NVIDIA DOCA App Shield

Programming Guide

MLNX-15-060508 _v1.2 | January 2022

Table of Contents

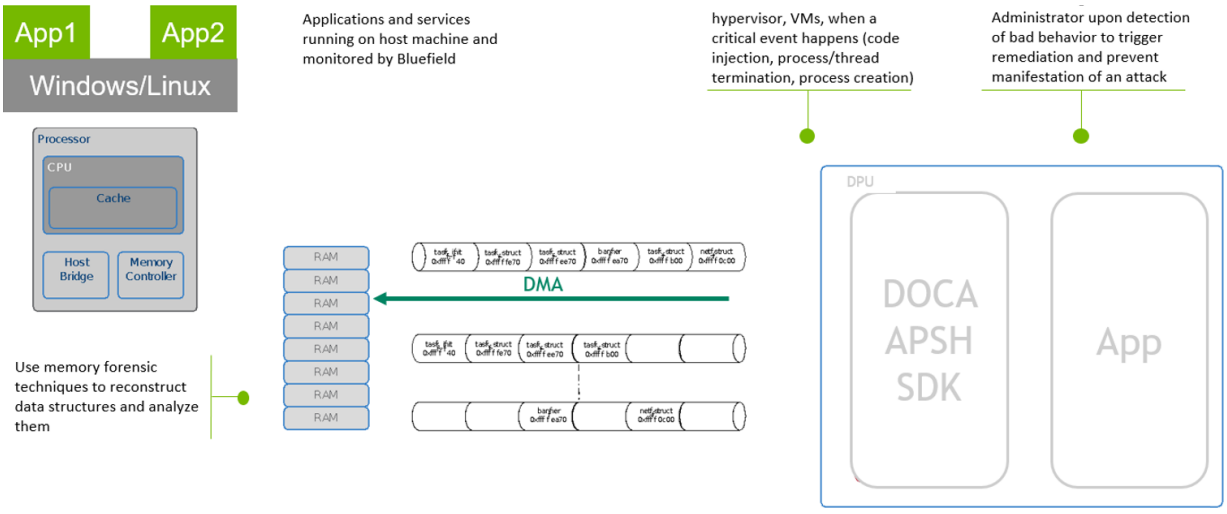
Chapter 1. Introduction

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 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)

Using App Shield, it is possible to detect attacks on critical services in a system. In many systems, those critical services are responsible for assuring the integrity/privacy of the execution of other applications. For example, a scrubbing service is responsible for erasing private data of users.

The following figure describes the relation between the DPU and the host memory where attacks may occur, and the green squares which 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.

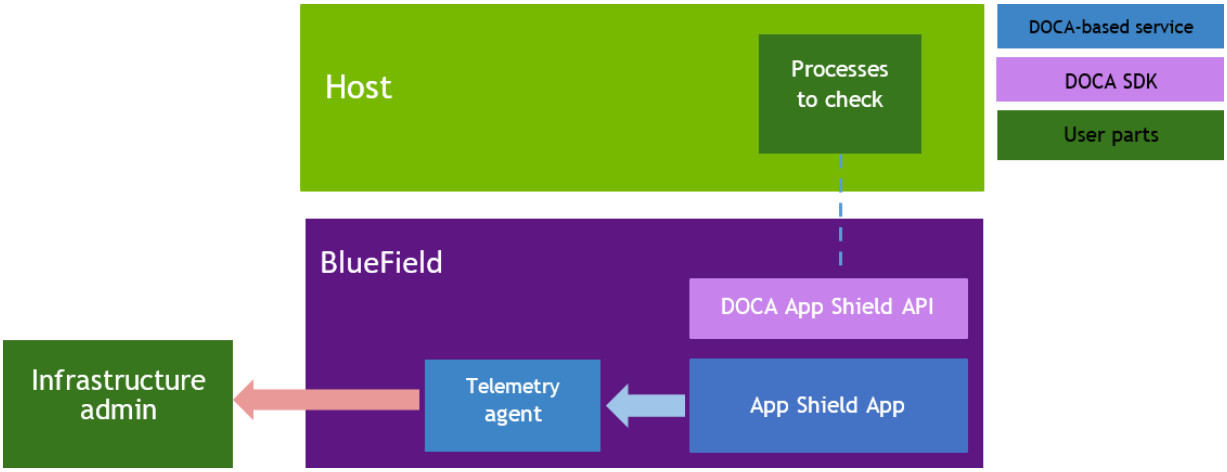


Chapter 2. Setup Configuration

The following code block describes how to configure DOCA App Shield on the DPU.

```
# On the bluefield system, configure PF base address register and NVME emulation

Arm> 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 huge-pages

Arm> rm -rf "/mnt/huge/*"

Arm> echo 42 > /sys/devices/system/node/node0/hugepages/hugepages-32768kB/
nr_hugepages

Arm> \
if [ ! -d "/mnt/huge" ] ; then
  mkdir "/mnt/huge"
fi

Arm> mount -t hugetlbfs -o pagesize=32MB none "/mnt/huge"


# Disable the mlnx-snap service

Arm> systemctl stop mlnx_snap
```

Chapter 3. App Shield Architecture



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

Chapter 4. App Shield Initialization and Teardown

In the App Shield API there are different structures which 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.

4.1. 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, a RegEx device and an RDMA device must be set, using these two functions:

```
int doca_apsh_dma_dev_set(struct doca_apsh_ctx *ctx, const char *dma_dev_name);
int doca_apsh_regex_dev_set(struct doca_apsh_ctx *ctx, const char *regex_dev_name);
```

For example:

```
int ret = doca_apsh_dma_dev_set(ctx, "mlx5_0");
```

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

```
int 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 deallocate any allocated memory:

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

4.2. Init System to Monitor

The system structure represents a system on the host that should be monitored. To instantiate an App Shield system, this 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 associate with many App Shield systems.

The App Shield system has the following attributes:

- layer – the type of the system. Types: Bare metal, virtual machine, or a container (for future use).
- PCI function – index of PCIe function connected to this system. Using DMA (direct memory access) read over this PCIe function representor on the DPU connected to a PCIe function on the host and is exposed to an OS that needs to be monitored. For example, for bare metal OS on the host, you can use the physical function (PF) that is usually index 0 (PF0). If you have a virtual function (VF) connected to a VM, to inspect that VM specify that VF's index.
- system/symbol map – includes information about the OS 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.
- memory regions – contains the allowed physical memory regions which 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 state of the device 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.

Each one of these attributes must be set by calling its suitable function:

```
int doca_apsh_sys_system_layer_set(struct doca_apsh_system *system, enum
 doca_apsh_system_layer layer_type);
int doca_apsh_sys_pcidev_set(struct doca_apsh_system *system, int pci_index);
int doca_apsh_sys_os_symbol_map_set(struct doca_apsh_system *system, const char
 *system_os_symbol_map_path);
int doca_apsh_sys_mem_region_set(struct doca_apsh_system *system, const char
 *system_mem_region_path);
```

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

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

Other functions can be called to retrieve information from the system's memory after 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);
```

Chapter 5. API

5.1. Capabilities Per System

For each initialized system, App Shield can retrieve the following information:

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	`int doca_apsh_modules_get(struct doca_apsh_system *system, struct doca_apsh_module ***modules);`	▶ Array of: `struct doca_apsh_module` ▶ int: Size of the returned array or a negative error code on error
Get processes	Returns an array with information about each of the processes running on the system	`int doca_apsh_processes_get(struct doca_apsh_system *system, struct doca_apsh_procces ***processes);`	▶ Array of: `struct doca_apsh_procces` ▶ int: Size of the returned array or a negative error code on error
Process refresh	Refreshes the information of a certain process	`int doca_apsh_proc_refresh(struct doca_apsh_process *process);`	0 on success, or a negative error code on error
Get library	For a specified process, this function returns an array with information about each of the libraries loaded into this process	`int doca_apsh_libs_get(struct doca_apsh_process *process, struct doca_apsh_lib ***libs);`	▶ Array of: `struct doca_apsh_lib` ▶ int: Size of the returned array or a negative error code on error
Get threads	For a specified process, this function returns an array with information about each of the threads running within this process	`int doca_apsh_threads_get(struct doca_apsh_process *process, struct doca_apsh_thread ***threads);`	▶ Array of: `struct doca_apsh_thread` ▶ int: Size of the returned array or a

Function Name	Functions Information	Functions Signature	Return Type
			negative error code on error
Get virtual memory areas	For a specified process, this function returns an array with information about each of the virtual memory areas within this process	`int doca_apsh_vmas_get(struct doca_apsh_process *process, struct doca_apsh_vad ***vmas);`	▶ Array of: `struct doca_apsh_vma` ▶ int: Size of the returned array or a negative error code on error
Process attestation	For a specified process, this function attests the memory pages of this process according to a precomputed golden hash file given as an input	`int doca_apsh_attestation_get(struct doca_apsh_process *process, const char *exec_hash_map_path, struct doca_apsh_attestation ***attestation);`	▶ Array of: `struct doca_apsh_attestation` ▶ int: Size of the returned array or a negative error code on error
Attestation refresh	Refresh a single attestation handler of a process with a new snapshot	`int doca_apsh_attst_refresh(struct doca_apsh_attestation ***attestation);`	▶ Array of: `struct doca_apsh_attestation` ▶ int: Size of the returned array or a negative error code on error

For each of the getter functions, a struct or an array of structs with the requested information is returned. To access this information, another getter function must be called specifying the exact information/attribute required from that struct.

```
const void *doca_apsh_proc_info_get(struct doca_apsh_procces *process, enum
 doca_apsh_process_attr attr);
const void *doca_apsh_module_info_get(struct doca_apsh_module *module, enum
 doca_apsh_module_attr attr);
const void *doca_apsh_lib_info_get(struct doca_apsh_lib *lib, enum
 doca_apsh_lib_attr attr);
const void *doca_apsh_thread_info_get(struct doca_apsh_thread *thread, enum
 doca_apsh_lib_attr attr);
const void *doca_apsh_vma_info_get(struct doca_apsh_vma *vma, enum
 doca_apsh_lib_attr attr);
const void *doca_apsh_attst_info_get(struct doca_apsh_attestation *attestation, enum
 doca_apsh_attestation_attr attr);
```

All the required attributes are defined in `/usr/include/doca_apsh_attr.h`.

5.2. Cleanup

Any of the structures retuned by the getter functions specified in Capabilities Per System must be freed after work is done with it. To destroy these structures, a destruction function must be called:

```
void doca_apsh_processes_free(struct doca_apsh_process **processes);
void doca_apsh_libs_free(struct doca_apsh_lib **libs);
void doca_apsh_threads_free(struct doca_apsh_thread **threads);
```

```
void doca_apsh_vads_free(struct doca_apsh_vad **vads);
void doca_apsh_attestation_free(struct doca_apsh_attestation **attestation);
```

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