NVIDIA DOCA App Shield

Programming Guide

MLNX-15-060508 _v2.0.2 | May 2023

Table of Contents

Chapter 1. Introduction

DOCA App Shield API offers a solution for strong intrusion detection capabilities using the DPU services to collect and analyze data from the host's (or a VM on the host) memory in real time. 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)
- Transparent to the host, such that the host does not need to install anything (other than providing some files obtained from the tool)

App Shield uses a DMA device to access the host's memory and analyze it. It also uses a RegEx device to scan the host's memory for regular expressions and signatures.

The App Shield API provides multiple functions that help with gathering data extracted from system's memory (e.g., processes list, modules list, connections). This data helps with detecting attacks on critical services or processes in a system (e.g., services that enforce integrity or privacy of the execution of different applications).

Chapter 2. Prerequisites

1. Make sure to run App Shield from the DPU as a root user.
2. If you are adding a RegEx device to App Shield, make sure to meet DOCA RegEx requirements which can be found in the NVIDIA DOCA RegEx Programming Guide.

Chapter 3. Dependencies

The library requires firmware version 24.32.1010 or higher.

Chapter 4. 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 APSH library is included in DOCA's regular definitions (i.e., `doca`).

The following sections provide additional details about the library API.

4.1. 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

4.2. doca_apsh_regex_dev_set

To attach a RegEx DOCA device to App Shield, calling this function is not mandatory (unless the user wants to use the netscan capability). 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

4.3. 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_process` ▶ `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	`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
	for this 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 * attestation_size);`	▶ Array of `struct doca_apsh_attestation` ▶ `int` – size of the returned array ▶ `doca_error` status

<table>
<tr><th>Function Name</th><th>Functions Information</th><th>Functions Signature</th><th>Return Type</th></tr>
<tr><td></td><td>hash file given as an input.

Note: Single-threaded processes are supported at beta level.</td><td></td><td></td></tr>
<tr><td>Attestation refresh</td><td>Refreshes a single attestation handler of a process with a new snapshot.</td><td><pre>doca_error_t doca_apsh_attst_refresh(struct doca_apsh_attestation ***attestation, int * attestation_size);</pre></td><td>▶ Array of struct doca_apsh_attestation
▶ int – size of the returned array
▶ doca_error status</td></tr>
<tr><td>Get NetScan</td><td>This function scans the system's physical memory and returns an array with information about each socket that resides in the memory.

Note:
Only available on hosts with one of the following Windows 10 OS builds:
<table><tr><th>Arch</th><th>Build No.</th></tr><tr><td>x86</td><td>10240</td></tr><tr><td></td><td>10586</td></tr><tr><td></td><td>14393</td></tr><tr><td></td><td>15063</td></tr><tr><td></td><td>17134</td></tr><tr><td></td><td>19041</td></tr><tr><td>x64</td><td>15063</td></tr><tr><td></td><td>16299</td></tr><tr><td></td><td>17134</td></tr><tr><td></td><td>17763</td></tr><tr><td></td><td>18362</td></tr></table></td><td><pre>doca_error_t doca_apsh_netscan_get(struct doca_apsh_system *system, struct doca_apsh_netscan ***connections, int *connections_size);</pre></td><td>▶ Array of struct doca_apsh_netscan
▶ int – size of the returned array
▶ doca_error status</td></tr>
</table>

Function Name	Functions Information	Functions Signature	Return Type
	Arch / Build No. 18363 19041 Note: This feature is currently supported at beta level.		
Get process parameters	For a specified process, this function returns a struct object (not an array) with information about the process' parameters (ones not included in the "get processes" capability). Note: Available on a Windows host only. Note: This feature is currently supported at beta level.	`doca_error_t doca_apsh_process_parameters_get(struct doca_apsh_process *process, struct doca_apsh_process_parameters **process_parameters);`	▶ An object of `struct doca_apsh_process_paramters` ▶ `doca_error` status
Get SIDs	For a specified process, this function returns an array with information about each SID (security identifier) included in the process's security context. Note: Available on a Windows host only.	`doca_error_t doca_apsh_sids_get(struct doca_apsh_process *process, struct doca_apsh_sid ***sids, int *sids_size);`	▶ An object of `struct doca_apsh_sid` ▶ `int` – size of the returned array ▶ `doca_error` status
Perform Yara scan	For a specified process, this function	`doca_error_t doca_apsh_yara_get(struct`	

Function Name	Functions Information	Functions Signature	Return Type
	returns an array with information about each Yara rule match found in the process' memory. Note: Available on a Windows host and Ubuntu 22.04 DPU.	doca_apsh_process *process, enum doca_apsh_yara_rule *yara_rules_arr, uint32_t yara_rules_arr_size, uint64_t scan_type, struct doca_apsh_yara ***yara_matches, int *yara_matches_size); Note: To get a better understanding of the arguments, refer to documentation in doca_apsh.h.	

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);
doca_apsh_netscan_info_get(struct doca_apsh_netscan *connection, enum
 doca_apsh_netscan_attr attr)
doca_apsh_process_parameters_info_get(struct doca_apsh_process_parameters
 *process_parameters, enum doca_apsh_process_parameters_attr attr);
doca_apsh_sid_info_get(struct doca_apsh_sid *sid, enum doca_apsh_sid_attr attr);
doca_error_t doca_apsh_yara_get(struct doca_apsh_process *process, enum
 doca_apsh_yara_rule *yara_rules_arr, uint32_t yara_rules_arr_size, uint64_t
 scan_type, struct doca_apsh_yara ***yara_matches, int *yara_matches_size);
```

The return type of the attribute getter 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 5. 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.

5.1. doca_apsh_ctx

`doca_apsh_ctx` is the basic struct used by App Shield which defines the DMA and RegEx devices used to perform the memory forensics techniques required to run App Shield.

Note: The same `doca_apsh_ctx` struct may be used to run multiple App Shield instances over different systems (e.g., two different VMs on the host).

1. To acquire an instance of the `doca_apsh_ctx` struct, use the following function:
```
struct doca_apsh_ctx *doca_apsh_create(void);
```
2. To configure the `doca_apsh_ctx` instance with DMA and RegEx (optional) devices to use:
```
doca_error_t doca_apsh_dma_dev_set(struct doca_apsh_ctx *ctx, struct doca_dev
 *dma_dev);
doca_error_t doca_apsh_regex_dev_set(struct doca_apsh_ctx *ctx, struct doca_dev
 *regex_dev);
```

   Note: Setting a RegEx device is only mandatory if the user wants to use the Netscan capability.

3. To start the `doca_apsh_ctx` instance, call the following function:
```
doca_error_t doca_apsh_start(struct doca_apsh_ctx *ctx);
```
4. To destroy the `doca_apsh_ctx` instance when it is no longer needed, call:
```
void doca_apsh_destroy(struct doca_apsh_ctx *ctx);
```

5.2. doca_apsh_system

The `doca_apsh_system` struct is built on the `doca_apsh_ctx` instance. This struct is created per system running App Shield. `doca_apsh_system` defines multiple attributes used by App Shield to perform memory analysis over the specific system successfully.

1. To acquire an instance of the `doca_apsh_system` struct, use the following function:
2. To configure different attributes for the system instance:

   - OS type – specifies the system's OS type.

     ```
     doca_error_t doca_apsh_sys_os_type_set(struct doca_apsh_system *ctx, enum
      doca_apsh_system_os os_type);
     ```

     Note: Currently supported types: Windows or Linux.

   - System representor – specifies the representor of the device connected to the system for App Shield to run on (which can be a representor of VF/PF). For information on querying the DOCA device, refer to the NVIDIA DOCA Core Programming Guide.

     After acquiring the DOCA device, use the following function to configure it into the system instance:

     ```
     doca_error_t doca_apsh_sys_dev_set(struct doca_apsh_system *system, struct
      doca_dev_rep *dev);
     ```

   - System symbols map – includes information about the OS that App Shield is attempting to run on (e.g., Window 10 Build 18363) and the size and fields of the OS structures, which helps App Shield with the memory forensic techniques it uses to access and analyze these structures in the system's memory. This can be obtained by running the `doca_apsh_config.py` on the system machine.

     After obtaining it, run:

     ```
     doca_error_t doca_apsh_sys_os_symbol_map_set(struct doca_apsh_system
      *system, const char *system_os_symbol_map_path);
     ```

   - Memory regions – includes the physical addresses of the memory regions which are mapped for system memory RAM. This is needed to prevent App Shield from accessing other memory regions, such as memory mapped I/O regions. This can be obtained by running the `doca_apsh_config.py` tool on the system machine.

     After obtaining it, run:

     ```
     doca_error_t doca_apsh_sys_mem_region_set(struct doca_apsh_system
      *system, const char *system_mem_region_path);
     ```

   - 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 system machine with the flag `find_kpgd=1`. Since setting this attribute is optional, App Shield can work without it, but providing it speeds up App Shield's initialization process.

After obtaining it, run:

```
doca_error_t doca_apsh_sys_kpgd_file_set(struct doca_apsh_system
 *system, const char *system_kpgd_file_path);
```

3. To start the `doca_apsh_system`:

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

4. To destroy the `doca_apsh_system` instance when it is no longer needed, call:

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

5.3. doca_apsh_config.py Tool

The `doca_apsh_config.py` tool is a python3 script which can be used to obtain all the attributes needed to run `doca_apsh_system` instance.

The following parameters are necessary to use the tool:

Parameter	Description
`pid` (optional)	The process ID of the process we want to run attestation capability on
`os` (mandatory)	The OS type of the machine (i.e., Linux or Windows)
`find_kpgd` (optional)	Relevant for Linux OS only, AS flag to enable/ disable creating `kpgd_file.conf`. Default 0.
`files` (mandatory)	A list of files for the tool to create. File options: `hash`, `symbols`, `memregions`, `kpgd_file` (only relevant for Linux). Note: Make sure that the value set is appropriate for your setup.
`path` (mandatory)	▶ Linux – path to the `dwarf2json` executable. Default `./dwarf2json`. This file can be obtained by compiling the following project using Go. ▶ Windows – path to `pdbparse-to-json.py`. Default `./pdbparse-to-json.py`. This file can be found here. Note: Make sure that the value set is appropriate for your setup.

The tool creates the following files:

- Symbol map – this file changes once the system kernel is updated or a kernel module is installed. The file does not change on system reboot.
- Memory regions – this file changes when adding or 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 but is unnecessary for all other capabilities. The ZIP file contains the required data to attest to a single process. The file changes on library or executable update.
- `kpgd_file.conf` (relevant for Linux OS only) – helps with faster initialization of the library. The file changes on system reboot.

Chapter 6. DOCA App Shield Samples

This section provides DOCA App Shield library sample implementations on top of BlueField DPU.

6.1. Sample Prerequisites

Follow the prerequisites in Prerequisites then copy the generated JSON files, `symbols.json` and `mem_regions.json`, to the `/tmp/` directory.

6.2. Running the Sample

1. Refer to the following documents:
   - NVIDIA DOCA Installation Guide for Linux 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. To build a given sample:

```
cd /opt/mellanox/doca/samples/doca_apsh/<sample_name>
meson build
ninja -C build
```

> Note: The binary `doca_<sample_name>` will be created under `./build/`.

3. Sample (e.g., `apsh_libs_get`) usage:

```
Usage: doca_apsh_libs_get [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 program
 <CRITICAL=20, ERROR=30, WARNING=40, INFO=50, DEBUG=60>

Program Flags:
  -p, --pid                         Process ID of process to be analyzed
  -f, --vuid                        VUID of the System device
  -d, --dma                         DMA device name
  -s, --osty <windows|linux>        System OS type
```

For additional information per sample, use the `-h` option:

```
./build/doca_<sample_name> -h
```

6.3. Samples

6.3.1. Apsh Libs Get

This sample illustrates how to properly initialize DOCA App Shield and use its API to get the list of loadable libraries of a specific process.

The sample logic includes:

1. Opening DOCA device with DMA ability.
2. Creating DOCA Apsh context.
3. Setting and starting the Apsh context.
4. Opening DOCA remote PCI device via given vendor unique identifier (VUID).
5. Creating DOCA Apsh system handler.
6. Setting fields and starting Apsh system handler.
7. Getting the list of system process using Apsh API and searching for a specific process with the given PID.
8. Geting the list of process-loadable libraries using `doca_apsh_libs_get` Apsh API call.
9. Querying the libraries for 3 selected fields using `doca_apsh_lib_info_get` Apsh API call.
10. Printing libraries' attributes to the terminal.
11. Cleaning up.

References:

- `/opt/mellanox/doca/samples/doca_apsh/apsh_libs_get/apsh_libs_get_sample.c`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_libs_get/apsh_libs_get_main.c`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_libs_get/meson.build`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_common.c`; `/opt/mellanox/doca/samples/doca_apsh/apsh_common.h`

6.3.2. Apsh Modules Get

This sample illustrates how to properly initialize DOCA App Shield and use its API to get the list of installed modules on a monitored system.

The sample logic includes:

1. Opening DOCA device with DMA ability.
2. Creating DOCA Apsh context.
3. Setting and starting the Apsh context.
4. Opening DOCA remote PCI device via given VUID.
5. Creating DOCA Apsh system handler.

6. Setting fields and start Apsh system handler.
7. Getting the the list of system-installed modules using `doca_apsh_modules_get` Apsh API call.
8. Querying the names of modules using `doca_apsh_module_info_get` Apsh API call.
9. Printing the attributes of up to 5 moduless attributes to the terminal.
10. Cleaning up.

References:

- `/opt/mellanox/doca/samples/doca_apsh/apsh_libs_get/apsh_libs_get_sample.c`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_libs_get/apsh_libs_get_main.c`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_libs_get/meson.build`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_common.c`; `/opt/mellanox/doca/samples/doca_apsh/apsh_common.h`

6.3.3. Apsh Pslist

This sample illustrates how to properly initialize DOCA App Shield and use its API to get the list of running processes on a monitored system.

The sample logic includes:

1. Opening DOCA device with DMA ability.
2. Creating DOCA Apsh context.
3. Setting and starting the Apsh context.
4. Opening DOCA remote PCI device via given VUID.
5. Creating DOCA Apsh system handler.
6. Setting fields and starting Apsh system handler.
7. Getting the list of processes running on the system using `doca_apsh_processes_get` Apsh API call.
8. Querying the processes for 4 chosen attributes using `doca_apsh_proc_info_get` Apsh API call.
9. Printing the attributes of up to 5 processes to the terminal.
10. Cleaning up.

References:

- `/opt/mellanox/doca/samples/doca_apsh/apsh_pslist/apsh_pslist_sample.c`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_pslist/apsh_pslist_main.c`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_pslist/meson.build`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_common.c`; `/opt/mellanox/doca/samples/doca_apsh/apsh_common.h`

6.3.4. Apsh Threads Get

This sample illustrates how to properly initialize DOCA App Shield and use its API to get the list of threads of a specific process.

The sample logic includes:

1. Opening DOCA device with DMA ability.
2. Creating DOCA Apsh context.
3. Setting and starting the Apsh context.
4. Opening DOCA remote PCI device via given VUID.
5. Creating DOCA Apsh system handler.
6. Setting fields and starting Apsh system handler.
7. Getting the list of system processes using Apsh API and searching for a specific process with the given PID.
8. Getting the list of process threads using `doca_apsh_threads_get` Apsh API call.
9. Querying the threads for up to 3 selected fields using `doca_apsh_thread_info_get` Apsh API call.
10. Printing thread attributes to the terminal.
11. Cleaning up.

References:

- `/opt/mellanox/doca/samples/doca_apsh/apsh_threads_get/apsh_threads_get_sample.c`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_threads_get/apsh_threads_get_main.c`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_threads_get/meson.build`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_common.c`; `/opt/mellanox/doca/samples/doca_apsh/apsh_common.h`

6.3.5. Apsh Vads Get

This sample illustrates how to properly initialize DOCA App Shield and use its API to get the list of virtual address descriptors (VADs) of a specific process.

The sample logic includes:

1. Opening DOCA device with DMA ability.
2. Creating DOCA Apsh context.
3. Setting and start the Apsh context.
4. Opening DOCA remote PCI device via given VUID.
5. Creating DOCA Apsh system handler.
6. Setting fields and starting Apsh system handler.
7. Getting the list of system processes using Apsh API and searching for a specific process with the given PID.

8. Getting the list of process VADs using `doca_apsh_vads_get` Apsh API call.
9. Querying the VADs for 3 selected fields using `doca_apsh_vad_info_get` Apsh API call.
10. Printing the attributes of up to 5 VADs to the terminal.
11. Cleaning up.

References:

- `/opt/mellanox/doca/samples/doca_apsh/apsh_vads_get/apsh_vads_get_sample.c`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_vads_get/apsh_vads_get_main.c`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_vads_get/meson.build`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_common.c`; `/opt/mellanox/doca/samples/doca_apsh/apsh_common.h`

6.3.6. Apsh Envars Get

This sample illustrates how to properly initialize DOCA App Shield and use its API to get the list of environment variables of a specific process.

Note: This sample works only on target systems with Windows OS.

The sample logic includes:

1. Opening DOCA device with DMA ability.
2. Creating DOCA Apsh context.
3. Setting and starting the Apsh context.
4. Opening DOCA remote PCIe device via given VUID.
5. Creating DOCA Apsh system handler.
6. Setting fields and starting Apsh system handler.
7. Getting the list of system processes using Apsh API and searching for a specific process with the given PID.
8. Getting the list of process envars using `doca_apsh_envars_get` Apsh API call.
9. Querying the envars for 2 selected fields using `doca_apsh_envar_info_get` Apsh API call.
10. Printing the envars attributes to the terminal.
11. Cleaning up.

References:

- `/opt/mellanox/doca/samples/doca_apsh/apsh_envars_get/apsh_envars_get_sample.c`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_envars_get/apsh_envars_get_main.c`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_envars_get/meson.build`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_common.c`; `/opt/mellanox/doca/samples/doca_apsh/apsh_common.h`

6.3.7. Apsh Privileges Get

This sample illustrates how to properly initialize DOCA App Shield and use its API to get the list of privileges of a specific process.

Note: This sample works only on target systems with Windows OS.

The sample logic includes:

1. Opening DOCA device with DMA ability.
2. Creating DOCA Apsh context.
3. Setting and starting the Apsh context.
4. Opening DOCA remote PCIe device via given VUID.
5. Creating DOCA Apsh system handler.
6. Setting fields and starting Apsh system handler.
7. Getting the list of system processes using Apsh API and searching for a specific process with the given PID.
8. Getting the list of process privileges using the `doca_apsh_privileges_get` Apsh API call.
9. Querying the privileges for 5 selected fields using the `doca_apsh_privilege_info_get` Apsh API call.
10. Printing the privileges attributes to the terminal.
11. Cleaning up.

References:

- `/opt/mellanox/doca/samples/doca_apsh/apsh_privileges_get/apsh_privileges_get_sample.c`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_privileges_get/apsh_privileges_get_main.c`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_privileges_get/meson.build`
- `/opt/mellanox/doca/samples/doca_apsh/apsh_common.c`; `/opt/mellanox/doca/samples/doca_apsh/apsh_common.h`

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