Virtual Network Broker

Functional Specification

Version 1.0

06/01/2017

Table of Contents

List of Figures

List of Tables

Record of Changes

Version	Date	Author	Description
V1.0	21/12/2015	Srinivas Addepalli, Kapil Sood	Initial Draft

Table 1. Record of changes

1. Abstract

Virtual Network Broker (VNB) also referred as SecMon is becoming priority for Enterprises that are moving sensitive workloads to clouds. With Enterprises adopting multi-cloud strategy, multiple security controls must work in various cloud environments with Enterprise controlled security console. Workload embedded with various security functions is thought to be one of the best solutions as the workload can be hosted anywhere with no dependency on the cloud operator.

VNB addresses one of the major analytics function: Security Monitoring (SecMon). Any data going-out / coming-in of the workload to any other workload is intercepted, filtered and monitored. It will allow to monitor the traffic flowing across the virtual machine running on OpenStack Compute Nodes. Hence Enterprises would have better idea of their network condition and traffic flowing across.

VNB supports NetFlow and SFlow as monitoring protocols. NetFlow services provide network administrators with access to information concerning IP flows within their data networks. Exported NetFlow data can be used for a variety of purposes, including network management and planning, enterprise accounting, and departmental chargebacks, data warehousing, combating Denial of Service (DoS) attacks, and data mining for marketing purposes. Currently there is support for Netflow in EMS as well as in SecMon Agent plugin.

While sFlow is an industry standard technology for monitoring high speed switched networks. It gives complete visibility into the use of networks enabling performance optimization, accounting/billing for usage, and defense against security threats**. Currently there is support for sFlow in EMS but it is disabled and when a plugin for same is added in SecMon Agent, EMS support can be enabled**.

The goals of this solution are:

1. To provide the solution catering two major monitoring and analysis protocols
    - NetFlow protocol
    - sFlow protocol (Support is present in EMS for configuration but protocol support needs to be provided as a plugin)
    - Raw packets forwarding
2. Framework allows different vendor to insert their pluggable modules to support filtering and monitoring based on different protocols and their configurations.

2. Solution Overview

Virtual Network Broker solution allows debugging cloud VM by monitoring VMs traffic from inside or outside cloud. It also allows different vendors to integrate their pluggable protocol to provide support for different monitoring and filtering needs. And it also allows different collector tools corresponding to the pluggable monitoring and filtering protocol.

Securing the traffic in transit, using virtual private network (VPN) implemented using IPsec standard, is used in many network deployments. Traditionally, IPsec VPN is used

- To secure the traffic across sites using IPsec Gateways in each site.
- Secure remote connectivity

IETF IPv6 specifications mandated the IPsec implementation to be part of each compute host, but left it to deployments to enable this. Due to that mandate, almost all hosts and virtual machine TCP/IP stacks have built-in IPsec.

IPsec at each host/VM also has additional advantages:

- Commodity hardware for security: No need for any specialized devices providing IPsec -> Cost benefit.
- Offload IPSec from VMs: Dedicated IPSec VMs can be created to offload the IPSec functionality from host/VMs. This type of offloading can be accelerated by using Hardware dedicated for crypto. Also result in the performance improvement of VMs in cloud environment.
- Using processor/server smarts: With increasing number of crypto needs, many server platforms are armed with crypto accelerators and they will be used to full extent.

2.1 Use Cases

Next, we describe the one of the use cases/verticals where this solution demands arises.

2.1.1 vEPC – Data Security and Monitoring

vEPC is a mobile-core network system that accommodates LTE access systems. vEPC allows following advantages:

1. Instead of using dedicated hardware, vEPC virtualizes all functions on a general-purpose IA server.
2. Software optimizes the allocation of server and network resources for each service on the virtualization platform.
3. vEPC can deliver all services on general-purpose IA servers, resulting in reduced maintenance cost.
4. vEPC works in an open NFVI (Network Functions Virtualization Infrastructure) environment.

Below figure shows the use case having vEPC VMs on Compute nodes of OpenStack. The dedicated IPSec VM are used to provide IPSec functionality. Also dedicated SecMon VM is used to monitor the vEPC VM's traffic. Multiple SecMon VM can be used between different vEPC VMs.

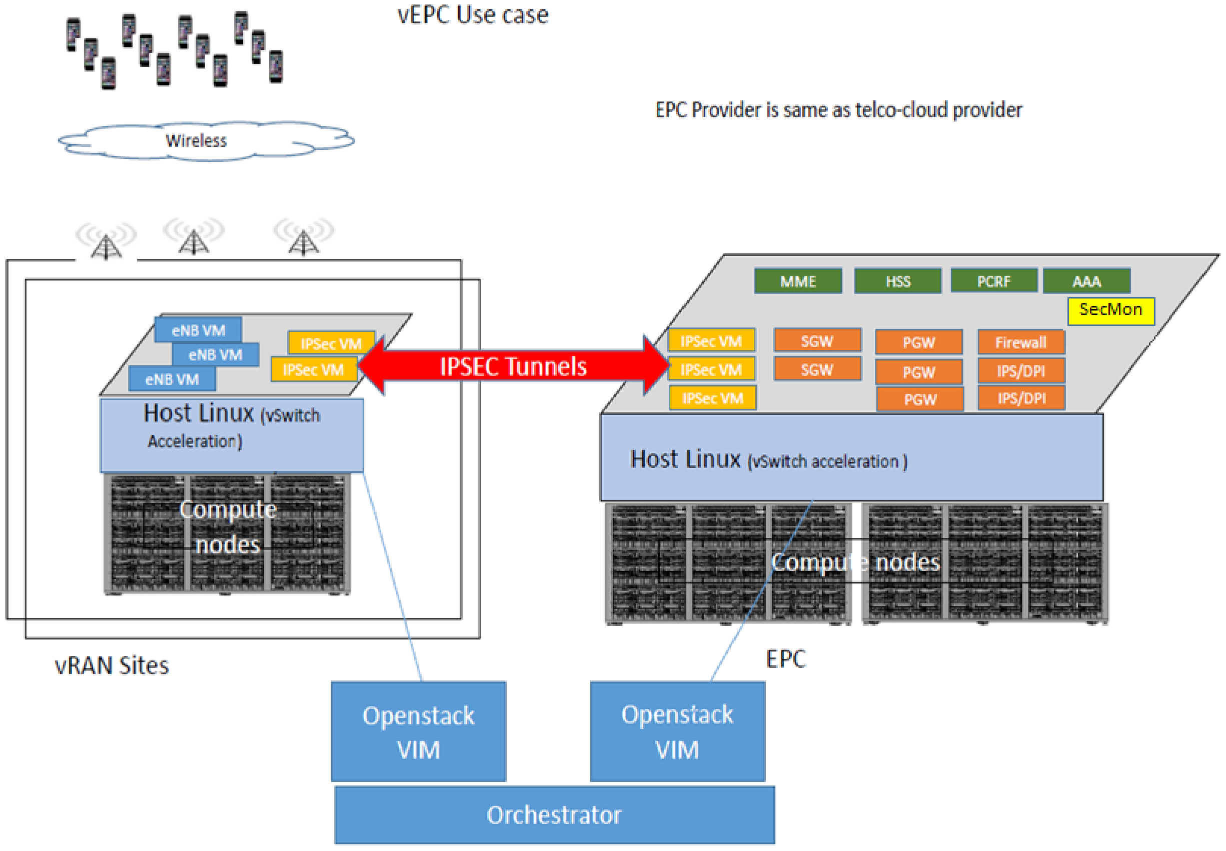


Figure 1 vEPC Use case setup with dedicated IPSEC VMs

3. Features Support

Solution comprises of components as described in detail in the following sub sections.

- Monitoring as a Service (SecMon EMS)
- IPsec function – IPsec VM

3.1 Feature overview

3.1.1 Monitoring as a Service (SecMon VNF)

Due to distributed nature of security functionality, it is necessary for administrators to know whether all the components of the security functions are working properly. Secure Monitoring functionality is to ensure that - there are no network attacks, hacks in the cloud network.

Monitoring as a service supports various functionalities such as security monitoring of cloud traffic, packet filtering, forwarding. All these are implemented based on the EMS architecture i.e. monitoring as a service can be viewed as a complete system which includes Security Monitoring Virtual Network Function (SecMon VNF) and EMS Server.

With this, it is possible to monitor tenant's own traffic, apply different filters and send to the security and monitoring analytics system for analysis. This is a requirement for Security Monitoring of VNF traffic. SecMon services are required in tenant's network (Monitoring of traffic, user defined selection of nodes for monitoring traffic), thereby architecture support running SecMon as VM, capable of running in any OpenStack Compute Node. It is also be possible to integrate SecMon in any legacy (non OpenStack nodes) based systems.

Following is high level view of SecMon & EMS:-

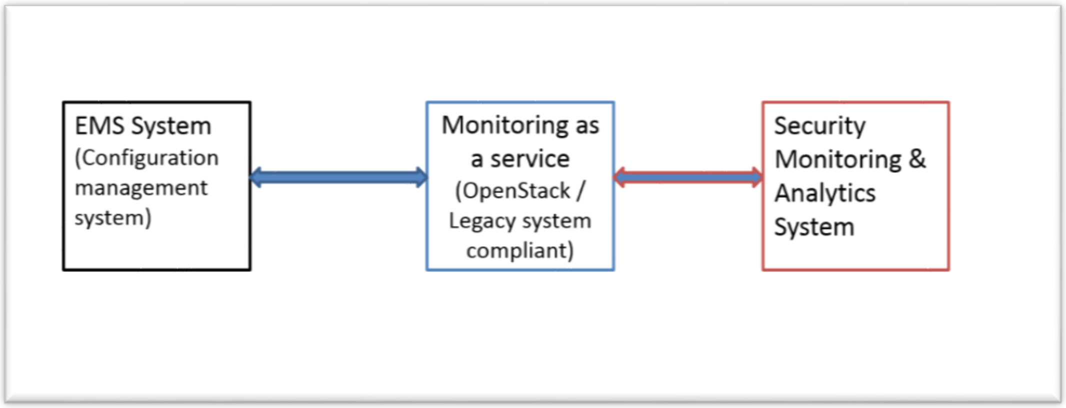

EMS System (Configuration management system)
Monitoring as a service (OpenStack / Legacy system compliant)
Security Monitoring & Analytics System

Figure 2. Top Level Block View of MaaS

Once traffic to be monitored is received by MaaS, it analyzes (based on various application /protocol supported) and forwards the traffic to Security monitoring and analytics systems where in various 3rd party tools shall be running.

.

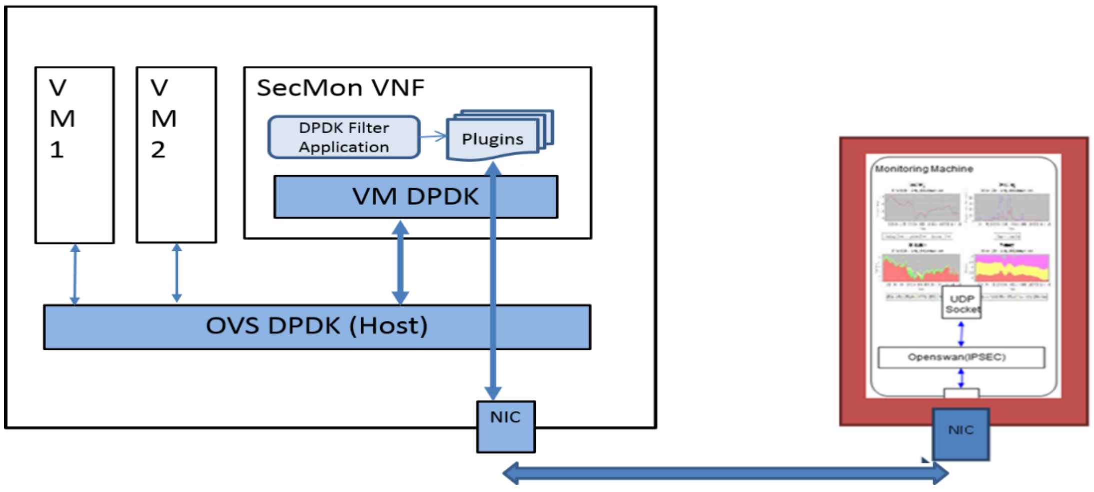


Figure 3. Top Level Block View of SecMon with OVS switch

When performance is the key consideration and SecMon is used in VM then, optionally DPDK enabled OVS can be used. OVS-DPDK used in Host machines/Computes, bypasses the host kernel and directly feeds the traffic to SecMon VM.

Following pluggable Filtering and Monitoring Services are provided by Monitoring as a Service (SecMon EMS)-

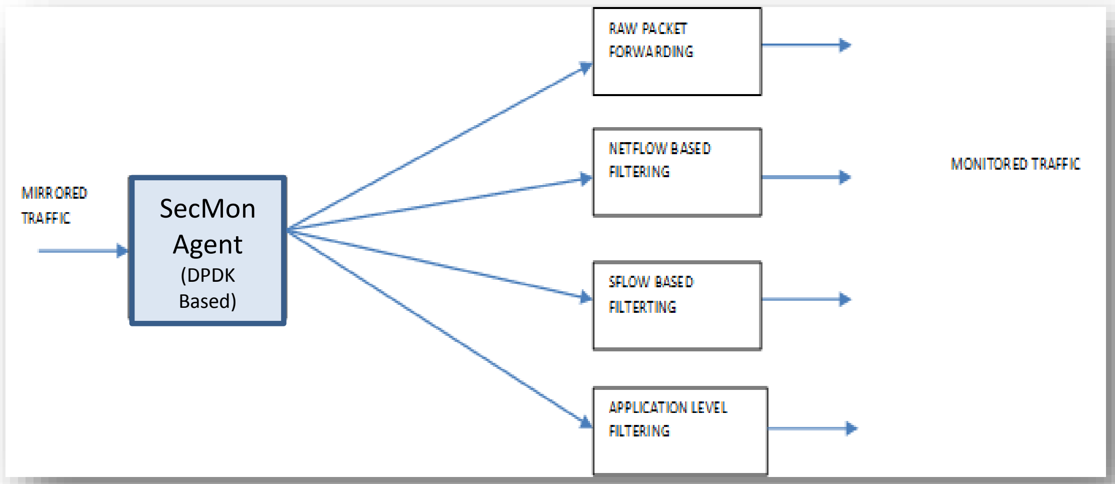


Figure 4. Components and functions of reference solution

Note: Support for Sflow is provided in SecMon EMS only.

3.1.1.1 **7T Filter**

Every plugin has their 7T Filter classification rules configured. These rules filter and forward the traffic based on configured 7 tuple filtering rules. The filtered traffic is passed to the core plugin module for monitoring and sample purposes.
Filtering based on some tuple saves the network bandwidth as some of the data traffic gets filtered out before going onto the network to reach on Monitoring Tool's Machine. Also this provides the flexibility to select the traffic for further processing on the basis of 7Tuples. Following are the 7 tuples used for:

1. *Source IP Address*
2. *Destination IP Address*
3. *Protocol*
4. *Source L4 Port*
5. *Destination L4 Port*
6. *Source Mac Address*
7. *Destination Mac Address*

If this option is not configured, all the traffic gets forwarded to subsequent plugins as explained below.

3.1.1.2 **Raw-Forward (UDP/Sflow Header Encapsulation)**

This is a forwarding option; it forwards all traffic that comes from 7Tuple filtering to the monitoring tools machine. Raw Forwarding does not perform any sampling/monitoring to the incoming traffic. It encapsulates the traffic in UDP and Sflow header; secure it using IPsec and forward towards the configured monitoring tools for traffic analysis.

The machine running monitoring tool can exist either in OpenStack cloud or outside the cloud.
This traffic can be monitored using packet capture tools viz., Wireshark, Snort etc.

3.1.1.3 **NetFlow**

NetFlow is a feature that provides the ability to collect IP network traffic as it enters or exits an interface. By analyzing the data provided by NetFlow, a network administrator can determine things such as the source and destination of traffic, class of service, and the causes of congestion.

SecMon VNF supports the NetFlow version 9 as one of the traffic monitoring and filtering feature. It works as flows are exported and creates the flow records for the traffic that is mirrored towards the SecMon VM. Here the traffic to be monitored is the tenant's VM traffic which is mirrored towards the SecMon VM. The created flow records are sent to the NetFlow Collector machine for further analysis.

Any NetFlow v9 compliant collector tool can be used to collect and analyze the exported NetFlow v9 flow records. Multiple collectors can be configured to get the exported flow records.

This traffic can be monitored using packet capture tools viz., Wireshark, NetFlow collector.

3.1.1.4 **Load Balancing**

Load Balancing stand for the ability to balance the traffic across two or more entities. MaaS provides load balancing by distributing the sessions to be monitored between two or more collectors. Here a session is defined as client-server communication whose traffic is to be monitored. The set of the collectors together are called a **Collectorset**. CollectorSet is defined per plugin i.e. NetFlow CollectorSet, SFlow CollectorSet and RawForward CollectorSet. The distribution of the session is defined by the load balancing algorithm.

MaaS provides three different algorithms to choose from:

- Session Based
- Round-Robin Based
- Weighted Round-Robin Based

Weighted Round-Robin: - In weighted round-robin algorithm, the sessions are distributed between the collectors based on the ratio of the weight provided per collector. Collectors with higher weight will receive higher percentage of traffic.

Round-robin Based: - In Round-Robin based algorithm, the sessions are distributed between the collectors in circular "Next-in-loop" manner i.e. the session will be assigned to the next collector in the loop.

Session Based: - In session-based algorithm, the sessions are distributed among the collectors such that the session is assigned to the "Next-available" collector from the collectorset which has minimum number of sessions at present.

3.1.1.5 **EMS System**

EMS stands for element management system, which is a centralized management and configuration system. It comprises of EMS server, which maintains the details of connecting VNFs and configurations of tools. When VNF contacts EMS server to fetch configurations, EMS provide the configurations to VNF. These communications happen through REST interface.

EMS server also support user roles. Each user can have separate rules specified by admin in EMS server. User can read, update and delete only those configurations of which it has permissions.

Both CLI and GUI based configuration management tools are supported. Configurations are maintained using consul DB.

3.1.2 **IPsec function**

IPsec function within the VM can be implemented in various ways to create various reference solutions:

- To work in various cloud environments
  - Workload embedded with IPsec in public clouds where operators are unwilling to enhance their VMM.

    - Fixed IPsec function virtual appliance in public cloud where operators are unwilling to enhance their VMM.
    - Workload embedded with IPsec in private clouds or public clouds where operators are willing to enhance their VMM.
    - Fixed IPsec function virtual appliance in private clouds or public clouds where operators are willing to enhance their VMM.

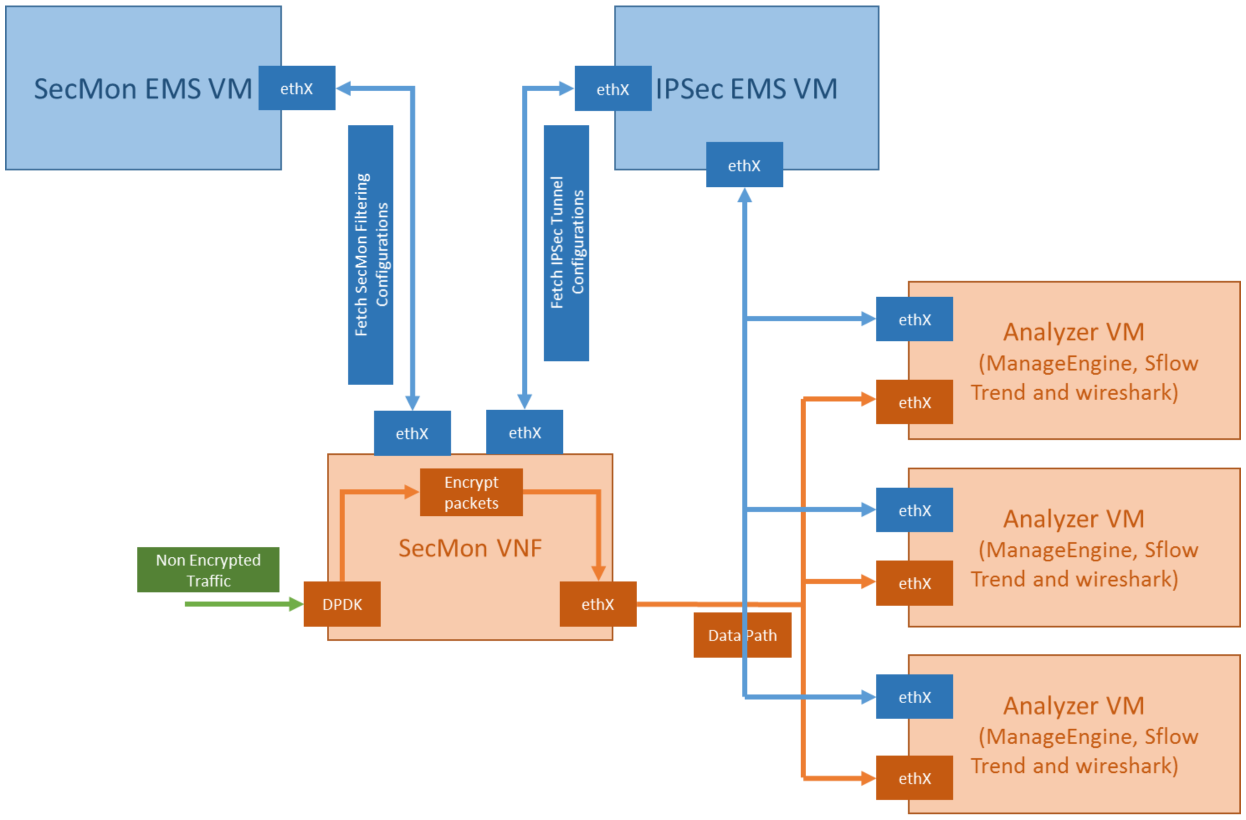
IPSec subsystem provides easy configuration between Single/Group SecMon VNF and Analyzer(s). Apart from SecMon VNF, it can also be used for standalone IPSec configuration between group of nodes.

The specifications of our IPsec functions as follows:

- Tunnel Mode
- ESP
- DSCP based IPsec tunnels
- IKEv1 and IKEv2
- Pre shared key
- Algorithms
    - Encryption : AES-CBC (128, 256 bits)
    - Authentication : SHA-1, SHA-2
    - Encryption & Authentication : AES-GCM

3.2 Monitoring as a Service - Architecture

- Monitoring as a Service is implemented as a distributed architecture. Configurations and filtering are managed in different node. It has below major components. Internal details of each of the components are described in subsequent sections.



SecMon EMS VM
ethX
Fetch SecMon Filtering Configurations
Fetch IPSec Tunnel Configurations
ethX
IPSec EMS VM
ethX
Analyzer VM (ManageEngine, Sflow Trend and wireshark)
ethX
ethX
ethX
Encrypt packets
SecMon VNF
Non Encrypted Traffic
DPDK
ethX
Data Path
Analyzer VM (ManageEngine, Sflow Trend and wireshark)
Analyzer VM (ManageEngine, Sflow Trend and wireshark)

<u>Secure Monitoring Architecture with IPSec Secure Tunneling</u>

Figure 5. SecMon VNF Internal Architecture with EMS

SecMon VNF (SecMon VNF)

- The SecMon VNF is responsible for all the monitoring/filtering services. This EMS instantiates and configures for tenants traffic monitoring. Further the traffic monitoring can be classified based on the scope with in the tenant network.
- Each SecMon VNF can be optionally implemented as VM in an OpenStack environment / non OpenStack environment and shall be running DPDK based plugins for different monitoring protocols.
- SecMon VNF supports dynamic plugin architecture, which means, at any point in time, supported plugins can be added /taken into consideration in a seamless manner. As per the current release, it supports monitoring protocols mentioned above. However, it is possible to

attach any third party plugin for with this DPDK based packet filter application to introduce other monitoring protocols.

- SecMon VNF plugin once added, interacts with SecMon EMS system to fetch configurations. Plugins configurations are updated and deleted from Graphical User Interface of SecMon EMS, which interact with SecMon EMS server backend.
- SecMon VNF can also contain IPSec Enforcer to fetch IPSec tunnel configurations from IPSec EMS server. IPSec EMS server configurations are updated and deleted from Graphical User Interface of IPSec EMS, which interact with IPSec EMS backend.

SecMon EMS

- SecMon EMS is central entity responsible for configuring all the SecMon VNF for specific tenant network and maintains the database for each SecMon VNF
- SecMon EMS can be executed inside the VM.
- It exposes the REST APIs to receive the SecMon EMS configurations from configuration management interface i.e., CLI / GUI and maintains them in the database.
- It interacts with SecMon VNF over the REST based interface.
- SecMon EMS uses single instance consul DB (bootstrap mode) database for storing Plugins configurations.

3.2.1 SecMon VNF Internal Design

This section describes the Component Level Design of SecMon. The functional blocks are:

SecMon Agent: It is a DPDK based application which runs in user space.

Monitoring Plugins: It is a filtering component which filters packets according to plugin rules mentioned in SecMon EMS.

Architecture is shown below:

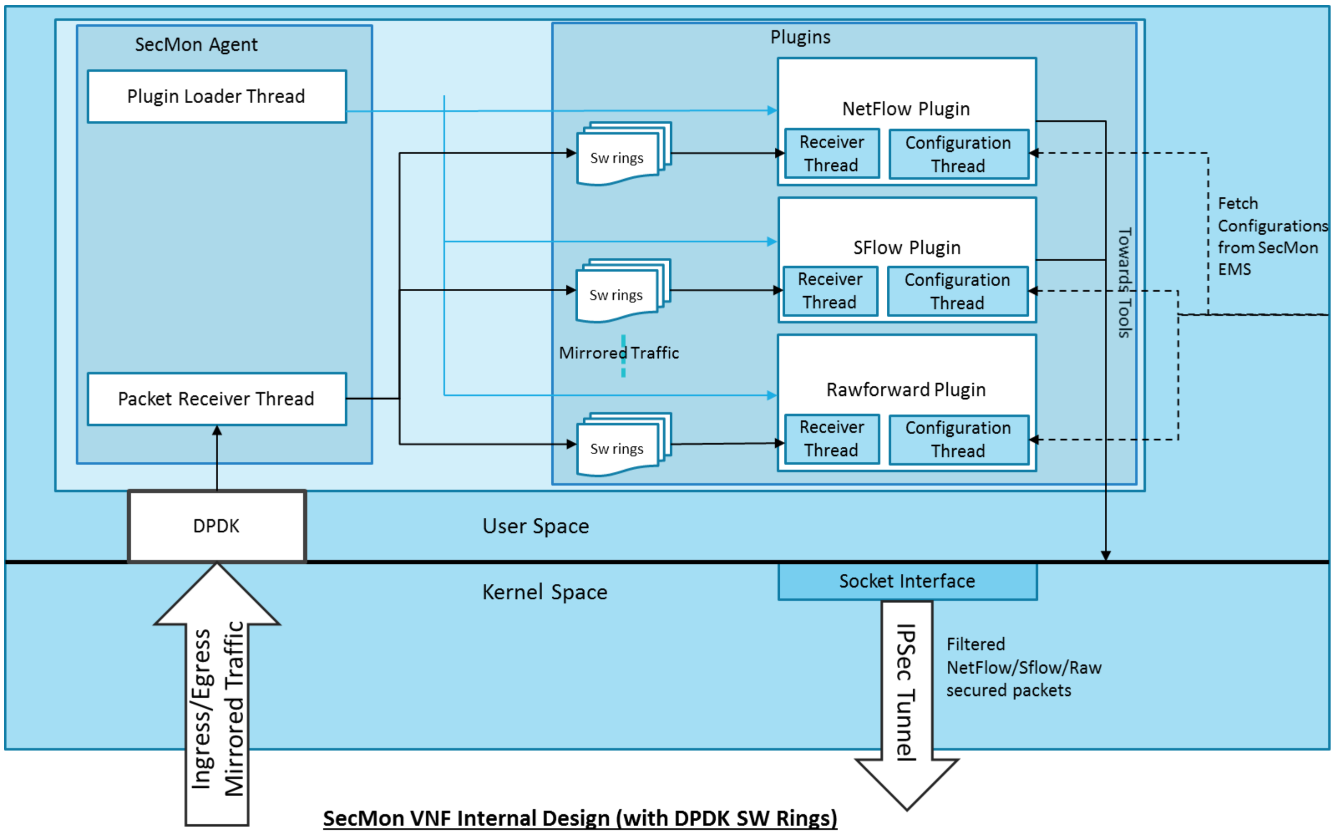


Figure 6. SecMon VNF Internal Design (with DPDK SW Rings)

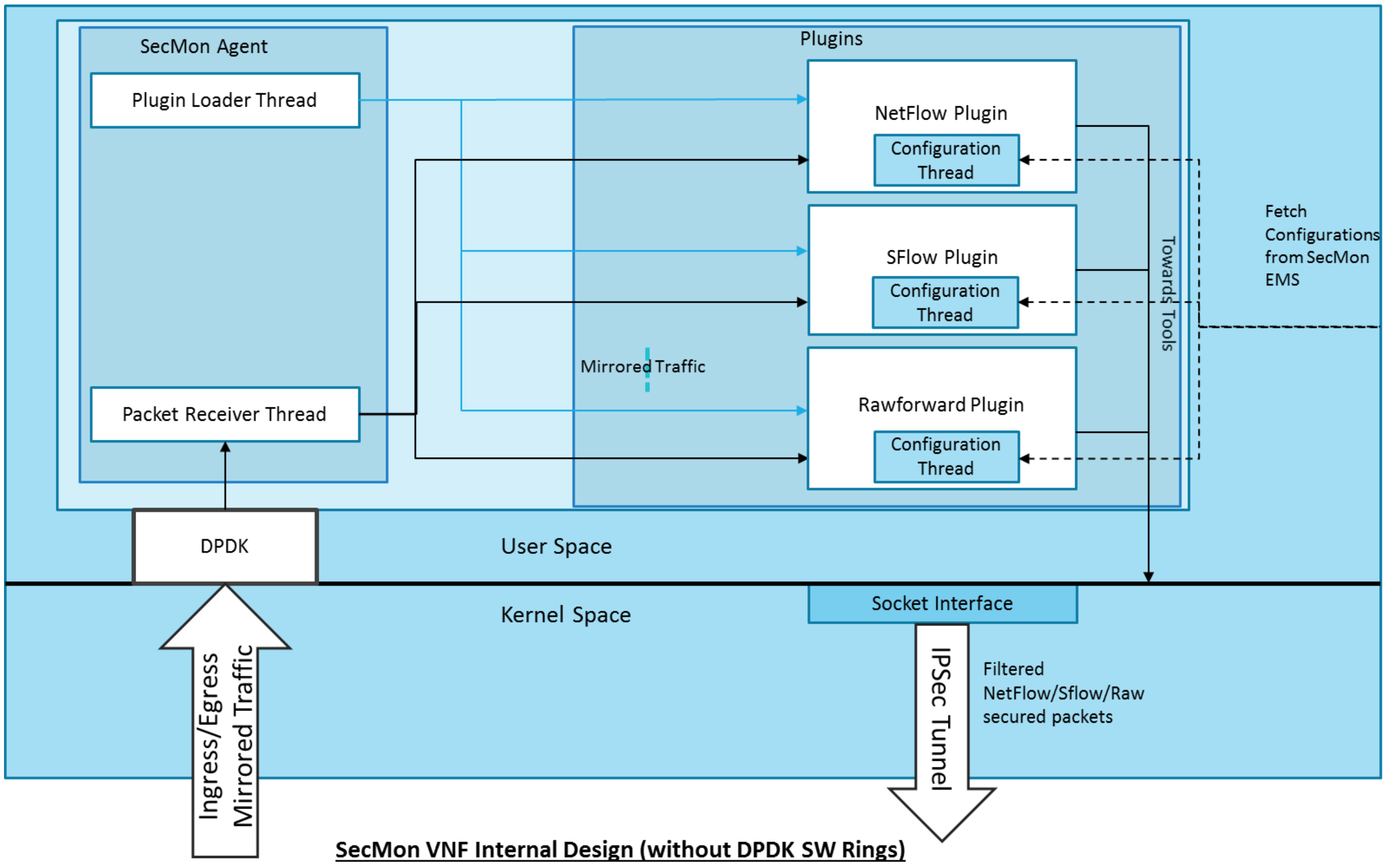


Figure 7. SecMon VNF Internal Design (without DPDK SW Rings)

3.2.1.1 SecMon Agent

It is a DPDK based application which loads filtering plugins and receive packets from network interface through DPDK library. Received packets then are provided to all the plugins which are registered.

3.2.1.1.1 SecMon Agent design

DPDK Packet Filter Application runs variable number of filtering services described above. It provides the framework for all the filtering services to be able to dynamically load and run within the DPDK based packet filter application.

This is a multithreaded DPDK application with threads running on different cores with CPU core affinity set.

Two logical cores are assigned to Packet Filter DPDK Application at startup. Following is the threaded breakup for Packet Filter Application:-

- Plugin Loader Thread: DPDK Packet Filter Application main thread is viewed as a plugin loader thread. It loads all the plugins provided as dynamic library at startup. Also monitors for any run time plugin addition. Plugins are loaded in the form of threads inside DPDK packet filter application.

Every monitoring plugin is built using SecMon plugin compliant framework. This means, in order to dynamically load and run a plugin, it is expected to comply to interface specification APIs.

- Packet Receiver Thread: - This thread runs on single core and receives all the mirrored traffic at SecMon VM.
  DPDK poll mode driver is binded to Linux Kernel Ethernet interface. This driver is polling Linux kernel Ethernet interface for packets. In SecMon Agent we read 32 packets at a time due to performance penalty in reading single packet at a time.
  After reading 32 packets from interface we pass individual packet to each plugin by calling interface of plugins (i.e receive_from_secmon).
  If DPDK software rings are used, then all the packets are enqueued in software ring by *receive_from_secmon* api. And separate thread in plugin will dequeue packets from this software rings.
  If DPDK software rings are not used, then all the packets are directly provided to filtering component of plugin by *receive_from_secmon* api. So one thread is removed from plugin which was previously dequeueing packets from software rings.

3.2.1.1.2 Interface between SecMon Agent and DPDK library

SecMon uses Intel's DPDK library for reading packets from Ethernet interfaces. DPDK library provide set of interfaces through which applications like SecMon can use to read packets directly from Ethernet interface without following traditional Linux TCP/IP stack. As packets does not follow whole TCP/IP stack, packets receiving capability is increased.

SecMon application uses these DPDK interfaces to read packets. Some of the interfaces are explained below:

- **rte_mempool_create**
  This is used to create memory pool in which DPDK store each packets it read from NIC. The size of each element is (2048 + sizeof(struct rte_mbuf) + RTE_PKTMBUF_HEADROOM) bytes and number of these elements are 8192. The reference of these packets then transferred to plugins. After usage by each plugins memory is returned to memory pool.
- **rte_eth_dev_configure**
  This is used to configure Ethernet device. This function take **portid** (device to configure), **nb_rx_queue** (number of rx queues), **nb_tx_queue** (number of tx queues) and **eth_conf** (configuration options). SecMon uses single rx and tx queues.
- **rte_eal_remote_launch**
  This is used to launch function in another lcore. SecMon launches **packet_receiver_lcore** function on another lcore. When execution of launched function completes it stores return value in local variable which can be checked from **rte_eal_wait_lcore.**

3.2.1.2 **SecMon Plugin**

3.2.1.2.1 **Plugin Design**

Each plugin is implemented as a dynamic loadable library which is loaded by DPDK packet filter application. Plugin architecture can be configured to use software rings or not for packets buffering according to requirement.

When plugin does not use software rings:
Plugins receive the data packets from DPDK interface directly (no intermediate software rings for buffering).
Each plugin has following sub-modules:

1. Plugin Core: The actual monitoring protocol is implemented here. This is solely protocol dependent. After the sampling/monitoring is done based on the protocol that the plugin is written for, the sampled/monitored packets are sent to the tools machine.
   Tools configuration is done via SecMon EMS GUI.

2. Filter Module: The classification filters respective to each plugins. The filter module contains the classification filter rules to filter the incoming traffic. The filtered traffic is passed to the plugin core module for actual processing. These filter rules are configured via SecMon EMS GUI.

3. Plugin Client: Each plugin has its internal REST based client which connect to its SecMon EMS server to fetch the initial configurations. Fetched configurations are used by plugin core for forwarding packets to Analyzer Tools and by filter module for filtering mirrored traffic.

4. Plugin Server: Plugin server is used to receive the notification and fetch updated configuration from SecMon EMS Server. Each plugin has its own REST based server to expose the APIs. These APIs are invoked by SecMon EMS Server whenever there is an update in the plugin's configurations.

When plugin use software rings:
Plugins receive the data packets from DPDK based software rings (the packets are enqueue in the ring by Packet Receiver thread) and dequeue by the plugin's receiver thread. Plugin have same submodules as when plugin don't use software rings for packets buffering. Receiver thread is extra component in this case which will dequeue packets from software rings.

3.2.1.2.2 **Interface expected by SecMon in plugins**

SecMon expects some interface functions to be present in plugins. The interfaces are listed below with their significance:

- **int init()**

This is the first function called by SecMon Agent. This function can be used for Memory initialization, thread initialization, opening log files, configuration files etc .
Params: void
Returns: 0 for success and other values will be considered as failure and other functions will not be called.

- **int deinit()**
  This is the exit function for the plugin. Cleanup tasks should be performed in this function like memory freeing, file closing, thread killing etc.
  Params: void
  Returns: 0 for success and negative value for failure.
- **int config()**
  - Native configuration
    If vendor has its own configuration system to configure the plugin. This function can be skipped just by creating an empty function.
  - SecMon Agent configuration system
    Plugin will fetch the configurations from the SecMon EMS via Rest API. This API should fetch all the available configurations from SecMon EMS. It should also have the ability to receive the configurations whenever there is any change/update in the configurations. This function should return as soon as possible.

  Params: void
  Returns: 0 for success and other values will be considered failure.
- **Int receive_data()**
  This function is invoked from SecMon Agent. This routine create three new threads. One for dequeueing packets from DPDK software rings, one for removing old entries from data structures after some time and one for configurations fetching from SecMon EMS.
  Params: void.
  Returns: 0 for success and negative for failure.
- **Int receive_from_secmon(struct rte_mbuf)**
  This function is to receive the packet's rte_mbuf direct pointer. It's up to the plugin as to how to process the packet, but it should return as soon as possible. Copying the packet's rte_mbuf pointer to a dpdk ring is preferred. SecMon Agent will do nothing in case of success or failure of this function.
  However if no DPDK ring is present then packet is directly provided to plugin filtering part without buffering in between.
  Params: Pointer to memory in which packet is stored.
  Returns: 0 for success and negative for failure.

3.2.1.2.3 Fetching plugins configurations from SecMon EMS

SecMon Agent loads plugins from /opt/secmon/plugins. Plugins have general interfaces which SecMon agent expects. Inside plugins we have *receive_data* interface which is called by SecMon agent after loading the plugins. Inside this routine three threads are created (if we are using DPDK software rings). One thread fetch packets from DPDK rings, second fetch configurations from SecMon EMS server and third check if some configurations are expired, if so then remove them.

Configuration thread fetch configurations from SecMon EMS server. Initial plugins configurations which are already present in SecMon EMS are first fetched and stored in data structures of respective plugins. After initial configurations are fetched and saved, plugins launch endless running server which waits for notification from EMS notifier for configurations changes in SecMon EMS. When configurations changes in SecMon EMS, plugin server also delete previous configurations and load new ones after flush operation in SecMon EMS.

The data structure to store configurations in plugins are linked lists. There is separate global linked lists for policies, associations, rules, classification objects, collectors and etc.

Configuration flow of NetFlow plugin is shared below. Flow for SFlow and Rawforward plugin is similar as NetFlow.

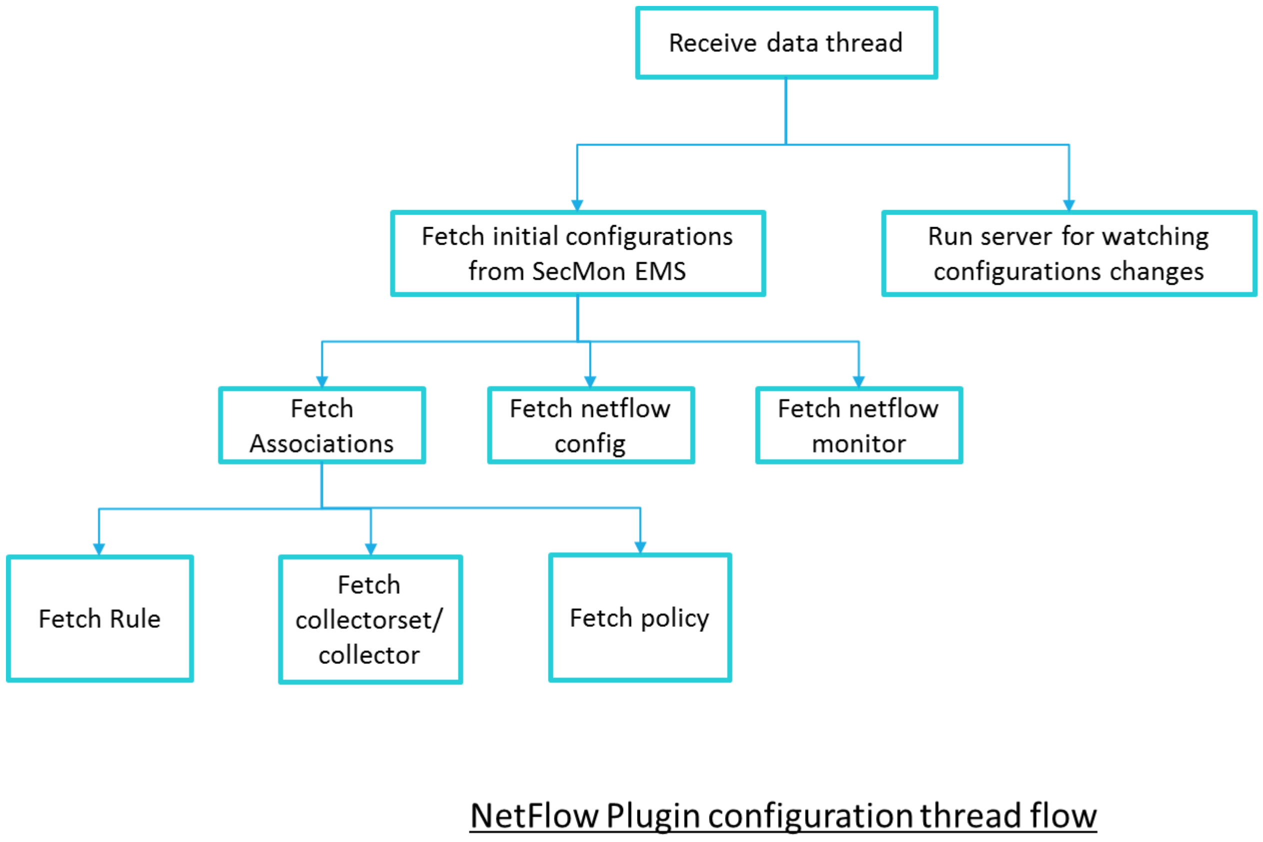

Receive data thread
Fetch initial configurations from SecMon EMS
Run server for watching configurations changes
Fetch Associations
Fetch netflow config
Fetch netflow monitor
Fetch Rule
Fetch collectorset/ collector
Fetch policy
NetFlow Plugin configuration thread flow

Figure 8. NetFlow configuration thread flow

All the SecMon plugin REST server APIs are listed below in document:

SecMon_Rest_API_Details _v1.2.docx

3.2.2 SecMon VNF Initialization Procedure

SecMon VNF can be instantiated as an OpenStack VM Instance from Horizon GUI. In order to provide SecMon VNF functionalities as mentioned in above sections, it has to move in Active-State after instantiation from GUI.

For SecMon VM, Active State is when it is successfully launched and is waiting for following tasks-

1. Receiving packets from DPDK application for mirrored port.
2. Listening to receive the configurations from EMS Controller.

Startup configurations are required inside the SecMon VM. These configurations are done at VM booting time via initialization script provided in init.d folder of SecMon VM disk image.

1. Allocate Hugepages.
2. Bind SecMon VM Port (which receives the mirrored traffic) to DPDK driver.
3. Establish IPsec Tunnel with SMA system from SecMon VM.
4. Start DPDK Packet Filter Application which in turn start plugin loader and packet receiver EAL threads.

3.2.3 SecMon EMS VM (Architecture & Design)

SecMon EMS Server can be optionally implemented inside OpenStack VM. This is python based server implemented using Django framework. It interfaces with EMS policy enforcer i.e. Horizon GUI / CLI and SecMon EMS VM over REST Based Interface.

The following diagram depicts architecture view of SecMon EMS with its components:

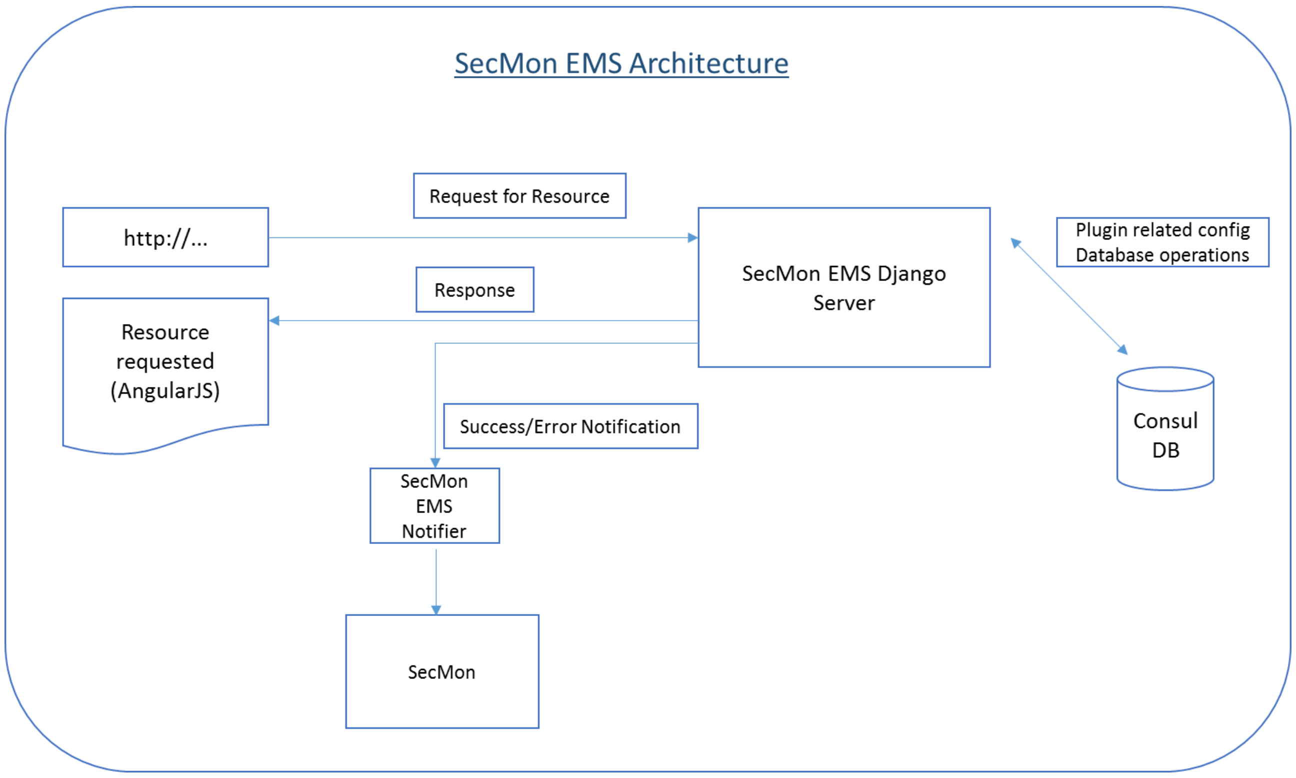


Figure 9. SecMon EMS Architecture

3.2.3.1 Components of SecMon EMS

- Django Server
  Server handle requests done by client and respond with proper response. Server also inform EMS notifier about the changes which in turn inform SecMon.
  All the Plugins configurations data is stored in consul DB database which Server query and update according to client request.
- Consul DB Database
  This is distributed database in which data is stored as key/value pairs. We have single database server architecture. So all the data is stored in single consul DB database.
- SecMon EMS Notifier
  Whenever user changes the plugins configurations in Django Server. Django server notify the SecMon plugins of configurations changes through EMS notifier.

3.2.3.2 SecMon EMS data models

It maintains the following SecMon EMS data models for every SecMon EMS and its plugins in Consul DB and depicted as below.

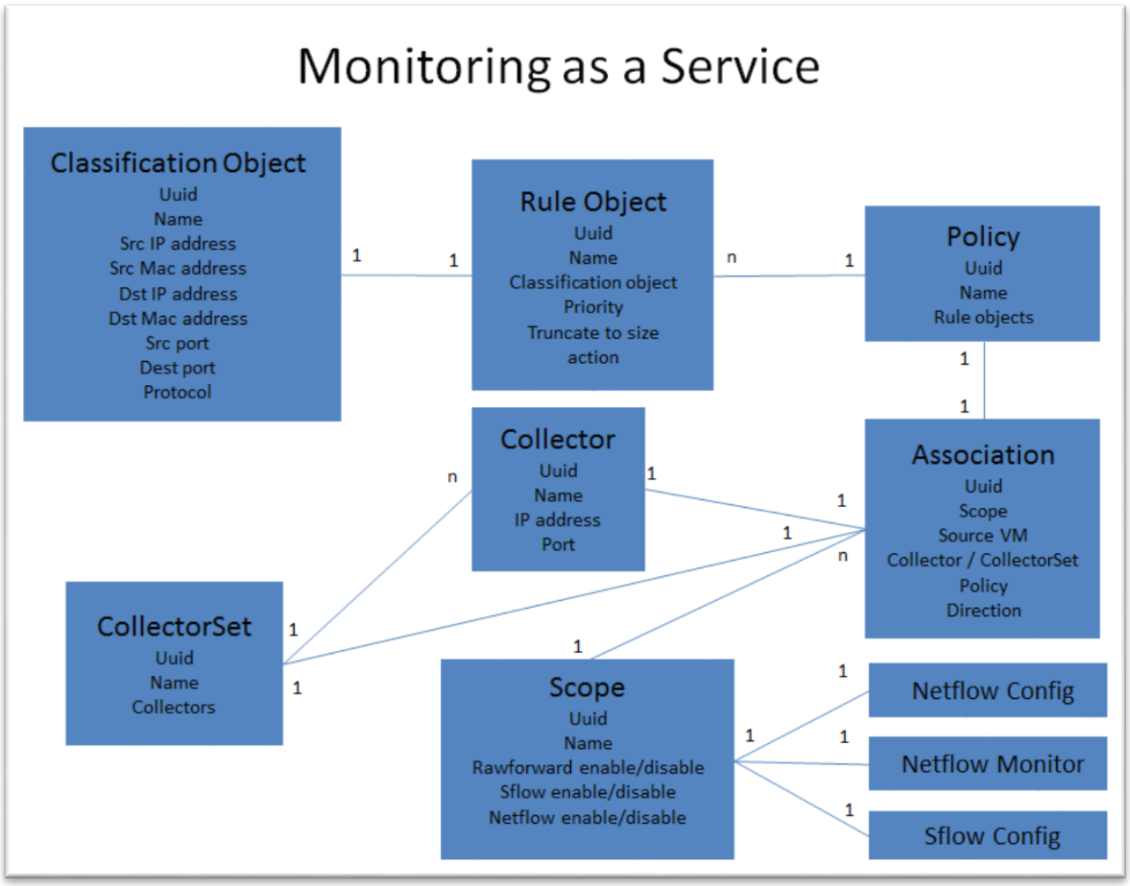


Figure 10. EMS Controller Data Model for SecMon EMS

3.2.3.3 SecMon EMS Admin Interface

SecMon EMS web interface is divided into two parts. One interface is available for admin user and other interface is available for normal user with restrictive permissions.

SecMon EMS Admin interface mainly can do three things create users, provide permissions to users and create plugins configurations. With RBAC integrated in SecMon EMS we can create many different user profiles and give them different access rights.

All SecMon EMS server APIs are listed in SecMon_EMS_Rest_Documentation.pdf:

3.3 IPSec Function – Architecture

3.3.1 Overview

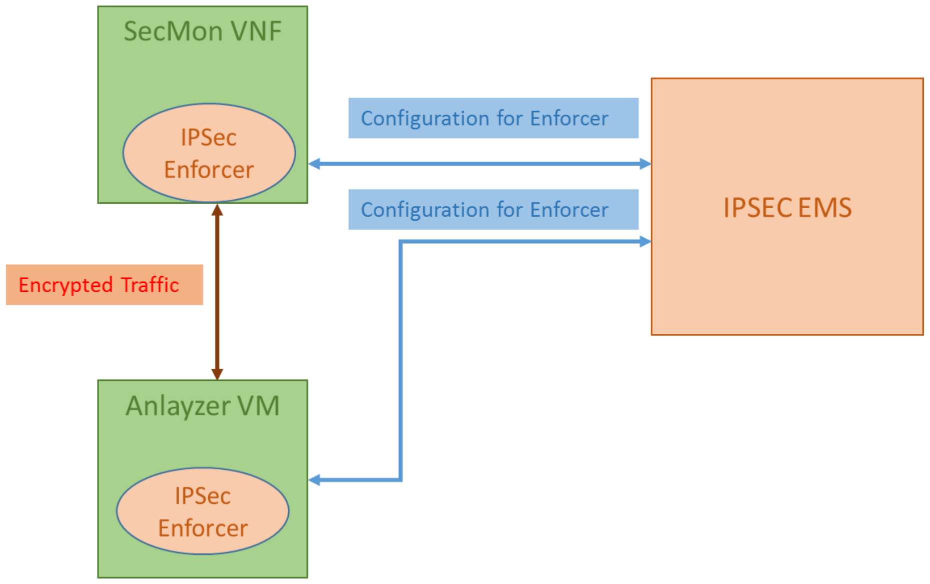
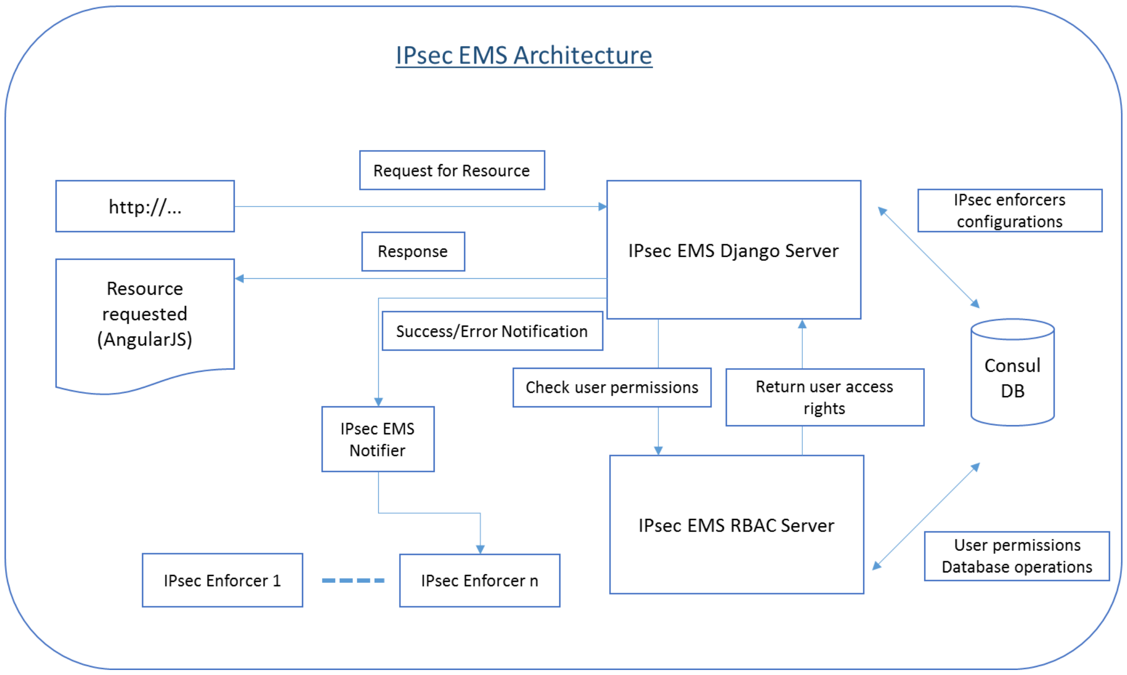


Figure 11. IPSec related components

Note: Section on IPSec may not be completely inline with the current code

IPSec EMS

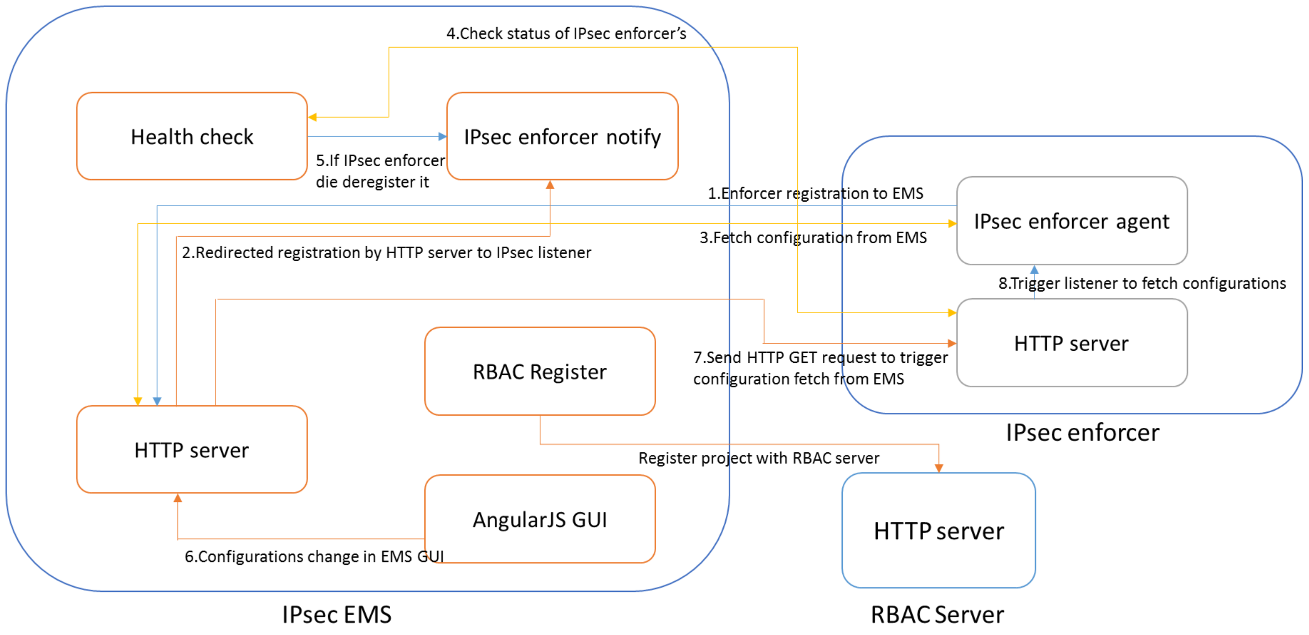
It provides User Interface for the easy configuration of IPSec and exposes REST APIs towards Enforcers. Architecture for IPSec is very similar to the SecMon EMS except for IPSec EMS RBAC Server, which is used to manage the permissions for each user.



IPSec Enforcers

Enforcers fetches IPsec configurations from IPsec EMS using RESTful API's exposed by IPsec EMS. These enforcers use the configuration for setting up IPSec between SecMon VNF (s) and Analyser(s). ipsec.secrets

3.3.2 IPSec EMS and Enforcer – Component Details


4.Check status of IPsec enforcer's
Health check
IPsec enforcer notify
5.If IPsec enforcer die deregister it
1.Enforcer registration to EMS
IPsec enforcer agent
3.Fetch configuration from EMS
2.Redirected registration by HTTP server to IPsec listener
8.Trigger listener to fetch configurations
HTTP server
RBAC Register
7.Send HTTP GET request to trigger configuration fetch from EMS
IPsec enforcer
HTTP server
Register project with RBAC server
AngularJS GUI
HTTP server
6.Configurations change in EMS GUI
IPsec EMS
RBAC Server

IPsec EMS functionality

- **Health check service** send HTTP GET requests to all the IPsec enforcers which are registered and they respond OK if they are alive otherwise no response. If for some back off period IPsec enforcer doesn't respond then that IPsec enforcer assumed to be dead and it's entry from database are removed.
- **IPsec enforcer notify** starts a listener socket on startup. Then waits for REGISTER, DEREGISTER, UPDATE and DELETE events.
- **RBAC register service** registers project with local RBAC service.
- **HTTP server** has two components. First, GUI which present user with GUI to modify configurations of IPsec tunnels. Second, REST API's which provide RESTful endpoints for resources.
- When IPsec enforcer agents startup, they first send HTTP POST requests to IPsec EMS server to register themselves.
- Then after registering they fetches IPsec tunnel configurations from IPsec EMS.

IPsec Enforcer functionality

- **IPsec enforcer agent** on startup register itself with IPsec EMS and fetches configurations from IPsec EMS also and then wait for event.
- **HTTP server** running using Django. When configurations changes in IPsec EMS it calls particular URL in this server. Which then wake listener who fetches updated records from IPsec EMS.

4. References

Table 2 Referenced documents

Document Name	Document Number and/or URL
OpenStack	https://www.OpenStack.org
Intel DPDK	http://dpdk.org/
NetFlow	https://www.ietf.org/rfc/rfc3954.txt
SFlow	http://www.sflow.org/

Acronyms

Instructions: Provide a list of acronyms and associated literal translations used within the document. List the acronyms in alphabetical order using a tabular format as depicted below.

Acronym	Literal Translation
VNB	Virtual Network Broker
IPSec	Internet Protocol Security
VNF	Virtual Network Function
REST	Representational State Transfer
DPDK	Data Plane Development Kit
VM	Virtual Machine
KVM	Kernel Virtual Machine
SSC	Security Service Controller
NFVI	Network Function Virtualization Infrastructure
VNFM	Virtual Network Function Manager

Table: Acronyms