

# **DOCA YARA Inspection Application Guide**

## Table of contents

Introduction
System Design
Application Architecture
DOCA Libraries
Limitations
Compiling the Application
Compiling All Applications
Compiling Only the Current Application
Troubleshooting
Running the Application
Prerequisites
Application Execution
Command Line Flags
Troubleshooting
Application Code Flow
References

This guide provides YARA inspection implementation on top of NVIDIA® BlueField® DPU.

### Introduction

YARA inspection monitors all processes in the host system for specific YARA rules using the <u>DOCA App Shield</u> library.

This security capability helps identify malware detection patterns in host processes from an independent and trusted DPU. This is an innovative Intrusion Detection System (IDS) as it is designed to run independently on the DPU's Arm cores without hindering the host.

This DOCA App Shield based application provides the capability to read, analyze, and authenticate the host (bare metal/VM) memory directly from the DPU.

Using the library, this application scans host processes and looks for pre-defined YARA rules. After every scan iteration, the application indicates if any of the rules matched. Once there is a match, the application reports which rules were detected in which process. The reports are both printed to the console and exported to the <u>DOCA Telemetry</u> <u>Service</u> (DTS) using inter-process communication (IPC).

This guide describes how to build YARA inspection using the DOCA App Shield library which leverages DPU abilities such as hardware-based DMA, integrity, and more.

## j) Note

As the DOCA App Shield library only supports the YARA API for Windows hosts, this application can only be used to inspect Windows hosts.

## System Design

The host's involvement is limited to generating the required ZIP and JSON files to pass to the DPU. This is done before the app is triggered, when the host is still in a "safe" state.

Generating the needed files can be done by running DOCA App Shield's doca\_apsh\_config.py tool on the host. See <u>DOCA App Shield</u> for more info.

— — → DMA read



### **Application Architecture**

The user creates the ZIP and JSON files using the DOCA tool doca\_apsh\_config.py and copies them to the DPU.

The application can report YARA rules detection to the:

- File
- Terminal
- DTS



- 1. The files are generated by running doca\_apsh\_config.py on the host against the process at time zero.
- 2. The following steps recur at regular time intervals:
  - 1. The YARA inspection app requests a list of all apps from the DOCA App Shield library.
  - 2. The app loops over all processes and checks for YARA rules match using the DOCA App Shield library.
  - 3. If YARA rules are found (1 or more), the YARA attestation app reports results with a timestamp and details about the process and rules to:
    - Local telemetry files a folder and files representing the data a real DTS would have received



DOCA log

- DTS IPC interface (even if no DTS is active)
- 3. The App Shield agent exits on first YARA rule detection.

### **DOCA Libraries**

This application leverages the following DOCA libraries:

- DOCA App Shield
- <u>2024-10-09\_07-10-18\_DOCA Telemetry</u>

Refer to their respective programming guide for more information.

### Limitations

- The application is only available on Ubuntu 22.04 environments
- The application only supports the inspection of Windows hosts

### **Compiling the Application**

### (i) Info

Please refer to the <u>DOCA Installation Guide for Linux</u> for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

#### Tip

For more information about the applications as well as development and compilation tips, refer to the <u>DOCA Reference Applications</u> page.

The sources of the application can be found under the application's directory: /opt/mellanox/doca/applications/yara\_inspection/.

### **Compiling All Applications**

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

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

### i) Info

doca\_yara\_inspection is created under
/tmp/build/yara\_inspection/.

### **Compiling Only the Current Application**

To directly build only the YARA inspection application:

cd /opt/mellanox/doca/applications/

```
meson /tmp/build -Denable_all_applications=false -
Denable_yara_inspection=true
ninja -C /tmp/build
```

(j) Info

doca\_yara\_inspection is created under
/tmp/build/yara\_inspection/.

Alternatively, one can set the desired flags in the meson\_options.txt file instead of providing them in the compilation command line:

- 1. Edit the following flags in
   /opt/mellanox/doca/applications/meson\_options.txt:
  - Set enable\_all\_applications to false
  - Set enable\_yara\_inspection to true
- 2. Run the following compilation commands :

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

### i Info

doca\_yara\_inspection is created under
/tmp/build/yara\_inspection/.

### Troubleshooting

Refer to the <u>DOCA Troubleshooting</u> for any issue encountered with the compilation of the application .

### **Running the Application**

### Prerequisites

- 1. Configure the BlueField's firmware
  - 1. On the BlueField system, configure the PF base address register and NVME emulation. Run:

dpu> mlxconfig -d /dev/mst/mt41686\_pciconf0 s
PF\_BAR2\_SIZE=2 PF\_BAR2\_ENABLE=1 NVME\_EMULATION\_ENABLE=1

- 2. Perform a <u>BlueField system reboot</u> for the <u>mlxconfig</u> settings to take effect.
- 3. This configuration can be verified using the following command:

dpu> mlxconfig -d /dev/mst/mt41686\_pciconf0 q | grep -E
"NVME|BAR"

- 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</pre>
```

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

• For CentOS:

```
host> yum install --enablerepo=base-debuginfo kernel-
devel-$(uname -r) kernel-debuginfo-$(uname -r) kernel-
debuginfo-common-$(uname -m)-$(uname -r)
```

- No action is needed for Windows
- 3. Perform IOMMU passthrough. This stage is only needed on some of the cases where IOMMU is not enabled by default (e.g., when the host is using an AMD CPU).

### (i) Note

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

```
host> dmesg
[ 3839.822897] mlx5_core 0000:81:00.0: AMD-Vi:
Event logged [I0_PAGE_FAULT domain=0x0047
address=0x2a0aff8 flags=0x0000]
```

 Locate your OS's grub file (most likely /boot/grub/grub.conf, /boot/grub2/grub.cfg, or /etc/default/grub) and open it for editing. Run:

host> vim /etc/default/grub

• Search for the line defining GRUB\_CMDLINE\_LINUX\_DEFAULT and add the argument iommu=pt. For example:

GRUB\_CMDLINE\_LINUX\_DEFAULT="iommu=pt <intel/amd>\_iommu=on"

• Run:

#### i) Note

Prior to performing a power cycle, make sure to do a <u>graceful shutdown</u>.

For Ubuntu:

host> sudo update-grub host> ipmitool power cycle

• For CentOS:

```
host> grub2-mkconfig -o /boot/grub2/grub.cfg
host> ipmitool power cycle
```

- For Windows targets: Turn off Hyper-V capability.
- 4. The DOCA App Shield library uses hugepages for DMA buffers. Therefore, the user must allocate 42 huge pages.
  - 1. Run:

```
dpu> nr_huge=$(cat
/sys/devices/system/node/node0/hugepages/hugepages-
2048kB/nr_hugepages)
    nr_huge=$((42+$nr_huge))
    echo $nr_huge | sudo tee -a
/sys/devices/system/node/node0/hugepages/hugepages-
2048kB/nr_hugepages
```

2. Create the ZIP and JSON files. Run:

```
target-system> cd /opt/mellanox/doca/tools/
target-system> python3 doca_apsh_config.py <pid-of-
process-to-monitor> --os <windows/linux> --path <path to
dwarf2json executable or pdbparse-to-json.py>
target-system> cp /opt/mellanox/doca/tools/*.* <shared-
folder-with-baremetal>
dpu> scp <shared-folder-with-baremetal>/* <path-to-app-
shield-binary>
```

If the target system does not have DOCA installed, the script can be copied from the BlueField.

The required dwaf2json and pdbparse-to-json.py are not provided with DOCA.



If the kernel and process <u>.exe</u> have not changed, there no need to redo this step.

### **Application Execution**

The YARA inspection application is provided in source form. Therefore, a compilation is required before the application can be executed.

1. Application usage instructions:

```
Usage: doca_yara_inspection [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>
  -j, --json <path>
                                     Parse all command flags
from an input json file
Program Flags:
  -m, --memr <path>
                                     System memory regions map
  -f, --vuid
                                     VUID of the System device
  -d, --dma
                                     DMA device name
  -o, --osym <path>
                                     System OS symbol map path
  -t, --time <seconds>
                                     Scan time interval in
seconds
```



```
./doca_yara_inspection -m mem_regions.json -o symbols.json -f
MT2125X03335MLNXS0D0F0VF1 -d mlx5_0 -t 3
```

```
(i) Note
```

All used identifiers (-f and -d flags) should match the identifier of the desired devices.

### **Command Line Flags**

Flag Type	Shor t Flag	Long Flag	Description
Gene ral flags	h	help	Prints a help synopsis
	V	vers ion	Prints program version information
	1	log- leve l	<ul> <li>Set the log level for the application:</li> <li>DISABLE=10</li> <li>CRITICAL=20</li> <li>ERROR=30</li> <li>WARNING=40</li> <li>INFO=50</li> <li>DEBUG=60</li> <li>TRACE=70 ( requires compilation with TRACE log level support )</li> </ul>
	N/A	sdk- log- leve l	Sets the log level for the program: DISABLE=10 CRITICAL=20 ERROR=30 WARNING=40 INFO=50 DEBUG=60 TRACE=70
	j	json	Parse all command flags from an input JSON file
Progr am flags	m	memr	Path to the pre-generated mem_regions.json file transferred from the host
	f	pcif	System PCIe function vendor unique identifier (VUID) of the VF/PF exposed to the target system. Used for DMA operations. To obtain this argument, run: target-system> lspci -vv   grep "\[VU\] Vendor specific:"
			Example output:
			[VU] Vendor specific: MT2125X03335MLNXS0D0F0

Flag Type	Shor t Flag	Long Flag	Description
			[VU] Vendor specific: MT2125X03335MLNXS0D0F1
			Two VUIDs are printed for each DPU connected to the target system. The first is of the DPU on pf0 and the second is of the DPU on port pf1.
			(i) Note Running this command on the DPU outputs VUIDs with an additional "EC" string in the middle. You must remove the "EC" to arrive at the correct VUID.
			The VUID of a VF allocated on PFO/1 is the VUID of the PF with an additional suffix, VF <vf-number>, where vf-number is the VF index +1. For example, for the output in the example above: • PFO VUID = MT2125X03335MLNXS0D0F0 • PF1 VUID = MT2125X03335MLNXS0D0F1 • VUID of VF0 on PF0 = MT2125X03335MLNXS0D0F0VF1</vf-number>
			VUIDs are persistent even on reset.
	d	dma	DMA device name to use
	0	osym	Path to the pre-generated symbols.json file transferred from the host
	t	time	Number of seconds to sleep between scans

## (i) Info

Refer to <u>DOCA Arg Parser</u> for more information regarding the supported flags and execution modes.

### Troubleshooting

Refer to the  $\underline{\text{DOCA Troubleshooting}}$  for any issue encountered with the installation or execution of the DOCA applications .

### **Application Code Flow**

- 1. Parse application argument.
  - 1. Initialize arg parser resources and register DOCA general parameters.

doca\_argp\_init();

2. Register application parameters.

register\_apsh\_params();

3. Parse the arguments.

```
doca_argp_start();
```

- 2. Initialize DOCA App Shield lib context.
  - 1. Create lib context.

```
doca_apsh_create();
```

2. Set DMA device for lib.

```
open_doca_device_with_ibdev_name();
doca_apsh_dma_dev_set();
```

3. Start the context

doca\_apsh\_start();
apsh\_system\_init();

3. Initialize DOCA App Shield lib system context handler.

1. Get the representor of the remote PCIe function exposed to the system.

open\_doca\_device\_rep\_with\_vuid();

2. Create and start the system context handler.

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

4. Telemetry initialization.

#### telemetry\_start();

- 1. Initialize a new telemetry schema.
- 2. Register YARA type event.
- 3. Set up output to file (in addition to default IPC).
- 4. Start the telemetry schema.
- 5. Initialize and start a new DTS source with the <a href="mailto:gethostname(">gethostname()</a> name as source ID.
- 5. Loop until YARA rule is matched.
  - 1. Get all processes from the host.

doca\_apsh\_processes\_get();

2. Check for YARA rule identification and send a DTS event if there is a match.

6. Telemetry destroy.

```
telemetry_destroy();
```

7. YARA inspection clean-up.

```
doca_apsh_system_destroy();
doca_apsh_destroy();
doca_dev_close();
doca_dev_rep_close();
```

8. Arg parser destroy.

doca\_argp\_destroy();

### References

/opt/mellanox/doca/applications/yara\_inspection/

<b>Notice</b><br/>br/><br/>br/>>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 ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes 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.<br/><br/>NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.<br/>br/>Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.<br/> <br/>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.<br/>>br/><br/>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.<br/>shr/><br/>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.<br/>sch/>sch/>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.<br/>schr/><br/>kr/><br/>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.<br/><br/><br/><br/><br/> 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 trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.<br/>

© Copyright 2025, NVIDIA. PDF Generated on 05/05/2025