Intel

OSC SDN Controller Plugin Integration SDK

Version – Draft 1.0

Table of Contents

[1 Revision History 2](#_Toc460425910)

[2 Introduction 3](#_Toc460425911)

[3 SDN Controller Plugin Integration 4](#_Toc460425912)

[3.1 Information APIs: 4](#_Toc460425913)

[3.2 Configuration APIs: 4](#_Toc460425914)

[3.3 Inspection Port Element: 5](#_Toc460425915)

[3.4 Inspection Hook APIs: 5](#_Toc460425916)

[3.5 Supported features APIs: 7](#_Toc460425917)

[3.6 Info Query APIs: 8](#_Toc460425918)

[4 Packaging 8](#_Toc460425919)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Rev | Author | Description | Date |
| 0.1 | Arvind Nadendla | Initial Version. | 04/8/2016 |
| 0.2 | Arvind Nadendla | Updating the SDK guides. | 5/25/2016 |
| 0.3 | Emanoel  Xavier | Making policy mapping optional for appliance managers. | 8/29/2016 |
| 0.4 | Arvind Nadendla | Added support for 2-arm security functions | 8/29/2016 |

# Introduction

Network Functions Virtualization (NFV) envisages the implementation of NFVs as software-only entities that run over the NFV Infrastructure (NFVI). Figure below illustrates the high-level NFV framework as referenced by ETSI. As such, three main working domains are identified in NFV:

* Virtualized Network Function (VNF), as the software implementation of a network function which is capable of running over the NFVI.
* NFV Infrastructure (NFVI), including the diversity of physical resources and how these can be virtualized. NFVI supports the execution of the VNFs.
* NFV Management and Orchestration, which covers the orchestration and lifecycle management of NFV’s that support the infrastructure virtualization. NFV Management and Orchestration focuses on all virtualization-specific management tasks necessary in the NFV framework



The NFV framework enables dynamic construction and management of VNF instances and the relationships between them regarding data, control, management, dependencies and other attributes. To this end, there are at least three architectural views of VNFs that are centered on different perspectives and contexts of a VNF. These perspectives include:

* a virtualization deployment/on-boarding perspective where the context can be a VM,
* a vendor-developed software package perspective where the context can be several inter-connected VMs and a deployment template that describes their attributes,
* An operator perspective where the context can be the operation and management of a VNF received in the form of a vendor software package.

# SDN Controller Plugin Integration

A third party that would be interested developing a SDN Controller integration plug-in to OSC needs to implement the four categories of APIs listed below:

This category deals with the information regarding plugin based on SDN controller in use. Here is the list of functions vendor needs to implement in for this categories of API:

OSC SDN Controller plug-in consist of a Jar file. Integrating vendors will need to create their plugin jar file by implementing OSC interfaces as listed below:

Import IscSdnControllerPlugin.jar into your project and implement SDN Controller Api Interfaces (com.intelsecurity.isc.plugin.controller.api.\*):

## Information APIs:

* Controller Plugin Name - String getName()
  + Name of SDN controller plugin.
  + This name will show up in SDN controller drop down when user is adding a new Virtualization Connector through OSC.
  + Must be unique cross all plugins.
  + OSC will call the following function to retrieve this information:
* Controller Plugin Version – String getVersion()
  + Plugin Version:
    - Metadata information
* Controller Status – Status getStatus()
  + Method returns Status object which will provide the following information
    - name: SDNController's Name
    - version: SDNController's version
    - controllerReady: This boolean flag if true represents that SDN controller is ready to serve

## Configuration APIs:

* Virtualization Connector– void setVirtualizationConnector(VirtualizationConnectorElement vc)
  + Set Virtualization Connector context through which authentication information can be obtained.
* Region - void setRegion(String region)
  + Set the region based on Data Center regions which is being used for this provisioning.

## Inspection Port Element:

Container for the below elements. This is to support ‘2-arm’ security functions where there are 2 ports for inspection one for ingress and another for egress

* NetworkPortElement getIngressPort()
* NetworkPortElement getEgressPort()

## Inspection Hook APIs:

An inspection hook is a network traffic redirection flow used to steer traffic from one port to another.

* Install Inspection Hook:

void installInspectionHook(NetworkPortElement inspectedPort, InspectionPortElement inspectionPort, Long tag, TagEncapsulationType encType, Long order, FailurePolicyType failurePolicyType) throws NetworkPortNotFoundException, Exception

* + Install a network hooks to inspect traffic of given network port by another network port which belongs to a security virtual function appliance.
  + Takes following:
    - Port to be inspected port
    - Port which will perform inspection
    - TAG. This parameter can have null value.
    - Encapsulation Type: VLAN, GRE etc. This parameter can have null value.
    - Flow Order used for Service chaining to define flow preference
* Remove Inspection Hook:

void removeInspectionHook(NetworkPortElement inspectedPort, InspectionPortElement inspectionPort) throws Exception

* + Remove given inspection hook on behalf of network port
* Remove all Inspection Hooks:

void removeAllInspectionHooks(NetworkPortElement inspectedPort) throws Exception

* + Remove all inspections hooks placed on behalf of network port
* Set Inspection Hook tag:

void setInspectionHookTag(NetworkPortElement inspectedPort, InspectionPortElement inspectionPort, Long tag) throws NetworkPortNotFoundException, Exception

* + Set data path policy tag of a specific inspection hook
  + Long value defining a security policy
* Get Inspection Hook tag:

Long getInspectionHookTag(NetworkPortElement inspectedPort, InspectionPortElement inspectionPort) throws NetworkPortNotFoundException, Exception

* + Return the current tag used for an installed inspection hook
* Set Inspection Hook Failure Policy

void setInspectionHookFailurePolicy(NetworkPortElement inspectedPort, InspectionPortElement inspectionPort, FailurePolicyType failurePolicyType) throws NetworkPortNotFoundException, Exception

* + Set the failure policy of an inspection hook
    - Fail Open
    - Fail Close
* Get Inspection Hook Failure Policy

FailurePolicyType getInspectionHookFailurePolicy(NetworkPortElement inspectedPort, InspectionPortElement inspectionPort) throws NetworkPortNotFoundException, Exception

* + Return the current failure policy used for an installed inspection hook
* Get Inspection Hook

InspectionHookElement getInspectionHook(NetworkPortElement inspectedPort, InspectionPortElement inspectionPort)throws NetworkPortNotFoundException, Exception

* + Return inspection port information for the given inspected port
  + Inspection Hook Element contains the following:
    - Hook ID
    - Tag
    - Order
    - Encapsulation Type
    - Failure Policy Type
    - Inspected Port
    - Inspection Port
* Update Inspection Hook

void updateInspectionHook(InspectionHookElement inspectionHook) throws NetworkPortNotFoundException, Exception

void updateInspectionHook(NetworkPortElement inspectedPort, InspectionPortElement inspectionPort, Long tag, TagEncapsulationType encType, Long order, FailurePolicyType failurePolicyType)throws NetworkPortNotFoundException, Exception

* + Update above mentioned information for an existing inspection hook.
* Register Inspection Hook

void registerInspectionPort(InspectionPortElement inspectionPort) throws NetworkPortNotFoundException, Exception

* + Register given port as an Inspection Port or Inspection Network Interface
* Get Inspection Port

InspectionPortElement getInspectionPort(InspectionPortElement inspectionPort)throws NetworkPortNotFoundException, Exception

* + Returns information of a registered inspection port for the given inspected port.
* Set Inspection Hook Order

void setInspectionHookOrder(NetworkPortElement inspectedPort, InspectionPortElement inspectionPort, Long order)throws NetworkPortNotFoundException, Exception

* + Sets order for this given Inspection Hook.
  + Used for service chaining
    - Order defines the position of this inspection port in service chain
* Get Inspection Hook Order

Long getInspectionHookOrder(NetworkPortElement inspectedPort, InspectionPortElement inspectionPort) throws NetworkPortNotFoundException, Exception

* + Returns order/position of an given inspection hook

## Supported features APIs:

* Redirection to remote hypervisor - boolean isOffboxRedirectionSupported()
  + True if SDN Controller supports traffic redirection to a VNF running on a remote hypervisor.
  + False otherwise
* Service Function Supported - boolean isServiceFunctionChainingSupported()
  + True if SDN Controller supports traffic redirection to multiple VNFs (service function chaining)
  + False otherwise
* Failure policy supported – boolean isServiceFunctionChainingSupported()
  + True if SDN Controller supports inspection port failure detection and can bypass traffic inspection
  + False otherwise
* Uses Virtualization Connectors Credentials - boolean isUsingProviderCreds()
  + True if SDN controller is using keystone credentials for communication.
  + False if SDN controller requires a different set of credentials then keystone.
* Supports Query Info - boolean isSupportQueryPortInfo()
  + True if SDN controller plugin can provide 5 tuple + timestamp information when asked.
  + False otherwise

## Info Query APIs:

* Query Port Info - HashMap<String, FlowPortInfo> queryPortInfo(HashMap<String, FlowInfo> portsQuery) throws Exception
  + Query port information based on 5 tuple flow info + timestamp.
  + Flow Port Info contains the following:
    - Source Port
    - Destination Port
    - Flow Info
  + Flow Info contains the following:
    - Source IP Address
    - Destination IP Address
    - Source Mac Address
    - Destination Mac Address
    - Source Port
    - Destination Port
    - Protocol ID
    - Flow time stamp

# Packaging

Once SDN controller Plugin Jar file is created, it is required to package it with metadata to properly import it to OSC. Follow the below steps to package OSC SDN controller Plugin:

* Create SDN controller plugin metadata file (file name must be called meta.json)
  + Plugin name. Must match value return by SdnControllerApi.getName()
  + Plugin Jar file name
  + SdnControllerApi implementer’s class name
  + Minimum OSC version requirement to support this plugin
  + Sample

{

"metaDataVersion": "1.0",

"pluginName": "Sample",

"jarFilename": "SampleSdnControllerPlugin.jar",

"sdnControllerApiClass": "com.mycompany.SampleMgrPlugin",

"minIscVersion": {

"major": 1,

"minor": 0,

"build": 2000

}

}

* Package SDN controller plugin jar along with the metadata file in zip format.