I2NSF Working Group
Internet-Draft
Intended status: Informational
Expires: October 28, 2017

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

April 26, 2017

Requirements for Client-Facing Interface to Security Controller
draft-ietf-i2nsf-client-facing-interface-req-01

Abstract

This document captures requirements for Client-Facing interface to Security Controller. The interface is expressed using objects and constructs understood by Security Admin as opposed to vendor or device specific expressions associated with individual product and feature. This document identifies a broad set of requirements needed to express security policies based on User-constructs which are well understood by User Community. This gives ability to decouple policy definition from policy enforcement on a specific element, be it a physical or virtual.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 28, 2017.

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 device and features. This has been the biggest challenge for both Service Provider and Enterprise, henceforth named as Security Admin in this document. This challenge is further amplified due to network virtualization with security functions deployed in physical and virtual form factor, henceforth named as network security function (NSF) in this document, from multiple vendors with proprietary interface.

Even if a Security Admin deploys a single vendor solution with one or more security appliances across its entire network, it is still very difficult to manage security policies due to complexity of security features, and mapping of business requirements to vendor specific configuration. The Security Admin may use vendor provided management systems to provision and manage security policies. But, the single vendor approach is highly restrictive in today's network for following reasons:

- An organization may not be able to rely on a single vendor because the changing security requirements may not align with vendor's release cycle.

- A large organization may have a presence across different sites and regions; which means, it may not be possible to deploy same solution from the same vendor because of regional regulatory and compliance policy.

- o If and when an organization migrates from one vendor to another, it is almost impossible to migrate security policies from one vendor to another without complex and time consuming manual workflows.

- o An organization may deploy multiple network security functions in virtual and physical forms to attain the flexibility, elasticity, performance scale, 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 An organization may choose all or part of their assets such as routers, switches, firewalls, and overlay-networks as policy enforcement points for operational and cost efficiency. It would be highly complex to manage policy enforcement with different tool set for each type of device.

In order to facilitate deployment of security policies across different vendor provided NSFs, the Interface to Network Security Functions (I2NSF) working group in the IETF is defining a Client-Facing interface to Security Controller [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 enforcement point. Using these interfaces, it becomes possible to write different kinds of security management applications (e.g. GUI portal, template engine, etc.) allowing Security Admin to express security policies in an abstract form with choice of wide variety of NSF for policy enforcement. The implementation of security management applications or controller is completely out of the scope of the I2NSF working group, which is only focused on interface definition.

This document captures the requirements for Client-Facing interface that can be easily used by Security Admin without a need for expertise in vendor and device specific feature set. 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 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 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 principle for Client-Facing Interface defintion

Client-Facing Interface must ensure that a Security Admin can deploy any NSF from any vendor and should still be able to use the same consistent interface. In essence, this interface must allow a Security Admin to express security policies independent of how NSFs are implemented in their deployment. Henceforth, in this document, we use "security policy management interface" interchangeably when we refer to Client-Facing interface.

3.1. User-construct based modeling

Traditionally, security policies have been expressed using proprietary interface. The interface is defined by a vendor based on proprietary command line text or a GUI based system with implementation specific constructs such IP address, protocol and L4-L7 information. This requires Security Admin to translate their business objectives into vendor provided constructs in order to express a security policy. But, this alone is not sufficient to render a policy in the network; the admin must also understand network and application design to locate a specific policy enforcement point to make sure policy is effective. This may be highly manual task based on specific network design and may become unmanageable in virtualized networks.

The User-construct based framework does not rely on lower level semantics due to problem explained above, but rather uses higher level constructs such as User-group, Application-group, Device-group, Location-group, etcetera. A Security Admin would use these constructs to express a security policy instead of proprietary implementation or feature 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-Facing Interface definition

The basic rules in defining the Client-Facing interfaces are as

follows:

- Not dependent on particular Network topology or the NSF location in the network
- Not requiring deep knowledge of proprietary features and capabilities supported in the deployed NSFsa€
- Independent of NSF type that will implement the user security policy be it a stateful firewall,IDP, IDS, Router, Switch
- Declarative/Descriptive model instead of Imperative/Prescriptive model - What security policy need to be enforced (declarative) instead of how it is implemented (imperative)
- Not dependent on any specific vendor implementation or form-factor (physical, virtual) of the NSF
- Not dependent on how a NSF becomes operational - Network connectivity and other hosting requirements.
- Not dependent on NSF control plane implementation (if there is one) E.g., cluster of NSFs active as one unified service for scale and/ or resilience.
- 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, which a user would use to express a high level policy. These rules do not apply to the lower layers e.g. Security Controller that convert 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.

3.3. Deployment Models for Implementing Security Policies

Traditionally, medium and large Enterprise 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 NSF.

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 various processing functions reside. These models are:

a. Policy management without an explicit management system for control of NSFs. In this deployment, Security Controller acts as a NSF management system; it takes information passed over Client-Facing interface and translates into data on I2NSF NSF-facing interface. The NSF-Facing interface is implemented by NSF vendors; this would usually be done by having an I2NSF agent embedded in the NSF. This deployment model is shown in Figure 1.

RESTful API

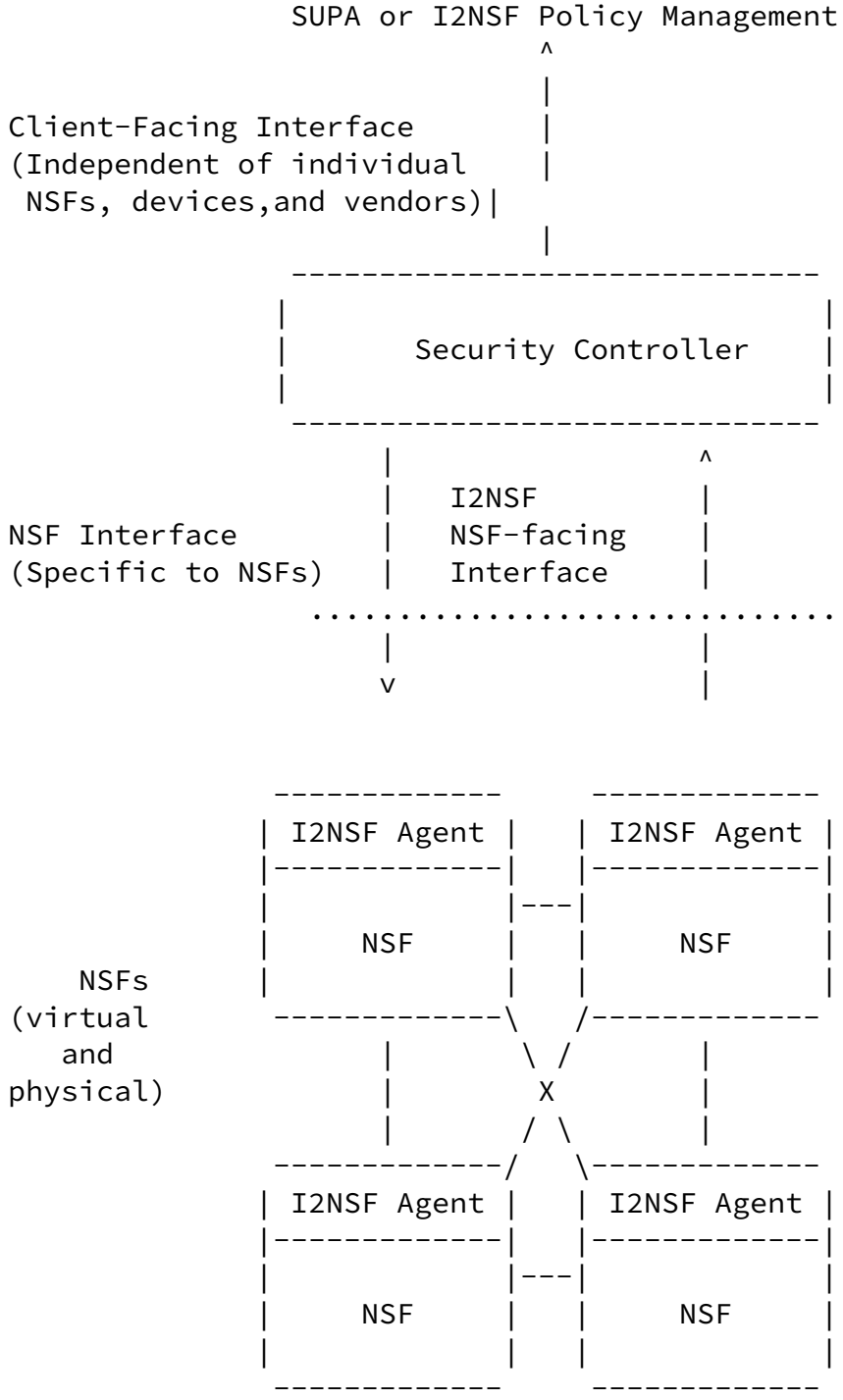
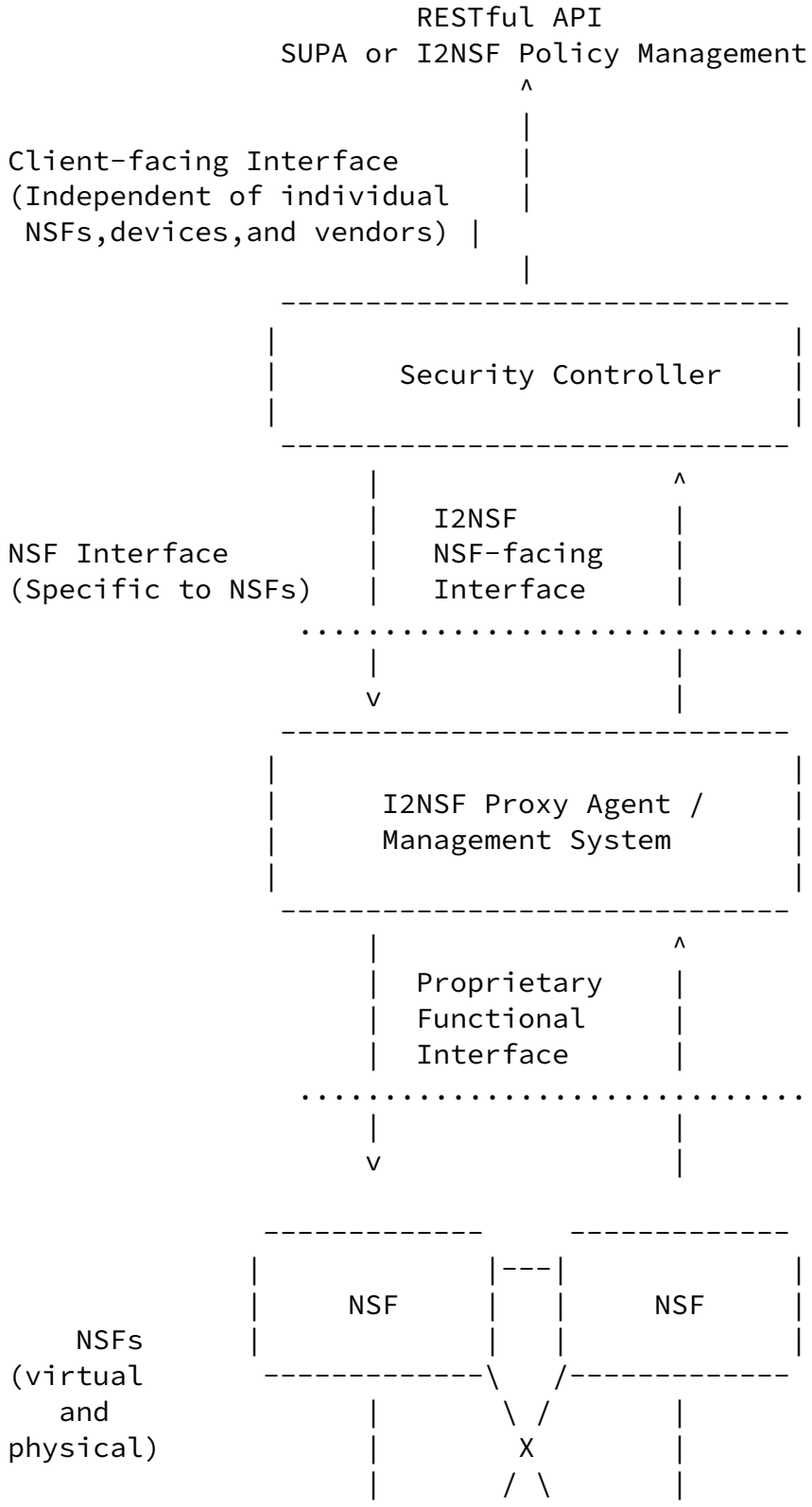


Figure 1: Deployment without Management System

b. Policy management with an explicit management system for control of NSFs. This model is similar to the model above except that Security Controller interacts with a vendor's dedicated management system that proxy I2NSF NSF-Facing interfaces as NSF

may not support NSF-Facing interface. This is a useful model to support legacy NSF. This deployment model is shown in Figure 2.



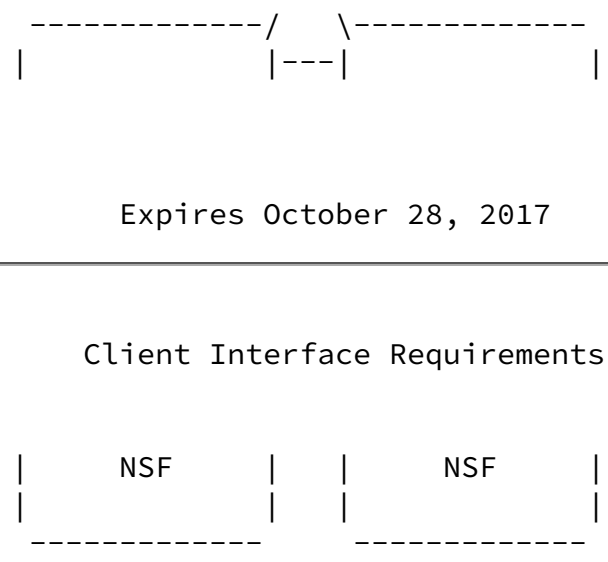


Figure 2: Deployment with Management System or I2NSF Proxy Agent

Although the deployment models discussed here don't necessarily affect the definition of Client-Facing 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 principle for defining the I2NSF Client-Facing interface, the security policies and the Client-Facing interface shall be defined from user's or client's perspective and abstracted away from type of NSF, NSF specific implementation, controller implementation, network topology, controller NSF-Facing interface. Thus, the security policy definition shall be declarative, expressed using User-construct, and driven by how Security Admin view security policies from definition, communication and deployment perspective.

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 requirement for Client-Facing interface is as follows:

- Multi-Tenancy
- Authentication and Authorization
- Role-Based Access Control (RBAC)
- Protection from Attacks
- 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 Security Admins, 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 Admin may have internal tenants and might want a framework wherein each tenant manages its own security policies with isolation from other tenants.

A Security Admin may be a cloud service provider with multi-tenant deployment, 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 for compliance.

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 Admin MUST authenticate to and be authorized by Security Controller before they are able to issue control commands and any policy data exchange commands.

There must be methods defined for Policy-Administrator to be authenticated and authorized to use Security Controller. There are several authentication methods available such as OAuth, XAuth and X.509 certificate based; the authentication may be mutual or single-sided based on business needs and outside the scope of I2NSF. In addition, there must be a method o authorize the Policy-Administrator to perform certain action. It should be noted that, Policy-Administrator authentication and authorization to perform actions could be part of Security Controller or outside; this document does not mandate any specific implementation but requires that such a scheme must be implemented.

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 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

The interface must be protections from attacks, malicious or otherwise, from clients or a client impersonator. Potential attacks could come from Botnets, 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 controller be encrypted. Furthermore, some straightforward traffic/session control mechanisms (i.e., Rate-limit, ACL, White/Black list) can be employed on 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 so that a policy becomes dynamically active/ enforced or inactive based on Security Admin's manual intervention or some external event.

One example of dynamic policy management is when Security Admin 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 Security Admin or associated management entity

o The policy may be dynamically activated by Security Controller upon detecting an external event or an event from I2NSF monitoring interface

o The policy can be statically configured but activated or deactivated upon policy attributes, such as timing calendar

Client-Facing interface should support the following policy attributes for policy enforcement:

Admin-Enforced: A policy, once configured, remains active/enforced until removed by Security Admin.

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

Event-Enforced: A policy configuration specifies the event profile that determines when the policy is to be 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 audit-ability purposes, it becomes important to name and version Security Policy. 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 operational state of policies by name and version. For instance, a client may probe Security Controller whether a Security Policy is enforced for a tenant and/or a sub-tenant (organization) for audit-ability or

verification purposes.

4.7. Requirement to define dynamic policy Endpoint group

When Security Admin 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 Admin is how to make sure that a Security Policy 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 (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 User, Application, or Device. The mapping from meta-data to dynamic content could come from a standards-based or proprietary tools. Security Controller could use any available mechanisms to derive this mapping and to make automatic updates to 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.

Client-Facing policy interface must support endpoint groups for expressing a Security Policy. 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 organization 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 management database (CMDB). For example, a Security Admin may want to classify all machines running a particular operating system into one group and machines running a different 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 Admin may want to classify all applications running in the Legal department into one group and all applications running in the HR department 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 Admin 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 Security Admin 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

Security Admin 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 a 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 Security Admin 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, Security Admin 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 Security Admin 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 Controller. 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

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 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 an 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 Client-Facing interface is of tremendous value to Security Admin as it gives them the flexibility of deploying any vendor's NSF without need to redefine their policies if or when a NSF is changed.

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

Client-Facing interface must allow Security Admin to collect various alarms and events from the NSF in the network. The events and alarms may be either related to security policy itself or related to NSF where the policy is implemented. The events and alarms could also be used as an input to the security policy for autonomous handling as specified in a given security policy.

5.5. Affinity

Client-Facing interface must allow Security Admin to pass any additional metadata that a user may want to provide for a security policy e.g. certain security policy needs to be implemented only on a very highly secure NSF with Trusted Platform Module (TPM) chip. A user may require Security Controller not to share the NSF with other tenant, this must be expressed through the interface API.

5.6. Test Interface

Client-Facing interface must allow Security Admin ability to test a security policy before it is enforced e.g. a user may want to verify whether the policy creates any potential conflicts with the existing policies or if there are even resources to enforce the policy. The test policy would provide such a capability without actually applying the policies.

6. Security Considerations

Client-Facing interface to Security controller must be protected to ensure that entire security posture is not compromised. This draft mandates that interface must have proper authentication and authorization with Role Based Access Controls to address multi-tenancy requirement. The draft does not specify a particular mechanism as different organization may have different needs based on their deployment. This has been discussed extensively in this draft.

Authentication and authorization alone may not be sufficient for Client-Facing interface; the interface API must be validated for proper inputs to guard against attacks. The type of checks and verification may be specific to each interface API, but a careful consideration must be made to ensure that Security Controller is not compromised.

7. IANA Considerations

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

8. 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.

9. Normative References

[I-D.ietf-i2nsf-problem-and-use-cases]
Hares, S., Lopez, D., Zarny, M., Jacquenet, C., Kumar, R., and J. Jeong, "I2NSF Problem Statement and Use cases", draft-ietf-i2nsf-problem-and-use-cases-15 (work in progress), April 2017.

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