DOCA App Shield
Table of contents

Introduction

Prerequisites

Dependencies

API

doca_apsh_dma_dev_set

Capabilities Per System

App Shield Initialization and Teardown

doca_apsh_ctx

doca_apsh_system

doca_apsh_config.py Tool

DOCA App Shield Samples

Sample Prerequisites

Running the Sample

Samples

Apsh Libs Get

Apsh Modules Get

Apsh Pslist

Apsh Threads Get

Apsh Vads Get

Apsh Envars Get

Apsh Privileges Get
This guide provides instructions on using the DOCA App Shield API.

**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 `doca_apsh_config.py` tool)

App Shield uses a DMA device to access the host's memory and analyze it.

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).

**Prerequisites**

1. Configure the NVIDIA® BlueField® networking platform's (DPU or SuperNIC) firmware.

   1. On BlueField, 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
   ```

   If working with VFs, configure NVME emulation, SR-IOV, and number of VFs. Run:
2. Perform graceful shutdown and a cold boot from the host.

**Info**

These configurations can be checked using the following command:

```
dpu> mlxconfig -d /dev/mst/mt41686_pciconf0 q | grep -E "NVME|BAR|SRIOV|NUM_OF_VFS"
```

2. 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-
  ```
No action is needed for Windows

3. Perform IOMMU passthrough. This stage is only necessary if IOMMU is not enabled by default (e.g., when the host is using an AMD CPU).

1. 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
   ```

2. 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"
   ```

3. Run:

   ```
   ```
Prior to performing a power cycle, make sure to do a **graceful shutdown**.

- **For Ubuntu:**
  ```
  host> sudo update-grub
  ```

- **For CentOS:**
  ```
  host> grub2-mkconfig -o /boot/grub2/grub.cfg
  ```

4. Prepare target:

1. Install DOCA on the target system.

2. Create the ZIP and JSON files. Run:

   ```
   target-system> cd /opt/mellanox/doca/tools/
   target-system> python3 doca_apsh_config.py --pid <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 BlueField.

   The required **dwarf2json** and **pdbparse-to-json.py** are not provided with DOCA.

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

**Dependencies**

The library requires firmware version 24.32.1010 or higher.

**API**

For the library API reference, refer to the DOCA APSH API documentation in the NVIDIA DOCA Library APIs.

ℹ️ **Note**

The pkg-config (*.pc file) for the APSH library is `doca-apsh`.

The following subsections provide more details about the library API.

### `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`.

```c
#define doca_apsh_dma_dev_set(doca_apsh_ctx, doca_dev)
```

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

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

<table>
<thead>
<tr>
<th>Getter Function Name</th>
<th>Functions Information</th>
<th>Functions Signature</th>
<th>Return Type</th>
</tr>
</thead>
</table>
| 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_process ***processes,
    int *processes_size);
``` | • Array of struct doca_apsh_process  
• int – size |
<table>
<thead>
<tr>
<th>Getter Function Name</th>
<th>Functions Information</th>
<th>Functions Signature</th>
<th>Return Type</th>
</tr>
</thead>
</table>
| 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 the returned array
• doca_error_stat
• Array of struct
doca_apsh_lib
• int – size of the returned array
• doca_error_stat
<p>|</p>
<table>
<thead>
<tr>
<th>Getter Function Name</th>
<th>Functions Information</th>
<th>Functions Signature</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get threads</td>
<td>For a specified process, this function returns an array with information about each thread running within this process.</td>
<td><code>doca_error_t doca_apsh_threads_get(struct doca_apsh_process *process, struct doca_apsh_thread ***threads, int *threads_size);</code></td>
<td>• Array of struct doca_apsh_thread &lt;br&gt; • int – size of the returned array &lt;br&gt; • doca_error_t status</td>
</tr>
<tr>
<td>Get virtual memory areas/virtual address descr</td>
<td>For a specified process, this function returns an array with information about each virtual memory area within this process.</td>
<td><code>doca_error_t doca_apsh_vads_get(struct doca_apsh_process *process, struct doca_apsh_vad ***vads, int *vads_size);</code></td>
<td>• Array of struct doca_apsh_vma &lt;br&gt; • int – size of the returned array</td>
</tr>
<tr>
<td>Getter Function Name</td>
<td>Functions Information</td>
<td>Functions Signature</td>
<td>Return Type</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------</td>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Get privileges</td>
<td>For a specified process, this function returns an array with information about each possible privilege for this process, as described <a href="#">here</a>.</td>
<td><code>doca_error_t doca_apsh_privileges_get(struct doca_apsh_process *process, struct doca_apsh_privilege ***privileges, int *privileges_size);</code></td>
<td><code>retrieved array</code></td>
</tr>
</tbody>
</table>

**Note**

Available on a Windows host only.
<table>
<thead>
<tr>
<th>Getter Function Name</th>
<th>Functions Information</th>
<th>Functions Signature</th>
<th>Return Type</th>
</tr>
</thead>
</table>
| Get environment variables | For a specified process, this function returns an array with information about each environment variable within this process. | `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_t` status |

**Note**
Available on a Windows host only.
<table>
<thead>
<tr>
<th>Getter Function Name</th>
<th>Functions Information</th>
<th>Functions Signature</th>
<th>Return Type</th>
</tr>
</thead>
</table>
| Get handles          | For a specified process, this function returns an array with information about each handle this process holds. | 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 |

ℹ️ **Note**
Available on a Windows host only.
<table>
<thead>
<tr>
<th>Getter Function Name</th>
<th>Functions Information</th>
<th>Functions Signature</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get LDR modules</td>
<td>For a specified process, this function returns an array with information about each loaded module within this process.</td>
<td>doca_error_t doca_apsh_ldrmodules_get(struct doca_apsh_process *process, struct doca_apsh_ldrmodule ***ldrmodules, int *ldrmodules_size);</td>
<td>• Array of struct doca_apsh_ldrmodule</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• int – size of the returned array</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• doca_error_status</td>
</tr>
<tr>
<td>Note</td>
<td>Available on a Windows host only.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process attestation</td>
<td>For a specified process, this function attests the memory pages of the process according to a precomputed golden hash file given as an input.</td>
<td>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);</td>
<td>• Array of struct doca_apsh_attestation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• int – size of the</td>
</tr>
<tr>
<td>Note</td>
<td>Single-threaded processes are</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Getter Function Name</td>
<td>Functions Information</td>
<td>Functions Signature</td>
<td>Return Type</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------</td>
<td>---------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 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);` | returned array  
  - `doca_error_t`  
  - `doca_apsh_attst_refresh(struct doca_apsh_attestation ***attestation, int *attestation_size);` |
<table>
<thead>
<tr>
<th>Getter Function Name</th>
<th>Functions Information</th>
<th>Functions Signature</th>
<th>Return Type</th>
</tr>
</thead>
</table>
| Get NetScan          | This function scans the system's physical memory and returns an array with information about each socket that resides in the memory. | `doca_error_t doca_apsh_netscan_get(struct doca_apsh_system *system, struct doca_apsh_netscan ***connections, int *connections_size);` | • Array of struct doca_apsh_netscan  
• int – size of the returned array  
• doca_error_status |

**Note**

Only available on hosts with one of the following Windows 10 OS builds:

<table>
<thead>
<tr>
<th>Arch</th>
<th>Build No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86</td>
<td></td>
</tr>
<tr>
<td></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></td>
</tr>
<tr>
<td></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>
<tr>
<td></td>
<td>18363</td>
</tr>
<tr>
<td></td>
<td>19041</td>
</tr>
<tr>
<td>Getter Function Name</td>
<td>Functions Information</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------</td>
</tr>
</tbody>
</table>

**Note**

This feature is currently supported at beta level.
<table>
<thead>
<tr>
<th>Getter Function Name</th>
<th>Functions Information</th>
<th>Functions Signature</th>
<th>Return Type</th>
</tr>
</thead>
</table>
| 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). | doca_error_t
doca_apsh_process_parameters_get(struct doca_apsh_process *process, struct doca_apsh_process_parameters **process_parameters); |          |
| Note                 | Available on a Windows host only.                                                                                                                                                                                      |                                                                         | An object of struct doca_apsh_process_parameters |
| Note                 | This feature is currently supported at beta level.                                                                                                                                                                   |                                                                         | doca_error_status |
| Get security identifier (SID) | For a specified process, this function returns an array with information about each SID (security identifier) included in the process's security context.                                                                 | doca_error_t
doca_apsh_sids_get(struct doca_apsh_process *process, struct doca_apsh_sid ***sids, int *sids_size); | Array of struct doca_apsh_sid |
<p>| Note                 | Available on a Windows host                                                                                                                                                                                          |                                                                         | int – size of the return          |</p>
<table>
<thead>
<tr>
<th>Getter Function Name</th>
<th>Functions Information</th>
<th>Functions Signature</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfor Yara scan</td>
<td>For a specified process, this function returns an array with information about each Yara rule match found in the process's memory.</td>
<td><code>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);</code></td>
<td><code>• Array of struct doca_apsh_yara&lt;br&gt;• int – size of the returned array&lt;br&gt;• doca_error_status</code></td>
</tr>
</tbody>
</table>

**Note**
Available on a Windows host and Ubuntu 22.04 DPU.

**i**  To get a better understanding of the arguments, refer to document
<table>
<thead>
<tr>
<th>Getter Function Name</th>
<th>Functions Information</th>
<th>Functions Signature</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get containers</td>
<td>Returns an array with information about each container running on the system.</td>
<td><code>doca_error_t</code> <code>doca_apsh_containers_get(struct doca_apsh_system *system, struct doca_apsh_container ***containers, int *containers_size);</code></td>
<td>• Array of struct doca_apsh_container • int – size of the returned array • doca_error_status</td>
</tr>
<tr>
<td></td>
<td>Note</td>
<td>/doca_apsh.h.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Note</td>
<td>Available on a Linux host only.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Note</td>
<td>Only available for containers on the following runtimes:  runc containerd</td>
<td></td>
</tr>
<tr>
<td>Get container's processes</td>
<td>For a specified container, this function returns an array with information about each process running within this container.</td>
<td><code>doca_error_t</code> <code>doca_apsh_container_processes_get(struct doca_apsh_container *container, struct</code></td>
<td>• Array of struct doca_apsh_container</td>
</tr>
<tr>
<td>Getter Function Name</td>
<td>Functions Information</td>
<td>Functions Signature</td>
<td>Return Type</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------</td>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>doca_apsh_process ***processes, int *processes_size);</td>
<td>h_process</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>int – size of the returned array</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>doca_error status</td>
</tr>
</tbody>
</table>

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_process *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_envvar_info_get(struct doca_apsh_envvar *envvar, enum doca_apsh_envvar_attr attr);
doca_apsh_handle_info_get(struct doca_apsh_handle *handle, enum doca_apsh_handle_attr attr);
```
The return type of the attribute getter can be found in `doca_apsh_attr.h`.

Usage example:

```c
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);
```

### App Shield Initialization and Teardown

To use App Shield, users must initialize and configure two main structs. This section presents these structs and explains how to interact with them.

#### doca_apsh_ctx

`doca_apsh_ctx` is the basic struct used by App Shield which defines the DMA device 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 device to use:

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

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);
   ```

---

**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:

   ```
   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);
   ```

2. To configure different attributes for the system instance:
- OS type – specifies the system’s OS type.

```c
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 DOCA Core.

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

```c
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:

```c
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:

```c
#include <doapsh.h>

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:

```c
#include <doapsh.h>

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`:

```c
#include <doapsh.h>

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:

```c
#include <doapsh.h>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pid (optional)</td>
<td>The process ID of the process we want to run attestation capability on</td>
</tr>
<tr>
<td>os (mandatory)</td>
<td>The OS type of the machine (i.e., Linux or Windows)</td>
</tr>
<tr>
<td>find_kpgd (optional)</td>
<td>Relevant for Linux OS only, AS flag to enable/disable creating kpgd_file.conf. Default 0.</td>
</tr>
<tr>
<td>files (mandatory)</td>
<td>A list of files for the tool to create. File options: hash, symbols, memregions, kpgd_file (only relevant for Linux).</td>
</tr>
</tbody>
</table>

*Note* Make sure that the value set is appropriate for your setup.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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.

**DOCA App Shield Samples**

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

⚠️ **Info**

All the DOCA samples described in this section are governed under the BSD-3 software license agreement.

**Sample Prerequisites**

Follow the guidelines in section "Prerequisites" then copy the generated JSON files, symbols.json and mem_regions.json, to the /tmp/ directory.

**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 samples.

2. To build a given sample:

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

**Note**

The binary `doca_<sample_name>` will be created under `/tmp/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 (numeric) log level for the program: 10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE
- `--sdk-log-level`  Set the SDK (numeric) log level for the program: 10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE
- `--json <path>`  Parse all command flags from an input json file

**Program Flags:**
- `-p, --pid`  Process ID of process to be analyzed
- `-f, --vuid`  VUID of the System device
- `-d, --dma`  DMA device name
4. For additional information per sample, use the `-h` option:

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

## Samples

### Aps h Lib s 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.
7. Getting the list of system process using Apsh API and searching for a specific process with the given PID.
8. Getting 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.
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.
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 modules 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`
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.
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_pplist/apsh_pplist_sample.c
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.
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 \texttt{doca\_apsh\_threads\_get} Apsh API call.
9. Querying the threads for up to 3 selected fields using \texttt{doca\_apsh\_thread\_info\_get} Apsh API call.
10. Printing thread attributes to the terminal.
11. Cleaning up.

References:

- \texttt{/opt/mellanox/doca/samples/doca\_apsh/apsh\_threads\_get/apsh\_threads\_get\_sample.c}
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.
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
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.

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.
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.

Note

This sample works only on target systems with Windows OS.

The sample logic includes:
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

**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.

ℹ️ **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.
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`

**Apsh Containers Get**

This sample illustrates how to properly initialize DOCA App Shield and use its API to get the list of running containers on a monitored system, as well as getting a list of processes for each container.

⚠️ **Note**

This sample works only on target systems with Linux 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 using specific VUID.

5. Creating DOCA Apsh system handler.


7. Getting the list of containers running on the system using \texttt{doca\_apsh\_containers\_get} Apsh API call.

8. Querying the containers for container ID attribute using \texttt{doca\_apsh\_container\_info\_get} Apsh API call.

9. Getting list of processes for each container using \texttt{doca\_apsh\_container\_processes\_get} Apsh API call.

10. Printing the attributes of up to 5 processes to the terminal.

11. Cleaning up.

References:

- \texttt{/opt/mellanox/doca/samples/doca\_apsh/apsh\_containers\_get/apsh\_containers\_get\_sample.c}
- \texttt{/opt/mellanox/doca/samples/doca\_apsh/apsh\_containers\_get/apsh\_containers\_get\_main.c}
- \texttt{/opt/mellanox/doca/samples/doca\_apsh/apsh\_containers\_get/meson.build}
- \texttt{/opt/mellanox/doca/samples/doca\_apsh/apsh\_common.c; /opt/mellanox/doca/samples/doca\_apsh/apsh\_common.h}