NVIDIA DOCA App Shield

Programming Guide

MLNX-15-060508 _v1.4 | November 2022

Table of Contents

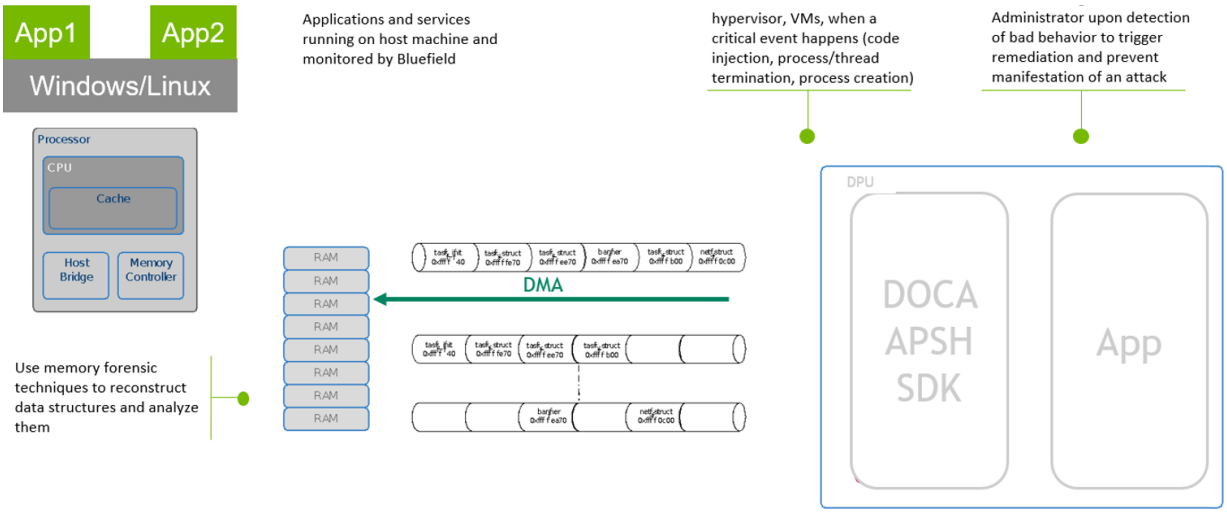
Chapter 1. Introduction

DOCA App Shield API offers a solution for strong intrusion detection capabilities using the DPU services to collect data from the host's memory. This solution provides intrusion detection and forensics investigation in a way that is:

- Robust against attacks on a host machine
- Able to detect a wide range of attacks (including zero-day attacks)
- Least disruptive to the execution of host application (where current detection solutions hinder the performance of host applications)

The App Shield detects attacks on critical services in a system. In many systems these services are responsible for assuring the integrity/privacy of the execution of other applications (e.g., a scrubbing service responsible for erasing the private data of users).

The following figure describes the relation between the DPU and the host memory where attacks may occur. The green squares are the assets that must resume operation unhindered. DOCA App Shield is responsible for acquiring information about processes to allow attack detection. To that end, DOCA App Shield exposes an API to the user allowing them to detect malicious activities (e.g., malicious processes, DLL files) by monitoring changes in critical memory parts directly from the Arm using DMA without involving the host OS or CPU.



Chapter 2. Prerequisites

To enable DOCA App Shield on the DPU, perform the following:

1. Enable NVMe emulation on the firmware.
2. Create huge pages (100 pages are recommended).
3. Disable `mlnx_snap` service.

Run a config command on the host/VM. Refer to doca_apsh_config for information on creating config files specific to the host/VM.

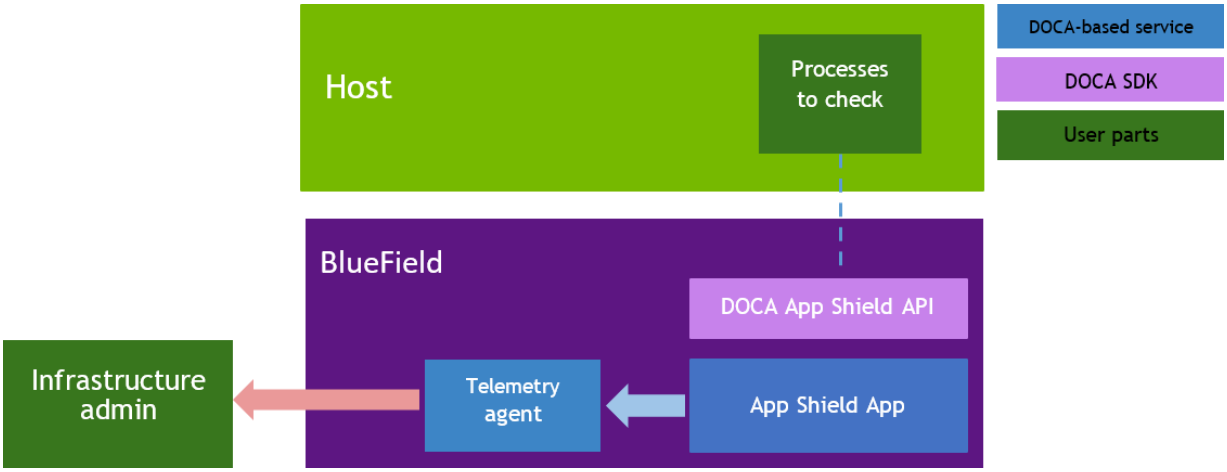
Run the following command to configure the DOCA:

```
# On the bluefield system, configure PF base address register and NVME emulation
dpu> mlxconfig -d /dev/mst/mt41686_pciconf0 s PF_BAR2_SIZE=2 PF_BAR2_ENABLE=1
 NVME_EMULATION_ENABLE=1
#Do Cold boot (from host)
host> ipmitool power cycle

## repeat after every reboot
# Allocate additional 2MB huge-pages for App Shield
dpu> nr_huge=$(cat /sys/devices/system/node/node0/hugepages/hugepages-2048kB/
nr_hugepages)
     nr_huge=$((100+$nr_huge))
     sudo echo $nr_huge > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/
nr_hugepages
# Disable the mlnx-snap service
dpu> systemctl stop mlnx_snap
```

Chapter 3. Architecture

The following block diagram illustrates the App Shield application flow.



- App Shield App – user application implementing the specific use case
- Telemetry Agent – collect telemetry metrics
- Processes to check – the host process to track

Chapter 4. Dependencies

The library requires firmware version 24.32.1010 or higher.

Chapter 5. API

For the library API reference, refer to the DOCA APSH API documentation in the NVIDIA DOCA Libraries API Reference Manual.

Note: The pkg-config (`*.pc` file) for the App Shield library is named `doca-apsh`.

The following sections provide additional details about the library API.

5.1. doca_apsh_dma_dev_set

To attach a DOCA DMA device to App Shield, calling this function is mandatory and must be done before calling `doca_apsh_start`.

```
doca_apsh_dma_dev_set(doca_apsh_ctx, doca_dev)
```

Where:

- `doca_apsh_ctx [in]` – App Shield opaque context struct
- `doca_dev [in]` – struct for DOCA device with DMA capabilities

5.2. doca_apsh_regex_dev_set

Calling this function is not mandatory to attach a RegEx DOCA device to App Shield. If the user wants to call the function, it must be done before calling `doca_apsh_start`.

```
doca_apsh_regex_dev_set(doca_apsh_ctx, doca_dev)
```

Where:

- `doca_apsh_ctx [in]` – App Shield opaque context struct
- `doca_dev [in]` – struct for DOCA device with RegEx capabilities

5.3. Capabilities Per System

For each initialized system, App Shield retrieves an array of the requested object according to the getter's name:

Function Name	Functions Information	Functions Signature	Return Type
Get modules	Returns an array with information about the system modules (drivers) loaded into the kernel of the OS.	`doca_error_t doca_apsh_modules_get(struct doca_apsh_system *system, struct doca_apsh_module ***modules, int *modules_size);`	▶ Array of `struct doca_apsh_module` ▶ `int`: Size of the returned array ▶ `doca_error` status
Get processes	Returns an array with information about each process running on the system.	`doca_error_t doca_apsh_processes_get(struct doca_apsh_system *system, struct doca_apsh_procces ***processes, int *processes_size);`	▶ Array of `struct doca_apsh_procces` ▶ `int`: Size of the returned array ▶ `doca_error` status
Get library	For a specified process, this function returns an array with information about each library loaded into this process.	`doca_error_t doca_apsh_libs_get(struct doca_apsh_process *process, struct doca_apsh_lib ***libs, int *libs_size);`	▶ Array of `struct doca_apsh_lib` ▶ `int`: Size of the returned array ▶ `doca_error` status
Get threads	For a specified process, this function returns an array with information about each thread running within this process.	`doca_error_t doca_apsh_threads_get(struct doca_apsh_process *process, struct doca_apsh_thread ***threads, int *threads_size);`	▶ Array of `struct doca_apsh_thread` ▶ `int`: Size of the returned array ▶ `doca_error` status
Get virtual memory areas/virtual address description	For a specified process, this function returns an array with information about each virtual memory area within this process.	`doca_error_t doca_apsh_vads_get(struct doca_apsh_process *process, struct doca_apsh_vad ***vads, int *vads_size);`	▶ Array of `struct doca_apsh_vma` ▶ `int`: Size of the returned array ▶ `doca_error` status
Get privileges	For a specified process, this function returns an array with information about each possible privilege for this	`doca_error_t doca_apsh_privileges_get(struct doca_apsh_process *process, struct doca_apsh_privilege ***privileges, int *privileges_size);`	▶ Array of `struct doca_apsh_privilege` ▶ `int`: Size of the returned array ▶ `doca_error` status

Function Name	Functions Information	Functions Signature	Return Type
	process, as described here. **Note:** Available on a Windows host only.		
Get environment variables	For a specified process, this function returns an array with information about each environment variable within this process. **Note:** Available on a Windows host only.	`doca_error_t doca_apsh_envars_get(struct doca_apsh_process *process, struct doca_apsh_envar ***envars, int *envars_size);`	▶ Array of `struct doca_apsh_envar` ▶ `int`: Size of the returned array ▶ `doca_error` status
Get handles	For a specified process, this function returns an array with information about each handle this process holds. **Note:** Available on a Windows host only.	`doca_error_t doca_apsh_handles_get(struct doca_apsh_process *process, struct doca_apsh_handle ***handles, int *handles_size);`	▶ Array of `struct doca_apsh_handle` ▶ `int`: Size of the returned array ▶ `doca_error` status
Get LDR modules	For a specified process, this function returns an array with information about each loaded module within this process. **Note:** Available on a Windows host only.	`doca_error_t doca_apsh_ldrmodules_get(struct doca_apsh_process *process, struct doca_apsh_ldrmodule ***ldrmodules, int *ldrmodules_size);`	▶ Array of `struct doca_apsh_ldrmodule` ▶ `int`: Size of the returned array ▶ `doca_error` status
Process attestation	For a specified process, this function attests the memory pages of the process according to a precomputed golden	`doca_error_t doca_apsh_attestation_get(struct doca_apsh_process *process, const char *exec_hash_map_path, struct doca_apsh_attestation ***attestation, int`	▶ Array of `struct doca_apsh_attestation` ▶ `int`: Size of the returned array ▶ `doca_error` status

Function Name	Functions Information	Functions Signature	Return Type
	hash file given as an input. **Note:** Single-threaded processes are supported at beta level.	`* attestation_size);`	
Attestation refresh	Refreshes a single attestation handler of a process with a new snapshot.	`doca_error_t doca_apsh_attst_refresh(struct doca_apsh_attestation ***attestation, int * attestation_size);`	▶ Array of `struct doca_apsh_attestation` ▶ `int`: Size of the returned array ▶ `doca_error` status

The following attribute getters return a specific attribute of an object, obtained from the array returned from the getter functions listed above, depending on the requested attribute:

```
doca_apsh_process_info_get(struct doca_apsh_proccess *process, enum
 doca_apsh_process_attr attr);
doca_apsh_module_info_get(struct doca_apsh_module *module, enum
 doca_apsh_module_attr attr);
doca_apsh_lib_info_get(struct doca_apsh_lib *lib, enum doca_apsh_lib_attr attr);
doca_apsh_thread_info_get(struct doca_apsh_thread *thread, enum doca_apsh_lib_attr
 attr);
doca_apsh_vad_info_get(struct doca_apsh_vad *vad, enum doca_apsh_vad_attr attr);
doca_apsh_privilege_info_get(struct doca_apsh_privilege *privilege, enum
 doca_apsh_privilege_attr attr);
doca_apsh_envar_info_get(struct doca_apsh_envar *envar, enum doca_apsh_envar_attr
 attr);
doca_apsh_handle_info_get(struct doca_apsh_handle *handle, enum
 doca_apsh_handle_attr attr);
doca_apsh_ldrmodule_info_get(struct doca_apsh_ldrmodule *ldrmodule, enum
 doca_apsh_ldrmodule_attr attr);
doca_apsh_attst_info_get(struct doca_apsh_attestation *attestation, enum
 doca_apsh_attestation_attr attr);
```

The return type of that attribute getters, based on the attribute, can be found in `doca_apsh_attr.h`.

Usage example:

```
const uint pid = doca_apsh_process_info_get(processes[i], DOCA_APSH_PROCESS_PID);
const char *proc_name = doca_apsh_process_info_get(processes[i],
 DOCA_APSH_PROCESS_COMM);
```

Chapter 6. App Shield Initialization and Teardown

There are different structures in App Shield that must be used for a BlueField client to be able to introspect into a system running on the host side, whether it is a bare-metal machine or a virtual machine.

6.1. Init App Shield

The App Shield context structure is used to init the devices on the DPU required to start monitoring App Shield systems.

To use `doca_apsh_ctx`, call:

```
struct doca_apsh_ctx *doca_apsh_create(void);
```

For `doca_app_shield_ctx` to work, an RDMA device must be set using the following function:

```
doca_error_t doca_apsh_dma_dev_set(struct doca_apsh_ctx *ctx, struct doca_dev
 *dma_dev);
```

For example:

```
doca_error_t ret = doca_apsh_dma_dev_set(ctx, dma_dev);
```

For `doca_app_shield_ctx` to use `doca_regex`, a RegEx device must be set using the following function:

```
doca_error_t doca_apsh_regex_dev_set(struct doca_apsh_ctx *ctx, struct doca_dev
 *regex_dev);
```

For example:

```
doca_error_t ret = doca_apsh_regex_dev_set(ctx, regex_dev);
```

After the above devices are set, the following function should be invoke:

```
doca_error_t doca_apsh_start(struct doca_apsh_ctx *ctx);
```

This establishes a connection to the devices.

When App Shield lib is no longer needed, a destruction must be called to release all the allocated resources:

```
void doca_apsh_destroy(struct doca_apsh_ctx *ctx);
```

6.2. Init System to Monitor

The system structure represents a system on the host that should be monitored. To instantiate an App Shield system, the following function must be called:

```
struct doca_apsh_system *doca_apsh_system_create(struct doca_apsh_ctx *ctx);
```

A single `doca_apsh_ctx` instance may be associated with many App Shield systems.

The App Shield system has the following attributes:

- Layer – specifies the system type. Types: Bare metal, virtual machine, or a container (for future use).
- System DOCA device – the reoresentor device obtained from the DPU. The device should be connected to the host/VM and functions as a representor VF/PF. To query/obtain the DOCA device, refer to the NVIDIA DOCA Libraries API Reference Manual.
- System/symbol map – includes information about the OS that App Shield needs to introspect (e.g., Window 10 Build 18363/Linux Ubuntu 20.04) and the size and fields of the OS structures such as process struct, which helps App Shield with the memory forensic techniques it uses to access and analyze these structures in the host's memory. This can be obtained by running the `doca_apsh_config.py` tool on the host.
- Memory regions – contains the allowed physical memory regions that App Shield can access. This information is needed since there are memory regions reserved by different PCIe devices. Some of these regions map device registers which change the device's state each time the regions (certain physical addresses in these regions) are read. These changes may confuse the device firmware and may, therefore, cause the system to crash/ freeze. This must be avoided. This can be obtained by running the `doca_apsh_config.py` tool on the host.
- KPGD file (optional and relevant only for Linux OS) – contains the KPGD physical address and the virtual address of `init_task`. This information is required since App Shield extracts data from the kernel struct in the physical memory. Thus, the kernel page directory table must translate the virtual addresses of these structs. This can be obtained by running the `doca_apsh_config.py` tool on the host with the flag `find_kpgd`.

Each one of these attributes (except for the KPGD file since it is optional) must be set by calling its respective function:

```
doca_error_t doca_apsh_sys_system_layer_set(struct doca_apsh_system *system, enum
 doca_apsh_system_layer layer_type);
doca_error_t doca_apsh_sys_dev_set(struct doca_apsh_system *system, struct
 doca_dev_remote *dev);
doca_error_t doca_apsh_sys_os_symbol_map_set(struct doca_apsh_system
 *system, const char *system_os_symbol_map_path);
doca_error_t doca_apsh_sys_mem_region_set(struct doca_apsh_system
 *system, const char *system_mem_region_path);
doca_error_t doca_apsh_sys_kpgd_file_set(struct doca_apsh_system *system, const char
 *system_kpgd_file_path);
```

For each system, after all the attributes are set, the following function must be called to start App Shield system monitoring:

```
doca_error_t doca_apsh_system_start(struct doca_apsh_system *system);
```

Other functions can be called to retrieve information from the system's memory after the App Shield system is started. These functions (also called capabilities) are expanded on in Capabilities Per System.

When the App Shield system is no longer needed, a destruction must be called to deallocate internal system memory:

```
void doca_apsh_system_destroy(struct doca_apsh_system *system);
```

6.3. doca_apsh_config

`doca_apsh_config` is used to get the config files necessary for running system analysis. Run the `doca_apsh_config` tool once system is up.

```
/opt/mellanox/doca/tools/doca_apsh_config.py --pid <pid> --os <os> --path
 <dwarf2json-path/pdbparse-to-json.py> --find_kpgd <0/1>
```

- Get the `dwarf2json` executable (relevant for Linux OS) which can be found here. Note that the executable must be compiled using Go Programming Language. For instructions, refer to the `dwarf2json` library.
- Get the `pdbparse-to-json.py` (relevant for Windows OS) which can be found here.

The tool creates the following files:

- Symbol map – this file changes once the system kernel is updated or the kernel module is installed. The file does not change on system reboot.
- Memory regions – this file changes when adding/removing hardware or drivers that affect the system's memory map (e.g., when adding register addresses). The file does not change on system reboot.
- `hash.zip` – this file is required for attestation API but is unnecessary for all other APIs. The zip file contains the required documentation to attest to a single process. The file changes on lib/executable update.
- `kpgd_file.conf` – this file is optional (relevant for Linux OS) and helps with faster initialization of the library. The file changes on system reboot.

Flags:

- `pid` – the process ID of the hashed process (only mandatory if the user wants to create `hash.zip`)
- `os` – `linux`/`windows`
- `path` –
  - Linux – path to the `dwarf2json` executable. Default `./dwarf2json`.
  - Windows – path to pdbparse-to-json.py. Default `./pdbparse-to-json.py`.
- `find_kpgd` – a flag to enable/disable creating `kpgd_file.conf` (relevant only to Linux OS). Default 0.

For example:

```
/opt/mellanox/doca/tools/doca_apsh_config.py --pid 100 --os linux --path ./
dwarf2json --find_kpgd 1
```

Chapter 7. Samples

Please refer to NVIDIA DOCA App Shield Sample Guide for more information about the API of this DOCA library.

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