I2NSF Data Model of Consumer-Facing Interface for Security Management

(draft-jeong-i2nsf-consumer-facing-interface-dm-00)

Jaehoon Paul Jeong, Mahdi Daghmehchi (Presenter), Tae-Jin Ahn, Rakesh Kumar, and Susan Hares

IETF 97, Seoul, Korea
November, 2016
Contents

I Introduction
II Motivation
III Architecture of Security Management
IV Use Case: VoIP-VoLTE Security Service
V Next Step
Introduction

• This document describes a data model for security management based on I2NSF framework by using NFV

• A data model to perform VoIP-VoLTE security service
Motivation

- Defining high level policies and translate them to several low level policies
- Updating low level policies based on NSF capabilities
- Monitoring network’s events and implementing security functions based on NFV
- Data modeling for I2NSF Customer-Facing Interface
Security Management Architecture (1/3)

- **Application Logic**
  Generating high level security policies

- **Event Controller**
  Event monitoring and sending to Application logic

- **Policy Updater**
  Distributing high level policies to the Security Controller

---

The diagram illustrates the interaction between the I2NSF User, Security Management System, and NSF Instance. The key components include:

- **Application Logic**: Generates high level security policies.
- **Event Controller**: Monitors events and sends them to the Application Logic.
- **Policy Updater**: Distributes high level policies to the Security Controller.

The Security Management System includes the **Security Controller** with components such as Security Policy Manager and NSF Capability Manager, as well as the **Registration Interface** and **Developer’s Mgmt System**.

The NSF Instance is connected via NSF Facing Interface, representing NSF instances and management system interactions.
Security Management Architecture (2/3)

- **Security Policy Manager**
  - Mapping high level policies into several low level policies
  - Delivering low level policies to NSF(s)

- **NSF capability manager**
  Storing the NSF’s capability and sharing it with Security policy manager

- **Developer’s Mgmt system**
  Registering new NSF’s capabilities into NSF capability manager
Security Management Architecture (3/3)

- NSF Instance
  Exploiting low level policies delivered by the Security policy manager
Use Case: Security Management for VoIP-VoLTE Service

VoIP-VoLTE Security Management: Application Logic

- Defining security conditions (e.g., blacklists of IP addresses & source ports, expire time, user agents)
- Updating the illegal devices information (manually/automatically)
- Generating new high-level security policies
- Updating the VoIP-VoLTE database based on the NSF’s anomalous detection

Information Model for Consumer-Facing Interface

Information Model for:

- Threat Prevention
  To reduce the attack surface (e.g., Botnet)
- Policy endpoint groups
  Where a security policy is to be applied
- Policy Instance
  A complete information for any policy instance (e.g., where/when a policy need to be applied)
High level policies basements:

- Blacklisting countries
- Time interval specification
- Caller’s priority levels

The data model consists of:

- Policy life cycle management
- Policy rule
- Action
Policy life cycle management
Specifies an expiration time and/or event to determine the life-time of the policy itself

Policy rule
Represents the specific information about a high-level policy e.g., service types, conditions and valid time interval

Action
Specifies the actions which should be performed when a policy rule is matched by NSF
YANG Data Model for VoIP-VoLTE Security Service

```yang
//Groupings
grouping policy {
  list policy-lifecycle {
    key "policy-lifecycle-id";
    description "The ID of the policy lifecycle for each policy. This must be unique.";

    leaf policy-lifecycle-id {
      type uint16;
      mandatory true;
      description "This is policy lifecycle-id.";
    }
  }

  container expiration-event {
    description "The event which makes the policy expired.";

    leaf enabled {
      type boolean;
      mandatory true;
      description "This represents whether the policy is enabled or disabled.";
    }

    leaf event-id {
      type uint16;
      mandatory true;
      description "The ID of the event. This must be unique.";
    }
  }

  list condition {
    key "condition-id";
    description "The ID of the condition. This must be unique.";

    leaf condition-id {
      type uint16;
      mandatory true;
      description
    }
  }
}
```

container service {
  description "The services which NSFs could perform to manage the security attacks. This consists of voip-handling and volte-handling. This will be extended in later version.";

  leaf voip-handling {
    type boolean;
    mandatory true;
    description "This field represents whether the policy contains handling the voip packet flow.";
  }

  leaf volte-handling {
    type boolean;
    mandatory true;
    description "This field represents whether the policy contains handling the volte packet flow.";
  }
}
Next Step

• **Testing our YANG data model**
  Testing our YANG data model in our I2NSF Framework Code (used in IETF-97 I2NSF Hackathon)

• **Implementing more use cases**
  e.g., untrusted domain (malware distributor) detecting and access control function (time/location depended)