Red Hat OpenShift

nShield® HSM Integration Guide

03 Dec 2021

© 2021 Entrust Corporation. All rights reserved.

Contents

1. Introduction

This guide describes how to integrate a Red Hat OpenShift cluster with an Entrust nShield Hardware Security Module (HSM), using the nShield Container Option Pack (nSCOP).

OpenShift is an application hosting platform by Red Hat. It makes it easy for developers to quickly build, launch, and scale container-based web applications in a public cloud environment. nSCOP allows application developers, in the container-based environment of OpenShift, to access the cryptographic functionality of an HSM.

1.1. Integration architecture

1.1.1. OpenShift cluster and HSM

In this integration, a Red Hat OpenShift cluster is deployed on a single VM using Red Hat CodeReady Containers. Container images are used from a third-party cloud registry.

1.1.2. Container images

Two container images were created for the purpose of this integration: a hardserver container, and one application container. These images are stored in an external registry to be deployed to OpenShift:

- `cv-nshield-hwsp-redhat`

  A hardserver container image that controls communication between the HSM(s) and the application containers. One or more hardserver containers are required per deployment, depending on the number of HSMs and the number and types of application containers.

- `cv-nshield-app-redhat`

  Application container image to run nShield commands. It is a Red Hat Universal Base Image container, in which Security World software is installed.

You can also create containers that contain your application. For instructions, see the *nShield Container Option Pack User Guide*.

1.2. Product configurations

Entrust has successfully tested the integration of an nShield HSM with Red Hat OpenShift in the following configurations:

Product	Version
Base OS	RHEL 8.4.18
CodeReady Containers	1.35.0
OpenShift	4.9.5
VMware	ESXi 7.0.1
nShield HSM	Connect XC, Connect +
nShield Image	Connect XC v12.60.2, Connect + v12.60.10
nShield Firmware	Connect XC v12.50.11, Connect + v12.50.8
nShield Software	12.71.0, 12.80.4
nSCOP	1.1.1

1.3. Requirements

1.3.1. Before starting the integration process

Familiarize yourself with:

- The documentation for the nShield HSM.
- The *nShield Container Option Pack User Guide*.
- The documentation and setup process for Red Hat OpenShift.

1.3.2. Minimum resource requirements

The VM running CodeReady Container and the OpenShift cluster node in this deployment were configured using the following specifications:

Machine	vCPU	Virtual RAM	Storage
VM Running CodeReady Containers	4	16 GB	100 GB

2. Procedures

2.1. Prerequisites

Before you can use nSCOP container images in OpenShift, you must complete the following steps:

1. Install OpenShift on a supported configuration. See the documentation provided by Red Hat.
2. Set up the HSM. See the *Installation Guide* for your HSM.
3. Configure the HSM(s) to have the IP address of your container host machine as a client. This is the host used to deploy CodeReady Containers in this integration.
4. Load an existing Security World or create a new one on the HSM. Copy the Security World and module files to your container host machine at a directory of your choice. Instructions on how to copy these two files into an OpenShift persistent volume accessible by the application containers are given in Copy the Security World and module files to the cluster persistent volume.

   For more information on configuring and managing nShield HSMs, Security Worlds, and Remote File Systems, see the *User Guide* for your HSM(s).

2.2. Install nSCOP

The installation process involves extracting the nSCOP tarball into `/opt/nscop`.

1. Make the installation directory:

   ```
   % sudo mkdir -p /opt/nscop
   ```

2. Extract the tarball:

   ```
   % sudo tar -xvf NSCOPTARFILE -C /opt/nscop
   ```

2.3. Build the nSCOP containers

This process will build nSCOP containers for the hardserver and application. Please note that:

- This guide uses the "red hat" flavor of the container.
- Docker needs to be installed for this process to be successful.

- You will also need the Security World ISO file to be able to build nSCOP.
- To configure the containers, you will need the HSM IP address, world and module files.
- The example below uses version 12.71 of the Security World client.

To build the nSCOP containers:

1. Mount the Security World Software ISO file:

```
% sudo mount -t iso9660 -o loop ISOFILE.iso /mnt/iso1
```

2. Build the nShield container for the hardserver and application (Red Hat):

```
% cd /opt/nscop
% sudo ./make-nshield-hwsp --tag nshield-hwsp-redhat:12.71 /mnt/iso1
% sudo ./make-nshield-application --tag nshield-app-redhat:12.71 /mnt/iso1
```

3. Validate the images have been built:

```
% sudo docker images
```

You should see the 2 images listed.

4. Build the `nshield-hwsp` configuration. You will need the HSM IP address during this process.

```
% cd /opt/nscop
% sudo mkdir -p /opt/nscop/config1
% sudo ./make-nshield-hwsp-config --output /opt/nscop/config1/config HSM_IP_ADDRESS
% cat /opt/nscop/config1/config
```

5. Build the nShield Application Container Security World. You will need the HSM world and module file during this process.

```
% sudo mkdir -p /opt/nscop/app1/kmdata/local
% sudo cp world /opt/nscop/app1/kmdata/local/.
% sudo cp module_<ESN> /opt/nscop/app1/kmdata/local/.
% ls /opt/nscop/app1/kmdata/*
```

6. Create a Docker socket:

```
% sudo docker volume create socket1
```

7. Check if the hardserver container can access the HSM using sockets:

```
% sudo docker run -v /opt/nscop/config1:/opt/nfast/kmdata/config:ro -v socket1:/opt/nfast/sockets nshield-hwsp-redhat:12.71 &
% dmountpoint=`sudo docker volume inspect --format '{{ .Mountpoint }}' socket1`
% export NFAST_SERVER=$dmountpoint/nserver
% /opt/nfast/bin/enquiry -m0
```

8. Check if the Container Application can access using the Security World:

```
% sudo docker run --rm -it -v /opt/nscop/app1/kmdata:/opt/nfast/kmdata:ro -v socket1:/opt/nfast/sockets -it nshield-app-redhat:12.71 /opt/nfast/bin/enquiry
```

2.4. Register the nSCOP containers to an external registry

In this guide, the external registry is *<external-docker-registry-IP-address>*. Register the Docker container images to a Docker registry so they can be used when you deploy Kubernetes pods into the OpenShift cluster.

Distribution of the nShield Container Image is not permitted because the software components are under strict export controls.

1. Log in to the Docker registry using your site credentials:

```
% sudo docker login -u YOURUSERID https://<external-docker-registry-IP-address>
```

2. Tag images:

```
% sudo docker tag nshield-hwsp-redhat:12.71" <external-docker-registry-IP-address>/nshield-hwsp-redhat
% sudo docker tag nshield-app-redhat:12.71" <external-docker-registry-IP-address>/nshield-app-redhat
```

3. Push images:

```
% sudo docker push <external-docker-registry-IP-address>/cv-nshield-hwsp-redhat
% sudo docker push <external-docker-registry-IP-address>/cv-nshield-app-redhat
```

4. Remove local images:

```
% sudo docker rmi <external-docker-registry-IP-address>/cv-nshield-hwsp-redhat
% sudo docker rmi <external-docker-registry-IP-address>/cv-nshield-app-redhat
```

5. Show images:

```
% sudo docker images
```

6. Pull images from the registry:

```
% sudo docker pull <external-docker-registry-IP-address>/cv-nshield-hwsp-redhat
% sudo docker pull <external-docker-registry-IP-address>/cv-nshield-app-redhat
```

7. Show images:

```
% sudo docker images
```

2.5. Deploying the nSCOP images

This section describes how to deploy nSCOP images to call `nfast` binaries using `hwsp` and application container images. The deployment consists of three pods, each of which contains a hardserver and an application container. Each application container executes a single nShield command.

A persistent volume is set up in the OpenShift cluster file system. This persistent volume contains the Security World and module files. The hardserver container and application containers will have access to these files.

2.5.1. Create the project

This section describes how to deploy the nSCOP image:

- `project.yaml`

  Contains the container project name. You can change the project name, but then that change will also need to be propagated across the namespace entry in the other YAML files, as well as the command examples in the instructions in this guide.

- `cm.yaml`

  Configuration map that contains the ESN and the IP address of the HSM. Edit this file to match your system.

See *Sample YAML files* that you can adapt to your system.

This guide uses CodeReady Container to deploy the OpenShift cluster. Once deployed, by default, it uses https://api.crc.testing:6443 for the cluster API address.

1. Log in to the server and launch a terminal window.
2. Copy the `project.yaml` and `cm.yaml` files to a local directory on the server. Edit the files to match your system.
3. Log in to the container platform:

```
% oc login -u kubeadmin -p <container-password> https://api.crc.testing:6443

Logged into "https://api.crc.testing:6443" as "kubeadmin" using existing credentials.

You have access to 64 projects, the list has been suppressed. You can list all projects with 'oc projects'

Using project "default".
```

4. Create a container project:

```
% oc create -f project.yaml

project.project.openshift.io/nscop-test created
```

5. Change from the current container project to the new one:

```
% oc project nscop-test

Now using project "nscop-test" on server "https://api.crc.testing:6443".
```

6. Create the configuration map for the HSM details:

```
% oc create -f cm.yaml

configmap/config created
```

7. Verify the HSM configuration:

```
% oc get configmap

NAME                       DATA   AGE
config                     1      74s
kube-root-ca.crt           1      3m16s
openshift-service-ca.crt   1      3m16s

% oc describe configmap/config

Name:         config
Namespace:    nscop-test
Labels:       <none>
Annotations:  <none>

Data
===
config:
----
syntax-version=1

[nethsm_imports]
local_module=1
remote_esn=BD10-03E0-D947
remote_ip=xx.xxx.xxx.xx
remote_port=9004
keyhash=2dd7c10c73a3c5346d1246e6a8cf6766a7088e41
privileged=0


BinaryData
===

Events:  <none>
```

8. If you have not yet enrolled the server as a client of the HSM, enroll it now. For instructions, see the *User Guide* for your HSM.

2.5.2. Create the registry secrets inside the cluster

At the beginning of this process you created nSCOP Docker containers and pushed them to the Docker registry. You must now configure the OpenShift cluster to authenticate to the Docker registry.

1. Create the secret in the cluster:

```
% oc create secret generic regcred --from-file=.dockerconfigjson=$HOME/.docker/config.json
--type=kubernetes.io/dockerconfigjson
```

2. Confirm that the secret was created:

```
% oc get secret regcred
```

2.5.3. Create the cluster persistent volumes

This section describes how the persistent volume is created in the OpenShift cluster. The following folder will contain the Security World and module files:

- `/opt/nfast/kmdata`

The following YAML files are used to create and claim the persistent volume:

- `pv_nfast_sockets_definition.yaml`
- `pv_nfast_sockets_claim.yaml`
- `pv_nfast_kmdata_definition.yaml`
- `pv_nfast_kmdata_claim.yaml`

See *Sample YAML files* that you can adapt to your system.

1. Log in to the container platform and create the persistent volume and claims:

```
% oc create -f pv_nfast_sockets_definition.yaml

persistentvolume/nfast-sockets created

% oc create -f pv_nfast_kmdata_definition.yaml

persistentvolume/nfast-kmdata created

% oc create -f pv_nfast_sockets_claim.yaml

persistentvolumeclaim/nfast-sockets created

% oc create -f pv_nfast_kmdata_claim.yaml

persistentvolumeclaim/nfast-kmdata created
```

2. Verify that the persistent volume has been created:

```
% oc get pv

NAME            CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM
STORAGECLASS   REASON   AGE
nfast-kmdata    1G         RWO            Retain           Bound       nscop-test/nfast-kmdata
manual                   2m53s
nfast-sockets   1G         RWO            Retain           Bound       nscop-test/nfast-sockets
manual                   11m
```

3. Verify that the claim has been created:

```
oc get pvc

NAME            STATUS   VOLUME          CAPACITY   ACCESS MODES   STORAGECLASS   AGE
nfast-kmdata    Bound    nfast-kmdata    1G         RWO            manual         96s
nfast-sockets   Bound    nfast-sockets   1G         RWO            manual         10m
```

2.5.4. Copy the Security World and module files to the cluster persistent volume

This section describes how to populate the `nfast-kmdata` persistent volume with the Security World and module files:

- `/opt/nfast/kmdata/local/world`
- `/opt/nfast/kmdata/local/module_<ESN>`

A dummy application container provides access to the persistent volume. This enables you to copy Security World and module files from the host server to the OpenShift cluster.

The following files are required to perform these steps:

- `pod_hwsp.yaml`
- `pod_dummy.yaml`

See *Sample YAML files* that you can adapt to your system.

The container running the hardserver needs to run at this time.

1. Log in to the container platform and create the hardserver container and dummy application container:

```
% oc create -f pod_hwsp.yaml

pod/nscop-hwsp-d9f9w created

% oc create -f pod_dummy.yaml

pod/nscop-test-dummy-wtcmz created

% oc get pods

NAME                    READY   STATUS    RESTARTS   AGE
nscop-hwsp-d9f9w        1/1     Running   0          4m7s
nscop-test-dummy-rqrqr  1/1     Running   0          8s
```

2. Create the `/opt/nfast/kmdata/local` directory in the cluster `nfast-kmdata` persistent volume:

```
% oc debug pod/nscop-test-dummy-rqrqr -- mkdir -p /opt/nfast/kmdata/local

Starting pod/nscop-test-dummy-rqrqr-debug, command was: sh -c sleep 3600
Removing debug pod ...
```

3. Copy the Security World and module files from the host directory to the cluster `nfast-kmdata` persistent volume:

```
% oc cp /opt/nfast/kmdata/local/world nscop-test/nscop-test-dummy-rqrqr:/opt/nfast/kmdata/local/world

% oc cp /opt/nfast/kmdata/local/module_<ESN> nscop-test/nscop-test-dummy-rqrqr:/opt/nfast/kmdata/local/.
```

4. Verify that the files have been copied:

```
% oc debug pod/nscop-test-dummy-rqrqr -- ls -al /opt/nfast/kmdata/local

Starting pod/nscop-test-dummy-rqrqr-debug, command was: sh -c sleep 3600
total 52
drwxr-xr-x. 2 root root    77 Nov 17 19:25 .
drwxr-xr-x. 3 root root    34 Nov 17 19:11 ..
-rwxrwxrwx. 1 root 1005  3488 Nov 17 19:25 module_BD10-03E0-D947
-rwxrwxrwx. 1 root 1005 39968 Nov 17 19:25 world
Removing debug pod ...
```

2.5.5. Create the application containers

This section describes how to create the application containers. A total of three application containers are created. Each application container executes a single nShield command. These are: `enquiry`, `sigtest`, and `nfkminfo`. The following YAML files are used to create the application containers:

- `pod_enquiry_redhat.yaml`
- `pod_sigtest_redhat.yaml`
- `pod_nfkminfo_redhat.yaml`

See *Sample YAML files* that you can adapt to your system.

To create and test the application containers:

1. Log in to the container platform and create the three application containers with the image:

```
% oc create -f pod_enquiry_redhat.yaml

pod/nscop-test-enquiry-8jqdj created

% oc create -f pod_sigtest_redhat.yaml

pod/nscop-test-sigtest-vjt7d created

% oc create -f pod_nfkminfo_redhat.yaml

pod/nscop-test-nfkminfo-7bc8n created
```

2. Wait a short period of time, then verify that the pods are running:

```
% oc get pods

NAME                         READY   STATUS    RESTARTS   AGE
nscop-hwsp-d9f9w             1/1     Running   0          23m
nscop-test-dummy-rqrqr       1/1     Running   0          19m
nscop-test-enquiry-8jqdj     1/1     Running   0          62s
nscop-test-nfkminfo-7bc8n    1/1     Running   0          22s
nscop-test-sigtest-vjt7d     1/1     Running   0          41s
```

3. Verify that nSCOP is running:

```
% oc get pods --all-namespaces | grep nscop

nscop-test                                  nscop-hwsp-d9f9w                                    1/1
Running     0              23m
nscop-test                                  nscop-test-dummy-rqrqr                              1/1
Running     0              19m
nscop-test                                  nscop-test-enquiry-8jqdj                            1/1
Running     0              106s
nscop-test                                  nscop-test-nfkminfo-7bc8n                           1/1
Running     0              66s
nscop-test                                  nscop-test-sigtest-vjt7d                            1/1
Running     0              85s
```

4. List the containers in the pods:

```
% oc get pods nscop-test-enquiry-8jqdj -n nscop-test -o jsonpath="{.spec.containers[*].image}"

registry.eselab.net/cv-nshield-app-redhat

% oc get pods nscop-test-sigtest-vjt7d -n nscop-test -o jsonpath="{.spec.containers[*].image}"

registry.eselab.net/cv-nshield-app-redhat

% oc get pods nscop-test-nfkminfo-7bc8n -n nscop-test -o jsonpath="{.spec.containers[*].image}"

registry.eselab.net/cv-nshield-app-redhat
```

2.5.6. Run the applications

The applications in this integration execute nShield commands. A correct response confirms that both the hardserver and the HSM are up and running, and that the HSM is available.

1. Log in to the container platform:

```
% oc login -u kubeadmin -p <container-password> https://api.crc.testing:6443:6443
```

2. To retrieve the log output of the `enquiry` container:

```
% oc logs pod/nscop-test-enquiry-8jqdj

Server:
 enquiry reply flags  none
 enquiry reply level  Six
```

```
 serial number        BD10-03E0-D947
 mode                 operational
 version              12.71.0
 speed index          15843
 rec. queue           368..566
 level one flags      Hardware HasTokens SupportsCommandState
 version string       12.71.0-353-f63c551, 12.50.11-270-fb3b87dd465b6f6e53d9f829fc034f8be2dafd13 2019/05/16 22:02:33
BST, Bootloader: 1.2.3, Security Processor: 12.50.11 , 12.60.10-708-ea4dc41d
 checked in           000000006053229a Thu Mar 18 09:51:22 2021
 level two flags      none
 max. write size      8192
 level three flags    KeyStorage
 level four flags     OrderlyClearUnit HasRTC HasNVRAM HasNSOPermsCmd ServerHasPollCmds FastPollSlotList HasSEE
HasKLF HasShareACL HasFeatureEnable HasFileOp HasLongJobs ServerHasLongJobs AESModuleKeys NTokenCmds
JobFragmentation LongJobsPreferred Type2Smartcard Serv\
erHasCreateClient HasInitialiseUnitEx AlwaysUseStrongPrimes Type3Smartcard HasKLF2
 module type code     0
 product name         nFast server
 device name
 EnquirySix version   4
 impath kx groups
 feature ctrl flags   none
 features enabled     none
 version serial       0
 level six flags      none
 remote server port   9004
 kneti hash           005429cd7e016a3407311081bbad84ec883fdd2d

Module #1:
 enquiry reply flags  UnprivOnly
 enquiry reply level  Six
 serial number        BD10-03E0-D947
 mode                 operational
 version              12.50.11
 speed index          15843
 rec. queue           43..150
 level one flags      Hardware HasTokens SupportsCommandState
 version string       12.50.11-270-fb3b87dd465b6f6e53d9f829fc034f8be2dafd13 2019/05/16 22:02:33 BST, Bootloader:
1.2.3, Security Processor: 12.50.11 , 12.60.10-708-ea4dc41d
 checked in           000000005cddcfe9 Thu May 16 21:02:33 2019
 level two flags      none
 max. write size      8192
 level three flags    KeyStorage
 level four flags     OrderlyClearUnit HasRTC HasNVRAM HasNSOPermsCmd ServerHasPollCmds FastPollSlotList HasSEE
HasKLF HasShareACL HasFeatureEnable HasFileOp HasLongJobs ServerHasLongJobs AESModuleKeys NTokenCmds
JobFragmentation LongJobsPreferred Type2Smartcard Serv\
erHasCreateClient HasInitialiseUnitEx AlwaysUseStrongPrimes Type3Smartcard HasKLF2
 module type code     12
 product name         nC3025E/nC4035E/nC4335N
 device name          Rt1
 EnquirySix version   7
 impath kx groups     DHPrime1024 DHPrime3072 DHPrime3072Ex
 feature ctrl flags   LongTerm
 features enabled     GeneralSEE StandardKM EllipticCurve ECCMQV AcceleratedECC HSMHighSpeed
 version serial       37
 connection status    OK
 connection info      esn = BD10-03E0-D947; addr = INET/10.194.148.36/9004; ku hash =
2dd7c10c73a3c5346d1246e6a8cf6766a7088e41, mech = Any
 image version        12.60.10-507-ea4dc41d
 level six flags      none
 max exported modules 100
 rec. LongJobs queue  42
 SEE machine type     PowerPCELF
 supported KML types  DSAp1024s160 DSAp3072s256
 using impath kx grp  DHPrime3072Ex
 active modes         UseFIPSApprovedInternalMechanisms AlwaysUseStrongPrimes
 hardware status      OK
```

3. To retrieve the log output of the `sigtest` container:

```
% oc logs pod/nscop-test-sigtest-vjt7d

 Hardware module #1 speed index 15843 recommended minimum queue 43
Found 1 module; using 43 jobs
Making 1024-bit RSAPrivate key on module #1;
  using Mech_RSApPKCS1 and PlainTextType_Bignum.
Generated and exported key from module #1.
Imported keys on module #1
 1, 800 320, 800 overall
 2, 5125 1922, 2562.5 overall
 3, 9477 2894, 3159 overall
 4, 14164 3611.2, 3541 overall
 5, 19905 4463.12, 3981 overall
 6, 25205 4797.87, 4200.83 overall
 7, 30571 5025.12, 4367.29 overall
 8, 34798 4705.87, 4349.75 overall
 9, 39759 4807.92, 4417.67 overall
 10, 45310 5105.15, 4531 overall
 11, 51052 5359.89, 4641.09 overall
 12, 55769 5102.74, 4647.42 overall
 13, 60482 4946.84, 4652.46 overall
 14, 65426 4945.7, 4673.29 overall
 15, 70741 5093.42, 4716.07 overall
 16, 76328 5290.85, 4770.5 overall
 17, 79571 4471.71, 4680.65 overall
 18, 83501 4255.03, 4638.94 overall
 19, 88857 4695.42, 4676.68 overall
...
```

4. Retrieve the log output of the `nfkminfo` container:

```
% oc logs pod/nscop-test-nfkminfo-7bc8n

World
 generation  2
 state       0x37a70008 Initialised Usable Recovery PINRecovery !ExistingClient RTC NVRAM FTO AlwaysUseStrongPrimes
!DisablePKCS1Padding !PpStrengthCheck !AuditLogging SEEDebug
 n_modules   1
 hknso       c9fb9e4cc5ec99fed1b92a766d90faccb639c7da
 hkm         235a046a49e6361470ba5a76dc1b3f745521dbd3 (type Rijndael)
 hkmwk       c2be99fe1c77f1b75d48e2fd2df8dffc0c969bcb
 hkre        4c8a5db06af0f51ab29a5ca5dacc72929f9f1b87
 hkra        6a1691c6d9a447ed90379fa49ebb6808a5b1851f
 hkmc        77bdd4664d681c9211b0fca71ced36351dfafc72
 hkp         2e1db243c305c1b0b9ff9a13b5039cce10119413
 hkrtc       6e100c78fd6e200e6fffd6419089d5dd34320097
 hknv        7d64bf068d30e0283219665b10aee498b061f85d
 hkdsee      059485679ff0a96048891bb6041cc11e4b0e9236
 hkfto       8aa8a2a902ffe4a6c83beaab3984aea1626b90d0
 hkmnull     0100000000000000000000000000000000000000
 ex.client   none
 k-out-of-n  1/1
 other quora m=1 r=1 p=1 nv=1 rtc=1 dsee=1 fto=1
 createtime  2021-06-21 20:55:08
 nso timeout 10 min
 ciphersuite DLf3072s256mAEScSP800131Ar1
 min pp      0 chars
 mode        none

Module #1
 generation 2
 state      0x2 Usable
 flags      0x0 !ShareTarget
 n_slots    4
 esn        BD10-03E0-D947
```

```
 hkml       7f6ee17f7d9c0c26297ee788a1e1a5e698040b5d

Module #1 Slot #0 IC 1
 generation    1
 phystype      SmartCard
 slotlistflags 0x2 SupportsAuthentication
 state         0x7 Error
 flags         0x0
 shareno       0
 shares
 error         UnlistedCard
No Cardset

Module #1 Slot #1 IC 0
 generation    1
 phystype      SoftToken
 slotlistflags 0x0
 state         0x2 Empty
 flags         0x0
 shareno       0
 shares
 error         OK
No Cardset

Module #1 Slot #2 IC 0
 generation    1
 phystype      SmartCard
 slotlistflags 0x80002 SupportsAuthentication DynamicSlot
 state         0x2 Empty
 flags         0x0
 shareno       0
 shares
 error         OK
No Cardset

Module #1 Slot #3 IC 0
 generation    1
 phystype      SmartCard
 slotlistflags 0x80002 SupportsAuthentication DynamicSlot
 state         0x2 Empty
 flags         0x0
 shareno       0
 shares
 error         OK
No Cardset

No Pre-Loaded Objects
```

Appendix A: Sample YAML files

A.1. project.yaml

```
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  annotations:
    openshift.io/description: ""
    openshift.io/display-name: test
    openshift.io/requester: kube:admin
  name: nscop-test
spec:
  finalizers:
  - kubernetes
status:
  phase: Active
```

A.2. cm.yaml

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: config
  namespace: nscop-test
data:
  config: |
    syntax-version=1

    [nethsm_imports]
    local_module=1
    remote_esn=BD10-03E0-D947
    remote_ip=10.194.148.36
    remote_port=9004
    keyhash=2dd7c10c73a3c5346d1246e6a8cf6766a7088e41
    privileged=0
```

A.3. pv_nfast_kmdata_definition.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfast-kmdata
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 1G
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: /opt/nfast/kmdata
```

A.4. pv_nfast_sockets_definition.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfast-sockets
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 1G
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: /opt/nfast/sockets
```

A.5. pv_nfast_kmdata_claim.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name : nfast-kmdata
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: local-storage
  resources:
    requests:
      storage: 1G
  storageClassName: manual
```

A.6. pv_nfast_sockets_claim.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name : nfast-sockets
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: local-storage
  resources:
    requests:
      storage: 1G
  storageClassName: manual
```

A.7. pod_dummy.yaml

```
kind: Pod
apiVersion: v1
metadata:
  generateName: nscop-test-dummy-
  namespace: nscop-test
  labels:
    app: nshield
spec:
  imagePullSecrets:
    - name: regcred
  containers:
    - name: nscop-redhat
      securityContext:
        privileged: true
      command:
        - sh
        - '-c'
        - sleep 3600
      image: >-
        <external-docker-registry-IP-address>/cv-nshield-app-redhat
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - mountPath: /opt/nfast/sockets
          name: nscop-sockets
        - mountPath: /opt/nfast/kmdata
          name: nscop-kmdata
  securityContext: {}
  volumes:
    - name: nscop-sockets
      persistentVolumeClaim:
        claimName: nfast-sockets
    - name: nscop-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
```

A.8. pod_enquiry_redhat.yaml

```
kind: Pod
apiVersion: v1
metadata:
  generateName: nscop-test-enquiry-
  namespace: nscop-test
  labels:
    app: nshield
spec:
  imagePullSecrets:
    - name: regcred
  containers:
    - name: nscop-redhat
      securityContext:
        privileged: true
      command:
        - sh
        - '-c'
        - /opt/nfast/bin/enquiry && sleep 3600
      image: >-
        <external-docker-registry-IP-address>/cv-nshield-app-redhat
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - mountPath: /opt/nfast/sockets
          name: nscop-sockets
        - mountPath: /opt/nfast/kmdata
          name: nscop-kmdata
  securityContext: {}
  volumes:
    - name: nscop-sockets
      persistentVolumeClaim:
        claimName: nfast-sockets
    - name: nscop-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
```

A.9. pod_sigtest_redhat.yaml

```yaml
kind: Pod
apiVersion: v1
metadata:
  generateName: nscop-test-sigtest-
  namespace: nscop-test
  labels:
    app: nshield
spec:
  imagePullSecrets:
    - name: regcred
  containers:
    - name: nscop-redhat
      securityContext:
        privileged: true
      command:
        - sh
        - '-c'
        - /opt/nfast/bin/sigtest && sleep 3600
      image: >-
        <external-docker-registry-IP-address>/cv-nshield-app-redhat
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - mountPath: /opt/nfast/sockets
          name: nscop-sockets
        - mountPath: /opt/nfast/kmdata
          name: nscop-kmdata
  securityContext: {}
  volumes:
    - name: nscop-sockets
      persistentVolumeClaim:
        claimName: nfast-sockets
    - name: nscop-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
```

A.10. pod_nfkminfo_redhat.yaml

```yaml
kind: Pod
apiVersion: v1
metadata:
  generateName: nscop-test-nfkminfo-
  namespace: nscop-test
  labels:
    app: nshield
spec:
  imagePullSecrets:
    - name: regcred
  containers:
    - name: nscop-redhat
      securityContext:
        privileged: true
      command:
        - sh
        - '-c'
        - /opt/nfast/bin/nfkminfo && sleep 3600
      image: >-
        <external-docker-registry-IP-address>/cv-nshield-app-redhat
      ports:
        - containerPort: 8080
          protocol: TCP
      resources: {}
      volumeMounts:
        - mountPath: /opt/nfast/sockets
          name: nscop-sockets
        - mountPath: /opt/nfast/kmdata
          name: nscop-kmdata
  securityContext: {}
  volumes:
    - name: nscop-sockets
      persistentVolumeClaim:
        claimName: nfast-sockets
    - name: nscop-kmdata
      persistentVolumeClaim:
        claimName: nfast-kmdata
```