming Guide for more info.


DMA read
Host/VM
doca_apsh_config.py
symbols.json
hash.zip
mem_regions.json
RAM
Inspected Process
libs
DMA
DMA
VF/PF
init
BlueField
VF/PF rep
App Shield Agent
DOCA Apsh
DMA device
Fail/pass
IPC
DTS (Optional)
Arm
Export

Chapter 3. Application Architecture

The user creates three mandatory files using the DOCA tool `doca_apsh_config.py` and copies them to the DPU. The application can report attestation results to the:

- File
- Terminal
- DTS



1. The files are generated by running `doca_apsh_config.py` on the host against the process at time zero.

> **Note:** The actions 2-5 recur at regular time intervals.

2. The App Shield agent requests new attestation from DOCA App Shield library.
3. The DOCA App Shield library creates a new attestation:
   a). Scans and hashes process memory pages (that are currently in use).
   b). Compares the hash to the original hash.
   c). Creates attestation for each lib/exe involved in the process. Each of attestation includes the number of valid pages and the number of pages.

4. The App Shield agent searches each attestation for inconsistency between number of used pages and number of valid pages.
5. The App Shield agent reports results with a timestamp and scan count to:
    a). Local telemetry files – a folder and files representing the data a real DTS would have received. These files are used for the purposes of this example only as normally this data is not exported into user-readable files.
    b). DOCA log (without scan count).
    c). DTS IPC interface (even if no DTS is active).
6. The App Shield agent exits on first attestation failure.

Chapter 4. DOCA Libraries

This application leverages following DOCA libraries:

- DOCA App Shield library
- DOCA Telemetry library

Chapter 5. Configuration Flow

1. Parse application argument.

   ```
   doca_argp_init();
   ```

   a). Initialize arg parser resources.

   b). Register DOCA general flags.

   ```
   register_apsh_params();
   ```

   c). Register App Shield Agent application flags.

   ```
   doca_argp_start();
   ```

   d). Parse app flags.

2. Initialize DOCA App Shield lib context.

   ```
   doca_apsh_create();
   ```

   a). Create lib context.

   ```
   doca_devinfo_list_create();
   doca_dev_open();
   doca_devinfo_list_destroy();
   doca_apsh_dma_dev_set();
   ```

   b). Set DMA device for lib.

   ```
   doca_apsh_start();
   apsh_system_init();
   ```

   c). Start the context.

3. Initialize DOCA App Shield lib system context handler.

   ```
   doca_devinfo_remote_list_create();
   doca_dev_remote_open();
   doca_devinfo_remote_list_destroy();
   ```

   a). Get the representor of the remote PCIe function exposed to the system.

   ```
   doca_apsh_system_create();
   doca_apsh_sys_os_symbol_map_set();
   doca_apsh_sys_mem_region_set();
   doca_apsh_sys_dev_set();
   doca_apsh_sys_os_type_set();
   doca_apsh_system_start();
   ```

   b). Create and start the system context handler.

4. Find target process by pid.

   ```
   doca_apsh_processes_get();
   ```

5. Telemetry initialization.

   ```
   telemetry_start();
   ```

   a). Initialize a new telemetry schema.

   b). Register attestation type event.

c). Set up output to file (in addition to default IPC).
d). Start the telemetry schema.
e). Initialize and start a new DTS source with the `gethostname()` name as source ID.

6. Get initial attestation of the process.
```
doca_apsh_attestation_get();
```
7. Loop until attestation validation fail.
```
doca_apsh_attst_refresh();
/* validation logic */
doca_telemetry_source_report();
DOCA_LOG_INFO();
sleep();
```
8. DOCA App Shield Agent destroy.
```
doca_apsh_attestation_free();
doca_apsh_processes_free();
doca_apsh_system_destroy();
doca_apsh_destroy();
doca_dev_close();
doca_dev_remote_close();
```
9. Telemetry destroy.
```
telemetry_destroy();
```
10. Arg parser destroy.
```
doca_argp_destroy();
```

Chapter 6. Dependencies

The minimum required firmware version is 24.32.1010.

The application is only supported in the Ubuntu 20.04 BlueField OS.

Chapter 7. Running Application

1. 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.
2. The App Shield Agent binary is located under `/opt/mellanox/doca/applications/app_shield_agent/bin/doca_app_shield_agent`. To build the applications together, run:

```
cd /opt/mellanox/doca/applications/
meson build
ninja -C build
```

3. To build only the App Shield Agent:
    a). Edit the following flags in `/opt/mellanox/doca/applications/meson_option.txt`:
    - Set `enable_all_applications` to `false`
    - Set `enable_app_shield_agent` to `true`

    b). Run the commands in step 2.

> **Note:** `doca_app_shield_agent` is created under `./build/app_shield_agent/src/`.

Application usage:

```
Usage: doca_app_shield_agent [DOCA Flags] [Program Flags]
DOCA Flags:
 -h, --help                    Print a help synopsis
 -v, --version                 Print program version information
 -l, --log-level               Set the log level for the app <CRITICAL=0, DEBUG=4>
Program Flags:
  -p, --pid <arg>                Pid of the process to monitor
  -e, --ehm <path>               Path to the process executable 'hash.zip'
 generated file
  -m, --memr <path>              Path to the system memory regions map –
 'mem_regions.json' generated file.
  -f, --pcif <arg>               System PCI (VF/PF) VUID to use for DMA connection
  -d, --dma <arg>                DMA device name
  -o, --osym <path>              Path to the system os symbol map – 'symbols.json'
 generated file.
  -s, --osty [windows|linux]     OS of the system where the process is running
```

```
-t, --time <seconds>          Time interval between scans
```

Note: For additional information on the application, use the `-h` flag:

```
/opt/mellanox/doca/applications/app_shield_agent/bin/doca_app_shield_agent
 -h
```

4. The following steps need to be done only once.
   - Configure the BlueField's firmware.
     - On the BlueField system, configure the PF base address register and NVME emulation. Run:

       ```
       dpu> mlxconfig -d /dev/mst/mt41686_pciconf0 s PF_BAR2_SIZE=2
        PF_BAR2_ENABLE=1 NVME_EMULATION_ENABLE=1
       ```
     - Perform a cold boot from the host. Run:

       ```
       host> ipmitool power cycle
       ```

     Note: These configurations can be checked using the following command:

     ```
     dpu> mlxconfig -d /dev/mst/mt41686_pciconf0 q | grep -E "NVME|BAR"
     ```
   - Download target system (host/VM) symbols.
     - For Ubuntu:

       ```
       host> sudo tee /etc/apt/sources.list.d/ddebs.list << EOF
       deb http://ddebs.ubuntu.com/ $(lsb_release -cs) main restricted universe
        multiverse
       deb http://ddebs.ubuntu.com/ $(lsb_release -cs)-updates main restricted
        universe multiverse
       deb http://ddebs.ubuntu.com/ $(lsb_release -cs)-proposed main restricted
        universe multiverse
       EOF
       host> sudo apt install ubuntu-dbgsym-keyring
       host> sudo apt-get update
       host> sudo apt-get install linux-image-$(uname -r)-dbgsym
       ```
     - For CentOS:

       ```
       host> yum install --enablerepo=base-debuginfo kernel-devel-$(uname -r)
        kernel-debuginfo-$(uname -r) kernel-debuginfo-common-$(uname -m)-$(uname
        -r)
       ```
     - No action is needed for Windows
   - Perform IOMMU passthrough. This stage is only needed on some of the cases where IOMMU is not enabled by default (e.g., when the host is using an AMD CPU).

     Note: Skip this step if you are not sure whether you need it. Return to it only if DMA fails with a message in `dmesg` similar to the following:

     ```
     host> dmesg
     [ 3839.822897] mlx5_core 0000:81:00.0: AMD-Vi: Event logged
      [IO_PAGE_FAULT domain=0x0047 address=0x2a0aff8 flags=0x0000]
     ```
     - Locate your OS's `grub` file (most likely `/boot/grub/grub.conf`, `/boot/grub2/grub.cfg`, or `/etc/default/grub`) and open it for editing. Run:

       ```
       host> vim /etc/default/grub
       ```
     - Search for the line defining `GRUB_CMDLINE_LINUX_DEFAULT` and add the argument `iommu=pt`. For example:

       ```
       GRUB_CMDLINE_LINUX_DEFAULT="iommu=pt <intel/amd>_iommu=on"
       ```

- Run:
  - For Ubuntu:
    ```
    host> sudo update-grub
    host> ipmitool power cycle
    ```
  - For CentOS:
    ```
    host> grub2-mkconfig -o /boot/grub2/grub.cfg
    host> ipmitool power cycle
    ```
- For Windows targets only: Turn off Hyper-V capability.

5. Running the application on BlueField:
   - Pre-run setup:

     a). The DOCA App Shield library uses hugepages for DMA buffers. Therefore, the user is required to provide allocate specific size huge pages. Run:
     ```
     dpu> rm -rf "/mnt/huge/*"
     dpu> sudo echo 42 > /sys/devices/system/node/node0/hugepages/
     hugepages-32768kB/nr_hugepages
     dpu> \
     if [ ! -d "/mnt/huge" ] ; then
       mkdir "/mnt/huge"
     fi
     dpu> mount -t hugetlbfs -o pagesize=32MB none "/mnt/huge"
     ```
     b). Create the ZIP and JSON files. Run:

     > **Note:** If the kernel and process .exe have not changed, there no need to redo this step.

     ```
     target-system> cd /opt/mellanox/doca/tools/
     target-system> python3 doca_apsh_config.py <pid-of-process-to-monitor> --
     os <windows/linux> --path <path to dwarf2json executable  or pdbparse-to-
     json.py>
     target-system> cp /opt/mellanox/doca/tools/*.* <shared-folder-with-
     baremetal>
     dpu> scp <shared-folder-with-baremetal>/* <path-to-app-shield-binary>
     ```
     If the target system does not have DOCA installed, the script can be copied from the BlueField.

     The required `dwaf2json` and `pdbparse-to-json.py` are not provided with DOCA. Follow the NVIDIA DOCA App Shield Programming Guide for more information.
   - CLI example for running the app:
     ```
     dpu> /opt/mellanox/doca/applications/app_shield_agent/bin/
     doca_app_shield_agent -p 13577 -e hash.zip -m mem_regions.json -o symbols.json
      -f MT2125X03335MLNXS0D0F0VF1 -d mlx5_0 -t 3 -s linux
     ```

Chapter 8. Arg Parser DOCA Flags

Refer to NVIDIA DOCA Arg Parser User Guide for more information.

Flag Type	Short Flag	Long Flag/JSON Key	Description
General flags	l	log-level	Set the log level for the application: ▶ CRITICAL=0 ▶ ERROR=1 ▶ WARNING=2 ▶ INFO=3 ▶ DEBUG=4
	v	version	Print program version information
	h	help	Print a help synopsis
Program flags	p	pid	PID of the process to be attested
	e	ehm	Path to the pre-generated `hash.zip` file transferred from the host
	m	memr	Path to the pre-generated `mem_regions.json` file transfered from the host
	f	pcif	System PCIe function vendor unique identifier (VUID) of the VF/PF exposed to the target system. Used for DMA operations. To obtain this argument, run: `target-system> lspci -vv | grep "\[VU\] Vendor specific:"`

Flag Type	Short Flag	Long Flag/JSON Key	Description

			Example output:
			`[VU] Vendor specific: MT2125X03335MLNXS0D0F0`
			`[VU] Vendor specific: MT2125X03335MLNXS0D0F1`
			Two VUIDs are printed for each DPU connected to the target system. The first is of the DPU on `pf0` and the second is of the DPU on port `pf1`.
			Note: Running this command on the DPU outputs VUIDs with an additional "EC" string in the middle. You must remove the "EC" to arrive at the correct VUID.
			The VUID of a VF allocated on PF0/1 is the VUID of the PF with an additional suffix "VF<vf-number>", where vf-number is the VF index +1.
			For example, for the output in the example above:
			▶ PF0 VUID = MT2125X03335MLNXS0D0F0
			▶ PF1 VUID = MT2125X03335MLNXS0D0F1
			▶ VUID of VF0 on PF0 = MT2125X03335MLNXS0D0F0VF1
			VUIDs are persistent even on reset.

	d	dma	DMA device name to use

Flag Type	Short Flag	Long Flag/JSON Key	Description
	o	osym	Path to the pre-generated `symbols.json` file transferred from the host
	s	osty	OS type (`windows` or `linux`) of the system where the process is running
	t	time	Number of seconds to sleep between scans

Chapter 9. References

- /opt/mellanox/doca/applications/app_shield_agent/src/app_shield_agent.c

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