I2NSF Working Group
Internet-Draft
Intended status: Informational
Expires: May 1, 2017

R. Kumar
A. Lohiya
Juniper Networks
D. Qi
Bloomberg
N. Bitar
S. Palislamovic
Nokia
L. Xia
Huawei

October 28, 2016

Requirements for Client-Facing Interface to Security Controller
draft-kumar-i2nsf-client-facing-interface-req-02

Abstract

This document captures the requirements for the client-facing interface to the security controller. The interfaces are based on user constructs understood by a security admin instead of a vendor or a device specific mechanism requiring deep knowledge of individual products and features. This document identifies the requirements needed to enforce the user-construct oriented policies onto network security functions (NSFs) irrespective of how those functions are realized. The function may be physical or virtual in nature and may be implemented in networking or dedicated appliances.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Programming security policies in a network has been a fairly complex task that often requires very deep knowledge of vendor specific devices. This has been the biggest challenge for both service providers and enterprises, henceforth named as security administrator in this document. The challenge is amplified due to virtualization with security appliances in physical and virtual form factor from a wide variety of vendors; each vendor have their own proprietary interfaces to express security policies on their devices.

Even if a security administrator deploys a single vendor solution with one or more security appliances across its entire network, it is still difficult to manage security policies due to complexity of security features, and difficulty in mapping business requirement to vendor specific configuration. The security administrator may use either vendor provided CLIs or management system with some abstraction to help provision and manage security policies. But, the single vendor approach is highly restrictive in today's network for the following reasons:

o The security administrator cannot rely on a single vendor because one vendor may not be able to keep up with their security requirements or specific deployment model.

o A large organization may have a presence across different sites and regions; which means, it may not be possible to deploy same solution from single vendor due to regulatory requirement or organizational policy.

o If and when security administrator migrates from one vendor to another, it is almost impossible to migrate security policies from one vendor solution to another without complex manual workflows.

o Security administrators deploy various security functions in virtual or physical forms to attain the flexibility, elasticity, performance, and operational efficiency they require. Practically, that often requires different sources (vendor, open source) to get the best of breed for any such security function.

o The security administrator might choose various devices or network services (such as routers, switches, firewall devices, and overlay-networks) as enforcement points for security policies.

This my be for reason (such as network design simplicity, cost, most-effective place, scale and performance).

In order to ease the deployment of security policies across different vendors and devices, the Interface to Network Security Functions (I2NSF) working group in the IETF is defining a client-facing interface from the security controller to clients [I-D. ietf-i2nsf-framework] [I-D. ietf-i2nsf-terminology]. Deployment facilitation should be agnostic to the type of device, be it physical or virtual, or type of the policy, be it dynamic or static. Using these interfaces, it would become possible to write different kinds of application (e.g. GUI portal, template engine, etc.) to control the implementation of security policies on security functional elements, though how these applications are implemente are completely out of the scope of the I2NSF working group, which is only focused on the interfaces.

This document captures the requirements for the client-facing interface that can be easily used by security administrators without knowledge of specific security devices or features. We refer to this as "user-construct" based interfaces. To further clarify, in the scope of this document, the "user-construct" here does not mean some free-from natural language input or an abstract intent such as "I want my traffic secure" or "I don't want DDoS attacks in my network"; rather the user-construct here means that policies are described using client-oriented expressions such as application names, application groups, device groups, user groups etc. with a vocabulary of verbs (e.g., drop, tap, throttle), prepositions, conjunctions, conditionals, adjectives, and nouns instead of using standard n-tuples from the packet header.

2. Conventions Used in this Document

BSS:	Business Support System
CLI:	Command Line Interface
CMDB:	Configuration Management Database
Controller:	Used interchangeably with Service Provider Security Controller or management system throughout this document
CRUD:	Create, Retrieve, Update, Delete
FW:	Firewall
GUI:	Graphical User Interface

IDS:	Intrusion Detection System
IPS:	Intrusion Protection System
LDAP:	Lightweight Directory Access Protocol
NSF:	Network Security Function, defined by [I-D.ietf-i2nsf-problem-and-use-cases]
OSS:	Operation Support System
RBAC:	Role Based Access Control
SIEM:	Security Information and Event Management
URL:	Universal Resource Locator
vNSF:	Refers to NSF being instantiated on Virtual Machines

3. Guiding principles for definition of Client-Facing Interfaces

The "Client-Facing Interface" ensures that a security administrator can deploy any device from any vendor and still be able to use a consistent interface. In essence, this interface gives ability to security admins to express their security policies independent of how security functions are implemented in their deployment. Henceforth, in this document, we use "security policy management interface" interchangeably when we refer to the client-facing interface.

3.1. User-construct based modeling

Traditionally, security policies have been expressed using proprietary interfaces. These interface are defined by a vendor either based on CLI or a GUI system; but more often these interfaces are built using vendor specific networking construct such IP address, protocol and application constructs with L4-L7 information. This requires security operator to translate their oragnzational business objectives into actionable security policies on the device using vendor specific configuration. But, this alone is not sufficient to render policies in the network as operator also need to identify the device in the network topology where a policy need to be enforced in a complex environment with potenial multiple policy enforcement points.

The User-construct based framework defines constructs such as user-group, application-group, device-group and location-group. The security admin would use these constructs to express a security policy instead of proprietary vendor specific constructs. The policy

defined in such a manner is referred to user-construct based policies in this draft. The idea is to enable security admin to use constructs they understand best in expressing security policies; which simplify their tasks and help avoiding human errors in complex security provisioning.

3.2. Basic rules for client interface definition

The basic rules in defining the client-facing interfaces are as follows:

o Not depending on particular network topology or the actual NSF location in the network

o Not requiring the exact knowledge of the concrete features and capabilities supported in the deployed NSFsa&#128;&#157;

- o Independent of the nature of the function that will apply the expressed policies be it stateful firewall,IDP, IDS, Router, Switch

- o Declarative/Descriptive model instead of Imperative/Prescriptive model - What security policies need to be enforced (declarative) instead of how they would be actually implemented (imperative)

- o Not depending on any specific vendor implementation or form-factor (physical, virtual) of the NSF

- o Not depending on how a NSF becomes operational - Network connectivity and other hosting requirements.

- o Not depending on NSF control plane implementation (if there is one) E.g., cluster of NSFs active as one unified service for scale and/ or resilience.

- o Not depending on specific data plane implementation of NSF i.e. Encapsulation, Service function chains.

Note that the rules stated above only apply to the client-facing interface where a user will define a high level policy. These rules do not apply to the lower layers e.g. security controller that convert the higher level policies into lower level constructs. The lower layers may still need some intelligence such as topology awareness, capability of the NSF and its functions, supported encapsulations etc. to convert and apply the policies accurately on the NSF devices.

3.3. Deployment Models for Implementing Security Policies

Traditionally, medium and larger operators deploy management systems to manage their statically-defined security policies. This approach may not be suitable nor sufficient for modern automated and dynamic data centers that are largely virtualized and rely on various management systems and controllers to dynamically implement security policies over any types of resources.

There are two different deployment models in which the client-facing interface referred to in this document could be implemented. These models have no direct impact on the client-facing interface, but illustrate the overall security policy and management framework and where the various processing functions reside. These models are:

a. Management without an explicit management system for control of devices and NSFs. In this deployment, the security controller acts as a NSF policy management system that takes information passed over the client security policy interface and translates into data on the I2NSF NSF-facing interface. The I2NSF interfaces are implemented by security device/function vendors. This would usually be done by having an I2NSF agent embedded in the security device or NSF. This deployment model is shown in Figure 1.

```
                                 RESTful API
```

```
                    SUPA or I2NSF Policy Management
                                  ^
                                  |
    Client-facing Interface       |
    (Independent of individual    |
     NSFs, devices,and vendors)|
                                  |
                    ------------------------------
                   |                              |
                   |       Security Controller    |
                   |                              |
                    ------------------------------
                         |                 ^
                         |   I2NSF         |
    NSF Interface        |   NSF-facing    |
    (Specific to NSFs)   |   Interface     |
                     ..............................
                         |                 |
                         v                 |


                   -------------     -------------
                  | I2NSF Agent |   | I2NSF Agent |
                  |-------------|   |-------------|
                  |             |---|             |
                  |     NSF     |   |     NSF     |
        NSFs      |             |   |             |
    (virtual       -------------\   /-------------
       and               |       \ /       |
    physical)            |        X        |
                         |       / \       |
                   -------------/   \-------------
                  | I2NSF Agent |   | I2NSF Agent |
                  |-------------|   |-------------|
                  |             |---|             |
                  |     NSF     |   |     NSF     |
                  |             |   |             |
                   -------------     -------------
```

Figure 1: Deployment without Management System

b. Management with an explicit management system for control of devices and NSFs. This model is similar to the model above except that security controller interacts with a dedicated management system which could either proxy I2NSF NSF-facing interfaces or could provide a layer where security devices or

NSFs do not support an I2NSF agent to process I2NSF NSF-facing interfaces. This deployment model is shown in Figure 2.

```
                            RESTful API
                    SUPA or I2NSF Policy Management
                                  ^
                                  |
    Client-facing Interface       |
    (Independent of individual    |
     NSFs,devices,and vendors)  |
                                  |
                    ------------------------------
                   |                              |
                   |       Security Controller    |
                   |                              |
                    ------------------------------
                         |                 ^
                         |   I2NSF         |
    NSF Interface        |   NSF-facing    |
    (Specific to NSFs)   |   Interface     |
                     ..............................
                         |                 |
                         v                 |
                    ------------------------------
                   |                              |
                   |      I2NSF Proxy Agent /     |
                   |      Management System       |
                   |                              |
                    ------------------------------
                         |                 ^
                         |  Proprietary    |
                         |  Functional     |
                         |  Interface      |
                     ..............................
                         |                 |
                         v                 |

                   -------------     -------------
                  |             |---|             |
                  |     NSF     |   |     NSF     |
        NSFs      |             |   |             |
    (virtual       -------------\   /-------------
       and               |       \ /       |
    physical)            |        X        |
                         |       / \       |
                -------------/   \-------------
               |             |---|             |
```

```
               |     NSF     |   |     NSF     |
               |             |   |             |
                -------------     -------------
```

Figure 2: Deployment with Management System or I2NSF Proxy Agent

Although the deployment models discussed here don't necessarily affect the client security policy interface, they do give an overall context for defining a security policy interface based on abstraction.

4. Functional Requirements for the Client-Facing Interface

As stated in the guiding principles for defining I2NSF client-facing interface, the security policies and the client-facing interface shall be defined from a user/client perspective and abstracted away from the type of NSF, NSF specific implementation, controller implementation, NSF topology, NSF interfaces, controller NSF-facing interfaces. Thus, the security policy definition shall be declarative, expressing the user construct, and driven by how security administrators view security policies from the definition, communication and deployment perspective.

The security controller's implementation is outside the scope of this document and the I2NSF working group.

In order to express and build security policies, high level requirements for the client-facing are as follows:

o Multi-Tenancy

o Authentication and Authorization

o Role-Based Access Control (RBAC)

o Protection from Attacks

o Protection from Misconfiguration

- Policy Lifecycle Management

- Dynamic Policy Endpoint Groups

- Policy Rules

- Policy Actions

- Generic Policy Model

- Policy Conflict Resolution

- Backward Compatibility

- Third-Party Integration

- Telemetry Data

The above requirements are by no means a complete list and may not be sufficient for all use-cases and all operators, but should be a good starting point for a wide variety of use-cases in Service Provider and Enterprise networks.

4.1. Requirement for Multi-Tenancy in client interface

A security administrator that uses security policies may have internal tenants and would like to have a framework wherein each tenant manages its own security policies with isolation from other tenants.

An operator may be a cloud service provider with multi-tenant deployments, where each tenant is a different customer. Each tenant or customer must be allowed to manage its own security policies.

It should be noted that tenants may have their own tenants, so a recursive relation may exist. For instance, a tenant in a cloud service provider may have multiple departments or organizations that need to manage their own security rules.

Some key concepts are listed below and used throughout the document hereafter:

Policy-Tenant: An entity that owns and manages the security Policies applied on its resources.

Policy-Administrator: A user authorized to manage the security policies for a Policy-Tenant.

Policy-User: A user within a Policy-Tenant who is authorized to access certain resources of that tenant according to the privileges assigned to it.

4.2. Requirement for Authentication and Authorization of client interface

Security administrators MUST authenticate to and be authorized by the security controller before they are able to issue control commands and any policy data exchange commences.

There must be methods defined for the Policy-Administrator to be authenticated and authorized to use the security controller. There are several authentication methods available such as OAuth, XAuth and X.509 certificate based. The authentication scheme between Policy-Administrator and security controller may also be mutual instead of one-way. Any specific method may be determined based on organizational and deployment needs and outside the scope of I2NSF. In addition, there must be a method to authorize the Policy-Administrator for performing certain action. It should be noted that, depending on the deployment model, Policy-Administrator authentication and authorization to perform actions communicated to the controller could be performed as part of a portal or another system prior to communication the action to the controller.

4.3. Requirement for Role-Based Access Control (RBAC) in client interface

Policy-Authorization-Role represents a role assigned to a Policy-User that determines whether a user or has read-write access, read-only access, or no access for certain resources. A User can be mapped to a Policy-Authorization-Role using an internal or external identity provider or mapped statically.

4.4. Requirement to protect client interface from attacks

There Must be protections from attacks, malicious or otherwise, from clients or a client impersonator. Potential attacks could come from a botnet or a host or hosts infected with virus or some unauthorized entity. It is recommended that security controller use a dedicated IP interface for client-facing communications and those communications should be carried over an isolated out-of-band network. In addition, it is recommended that traffic between clients and security controllers be encrypted. Furthermore, some straightforward traffic/session control mechanisms (i.e., Rate-limit, ACL, White/Black list) can be employed on the security controller to defend against DDoS flooding attacks.

4.5. Requirement to protect client interface from misconfiguration

There Must be protections from mis-configured clients. System and policy validations should be implemented to detect this. Validation may be based on a set of default parameters or custom tuned thresholds such as the number of policy changes submitted, number of objects requested in a given time interval, etc.

4.6. Requirement to manage policy lifecycle with diverse needs

In order to provide more sophisticated security framework, there should be a mechanism to express that a policy becomes dynamically active/enforced or inactive based on either security administrator's manual intervention or an event.

One example of dynamic policy management is when the security administrator pre-configures all the security policies, but the policies get activated or deactivated based on dynamic threats.

Basically, a threat event may activate certain inactive policies, and once a new event indicates that the threat has gone away, the policies become inactive again.

There are following ways for dynamically activating policies:

o The policy may be dynamically activated by the I2NSF client or associated management entity, and dynamically communicated over the I2NSF client-facing interface to the controller to program I2NSF functions using the I2NSF NSF-facing interface

o The policy may be pulled dynamically by the controller upon detecting an event over the I2NSF monitoring interface

o The policy may be statically pushed to the controller and dynamically programmed on the NSFs upon potentially detecting another event

o The policy can be programmed in the NSF, and activated or deactivated upon policy attributes, like time or admin enforced.

The client-facing interface should support the following policy attributes for policy enforcement:

Admin-Enforced: The policy, once configured, remains active/enforced until removed by the security administrator.

Time-Enforced: The policy configuration specifies the time profile that determines when policy is activated/enforced. Otherwise, it is de-activated.

Event-Enforced: The policy configuration specifies the event profile that determines when policy is activated/enforced. It also specifies the duration attribute of that policy once activated based on event. For instance, if the policy is activated upon detecting an application flow, the policy could be de-activated when the corresponding session is closed or the flow becomes inactive for certain time.

A policy could be a composite policy, that is composed of many rules, and subject to updates and modification. For the policy maintenance, enforcement, and auditability purposes, it becomes important to name and version the policies. Thus, the policy definition SHALL support policy naming and versioning. In addition, the i2NSF client-facing interface SHALL support the activation, deactivation, programmability, and deletion of policies based on name and version. In addition, it should support reporting on the state of policies by name and version. For instance, a client may probe the controller about the current policies enforced for a tenant and/or a sub-tenant (organization) for auditability or verification purposes.

4.7. Requirement to define dynamic policy Endpoint group

When the security administrator configures a security policy, it may have requirement to apply this policy to certain subsets of the network. The subsets may be identified based on criteria such as users, devices, and applications. We refer to such a subset of the network as a "Policy Endpoint Group".

One of the biggest challenges for a security administrator is how to make sure that security policies remain effective while constant changes are happening to the "Policy Endpoint Group" for various reasons (e.g., organizational, network and application changes). If a policy is created based on static information such as user names, application, or network subnets; then every time this static information change, policies need to be updated. For example, if a policy is created that allows access to an application only from the group of Human Resource users (the HR-users group), then each time the HR- users group changes, the policy needs to be updated.

We call these dynamic Policy Endpoint Groups "Meta-data Driven Groups". The meta-data is a tag associated with endpoint information such as users, applications, and devices. The mapping from meta-data to dynamic content could come either from standards-based or proprietary tools. The security controller could use any available mechanisms to derive this mapping and to make automatic updates to the policy content if the mapping information changes. The system SHOULD allow for multiple, or sets of tags to be applied to a single network object.

The client-facing policy interface must support endpoint groups for user-construct based policy management. The following meta-data driven groups MAY be used for configuring security polices:

User-Group: This group identifies a set of users based on a tag or on static information. The tag to identify user is dynamically derived from systems such as Active Directory or LDAP. For example, an operator may have different user-groups, such as HR-users, Finance-users, Engineering-users, to classify a set of users in each department.

Device-Group: This group identifies a set of devices based on a tag or on static information. The tag to identify device is dynamically derived from systems such as configuration mannagement database (CMDB). For example, a security administrator may want to classify all machines running one operating system into one group and machines running another operating system into another group.

Application-Group: This group identifies a set of applications based on a tag or on static information. The tag to identify application is dynamically derived from systems such as CMDB. For example, a security administrator may want to classify all applications running in the Legal department into one group and all applications running under a specific operating system into another group. In some cases, the application can semantically associated with a VM or a device. However, in other cases, the application may need to be associated with a set of identifiers (e.g., transport numbers, signature in the application packet payload) that identify the application in the corresponding packets. The mapping of application names/tags to signatures in the associated application packets should be defined and communicated to the NSF. The client-facing Interface shall support the communication of this information.

Location-Group: This group identifies a set of location tags. Tag may correspond 1:1 to location. The tag to identify location is either statically defined or dynamically derived from systems such as CMDB. For example, a security administrator may want to classify all sites/locations in a geographic region as one group.

4.8. Requirement to express rich set of policy rules

The security policy rules can be as simple as specifying a match for the user or application specified through "Policy Endpoint Group" and take one of the "Policy Actions" or more complicated rules that specify how two different "Policy Endpoint Groups" interact with each other. The client-facing interface must support mechanisms to allow the following rule matches.

Policy Endpoint Groups: The rule must allow a way to match either a single or a member of a list of "Policy Endpoint Groups".

There must be a way to express a match between two "Policy Endpoint Groups" so that a policy can be effective for communication between two groups.

Direction: The rule must allow a way to express whether the security administrator wants to match the "Policy Endpoint Group" as the source or destination. The default should be to match both directions, if the direction rule is not specified in the policy.

Threats: The rule should allow the security administrator to express a match for threats that come either in the form of feeds (such as botnet feeds, GeoIP feeds, URL feeds, or feeds from a SIEM) or speciality security appliances. Threats could be identified by Tags/names in policy rules. The tag is a label of one or more event types that may be detected by a threat detection system.

The threat could be from malware and this requires a way to match for virus signatures or file hashes.

4.9. Requirement to express rich set of policy actions

The security administrator must be able to configure a variety of actions within a security policy. Typically, security policy specifies a simple action of "deny" or "permit" if a particular condition is matched. Although this may be enough for most of the simple policies, the I2NSF client-facing interface must also provide a more comprehensive set of actions so that the interface can be used effectively across various security functions.

Policy action MUST be extensible so that additional policy action specifications can easily be added.

The following list of actions SHALL be supported:

Permit: This action means continue processing the next rule or allow the packet to pass if this is the last rule. This is often a default action.

Deny: This action means stop further packet processing and drop the packet.

Drop connection: This action means stop further packet processing, drop the packet, and drop connection (for example, by sending a TCP reset).

Log: This action means create a log entry whenever a rule is matched.

Authenticate connection: This action means that whenever a new connection is established it should be authenticated.

Quarantine/Redirect: This action may be relevant for event driven policy where certain events would activate a configured policy that quarantines or redirects certain packets or flows. The redirect action must specify whether the packet is to be tunneled and in that case specify the tunnel or encapsulation method and destination identifier.

Netflow: This action creates a Netflow record; Need to define Netflow server or local file and version of Netflow.

Count: This action counts the packets that meet the rule condition.

Encrypt: This action encrypts the packets on an identified flow. The flow could be over an Ipsec tunnel, or TLS session for instance.

Decrypt: This action decrypts the packets on an identified flow. The flow could be over an Ipsec tunnel, or TLS session for instance.

Throttle: This action defines shaping a flow or a group of flows that match the rule condition to a designated traffic profile.

Mark: This action defines traffic that matches the rule condition by a designated DSCP value and/or VLAN 802.1p Tag value.

Instantiate-NSF: This action instantiates an NSF with a predefined profile. An NSF can be any of the FW, LB, IPS, IDS, honeypot, or VPN, etc.

WAN-Accelerate: This action optimizes packet delivery using a set of predefined packet optimization methods.

Load-Balance: This action load balances connections based on predefined LB schemes or profiles.

The policy actions should support combination of terminating actions and non-terminating actions. For example, Syslog and then Permit; Count and then Redirect.

Policy actions SHALL support any L2, L3, L4-L7 policy actions.

4.10. Requirement to express policy in a generic model

Client-facing interface SHALL provide a generic metadata model that defines once and then be used by appropriate model elements any times, regardless of where they are located in the class hierarchy, as necessary.

Client-facing interface SHALL provide a generic context model that enables the context of an entity, and its surrounding environment, to be measured, calculated, and/or inferred.

Client-facing interface SHALL provide a generic policy model that enables context-aware policy rules to be defined to change the configuration and monitoring of resources and services as context changes.

Client-facing interface SHALL provide the ability to apply policy or multiple sets of policies to any given object. Policy application process SHALL allow for nesting capabilities of given policies or set of policies. For example, an object or any given set of objects could have application team applying certain set of policy rules, while network team would apply different set of their policy rules. Lastly, security team would have an ability to apply its set of policy rules, being the last policy to be evaluated against.

4.11. Requirement to detect and correct policy conflicts

Client-facing interface SHALL be able to detect policy "conflicts", and SHALL specify methods on how to resolve these "conflicts"

For example: two clients issues conflicting set of security policies to be applied to the same Policy Endpoint Group.

4.12. Requirement for backward compatibility

It MUST be possible to add new capabilities to client-facing interface in a backward compatible fashion.

4.13. Requirement for Third-Party integration

The security policies in the security administrator's network may require the use of specialty devices such as honeypots, behavioral analytics, or SIEM in the network, and may also involve threat feeds, virus signatures, and malicious file hashes as part of comprehensive security policies.

The client-facing interface must allow the security administrator to configure these threat sources and any other information to provide integration and fold this into policy management.

4.14. Requirement to collect telemetry data

One of the most important aspect of security is to have visibility into the networks. As threats become more sophisticated, the security administrator must be able to gather different types of telemetry data from various devices in the network. The collected data could simply be logged or sent to security analysis engines for behavioral analysis, policy violations, and for threat detection.

The client-facing interface MUST allow the security administrator to collect various kinds of data from NSFs. The data source could be syslog, flow records, policy violation records, and other available data.

Detailed client-facing interface telemetry data should be available between clients and security controllers. Clients should be able to subscribe and receive these telemetry data.

client should be able to receive notifications when a policy is dynamically updated.

5. Operational Requirements for the Client-Facing Interface

5.1. API Versioning

The client-facing interface must support a version number for each RESTful API. This is very important because the client application and the controller application may most likely come from different vendors. Even if the vendor is same, it is hard to imagine that two different applications would be released in lock step.

Without API versioning, it is hard to debug and figure out issues if application breaks. Although API versioning does not guarantee that applications will always work, it helps in debugging if the problem is caused by an API mismatch.

5.2. API Extensiblity

Abstraction and standardization of the client-facing interface is of tremendous value to security administrators as it gives them the flexibility of deploying any vendor's NSF without needing to redefine their policies or change the client interface. However this might also look like as an obstacle to innovation.

If a vendor comes up with new feature or functionality that can't be expressed through the currently defined client-facing interface, there must be a way to extend existing APIs or to create a new API that is relevant for that NSF vendor only.

5.3. APIs and Data Model Transport

The APIs for client interface must be derived from the YANG based data model. The YANG data model for client interface must capture all the requirements as defined in this document to express a security policy. The interface between a client and controller must be reliable to ensure robust policy enforcement. One such transport mechanism is RESTCONF that uses HTTP operations to provide necessary CRUD operations for YANG data objects, but any other mechanism can be used.

5.4. Notification

The client-facing interface must allow the security administrator to collect various alarms and events from the NSF in the network. The events and alarms may be either related to security policy enforcement or NSF operation. The events and alarms could also be used as a input to the security policy for autonomous handling.

5.5. Affinity

The client-facing interface must allow the security administrator to pass any additional metadata that a user may want to provide for a security policy e.g. certain security policy needs to be applied only on linux machine or windows machine or that a security policy must be applied on the device with Trusted Platform Module chip.

5.6. Test Interface

The client-facing interface must allow the security administrator the ability to test the security policies before the policies are actually applied e.g. a user may want to verify if a policy creates potential conflicts with the existing policies or whether a certain policy can be implemented. The test interface provides such capabilities without actually applying the policies.

6. IANA Considerations

This document requires no IANA actions. RFC Editor: Please remove this section before publication.

7. Acknowledgements

The authors would like to thank Adrian Farrel, Linda Dunbar and Diego R.Lopez from IETF I2NSF WG for helpful discussions and advice.

The authors would also like to thank Kunal Modasiya, Prakash T. Sehsadri and Srinivas Nimmagadda from Juniper networks for helpful

discussions.

8. Normative References

[I-D.ietf-i2nsf-problem-and-use-cases]
Hares, S., Dunbar, L., Lopez, D., Zarny, M., and C. Jacquenet, "I2NSF Problem Statement and Use cases", draft-ietf-i2nsf-problem-and-use-cases-02 (work in progress), October 2016.

Authors' Addresses

Rakesh Kumar
Juniper Networks
1133 Innovation Way
Sunnyvale, CA 94089
US

Email: rkkumar@juniper.net

Anil Lohiya
Juniper Networks
1133 Innovation Way
Sunnyvale, CA 94089
US

Email: alohiya@juniper.net

Dave Qi
Bloomberg
731 Lexington Avenue
New York, NY 10022
US

Email: DQI@bloomberg.net

Nabil Bitar
Nokia
755 Ravendale Drive
Mountain View, CA 94043
US

Email: nabil.bitar@nokia.com

Senad Palislamovic
Nokia
755 Ravendale Drive
Mountain View, CA 94043
US

Email: senad.palislamovic@nokia.com

Liang Xia
Huawei
101 Software Avenue
Nanjing, Jiangsu 210012
China

Email: Frank.Xialiang@huawei.com