I2NSF Working Group
Internet-Draft
Intended status: Informational
Expires: January 4, 2018

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

July 3, 2017

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

Abstract

This document captures requirements for Client-Facing interface to the Security Controller as defined by [I-D.ietf-i2nsf-framework]. 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 the User Community. This gives ability to decouple policy definition from policy enforcement on a specific security functional element, be it a physical or virtual.

Table of Contents

1. Introduction

Programming security policies in a network has been a fairly complex task that often requires deep knowledge of vendor specific devices and features. This has been the biggest challenge for both Service Providers and Enterprises, henceforth named as Security Admins in this document. This challenge is further amplified due to network virtualization with security functions deployed in physical and virtual form factors, henceforth named as network security function (NSF) in this document, from multiple vendors with proprietary interfaces.

Even if 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 that requires mapping of business needs to complex security features with vendor specific configurations. 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.

- 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.

- An organization may deploy multiple security functions in either virtual or physical form 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 a given security function.

- 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 Policy in an abstract form with choice of wide variety of NSF as policy enforcement point. The implementation of security management applications or controller is out of scope for I2NSF working group.

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 Security 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 definition

Client-Facing Interface must ensure that a Security Admin can deploy a NSF from any vendor and should still be able to use the same consistent interface. In essence, this interface allows a Security Admin to express a Security Policy enforced on the NSF to be independent of vendor and its implementation. 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 vendor 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. To further complicate the matters, when changes happen in the network topology, the Security Policy may require modifications accordingly. This may be a highly manual task based on network design and becomes unmanageable in virtualized environment.

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 a particular network topology or the NSF location in the network
- Not forced to express Security Policy with proprietary vendor specific interfaces for a given NSFa€
- Independent of NSF type that will implement a specific Security Policy; e.g., the interface remains same no matter if a specific Security Policy is enforced on a stateful firewall,IDP, IDS, Router or a Switch
- Declarative/Descriptive model instead of Imperative/Prescriptive model - What security policy need to be expressed (declarative) instead of how it is implemented (imperative)
- Not dependent on vendor's' 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, e.g. encapsulation, service function chains.

Note that the rules stated above only apply to the Client-Facing interface, which a Security Admin 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 Enterprises deploy vendor provided management systems to create Security Policies and any changes to these Security Policies are made manually over time by Security Admin. This approach may not be suitable and nor sufficient for modern highly automated data centers that are largely virtualized and rely on various management systems and controllers to implement dynamic Security Policies over large number of NSF in the network.

There are two distinct deployment models for Security Controller. Although, these have no direct impact on the Client-Facing interface, but illustrate the overall Security Policy management framework in an organization and how the Client-Facing interface remain same which is the main objective of this document. 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.

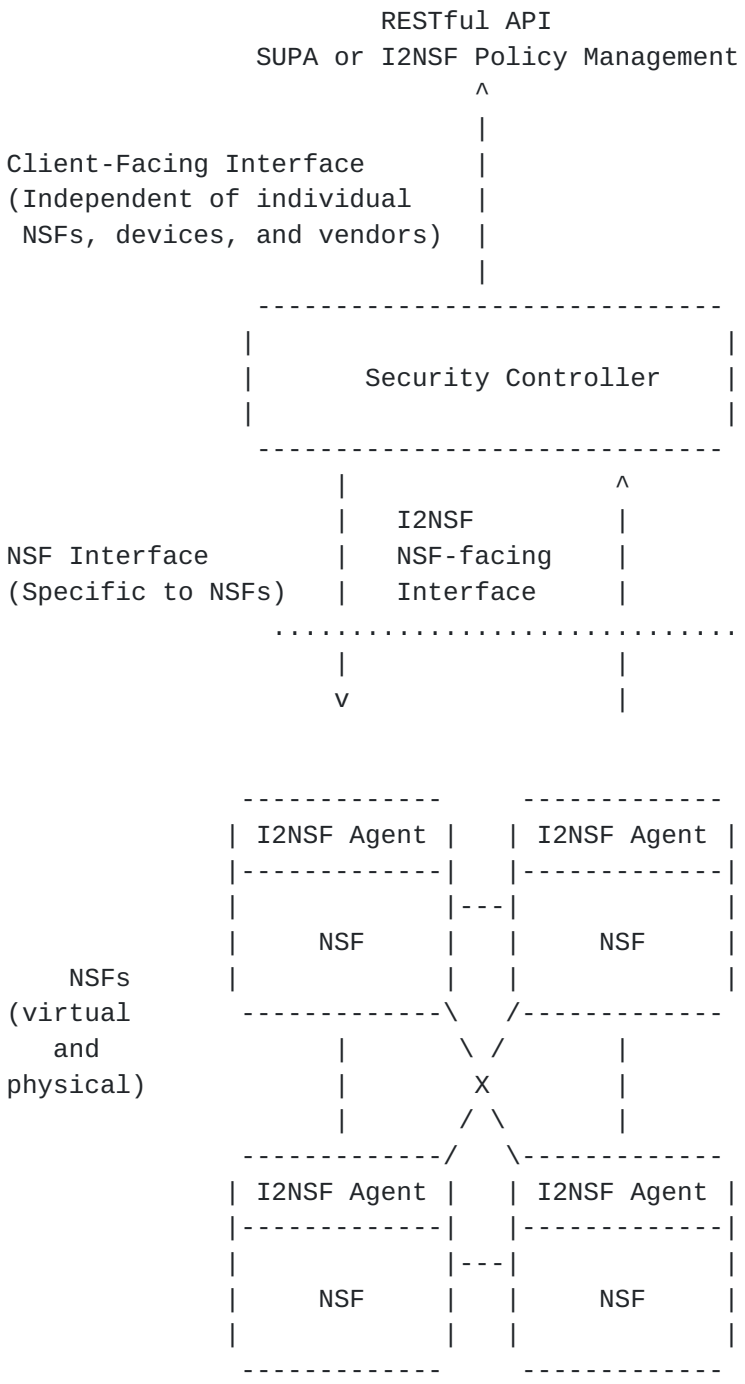

Figure 1: Deployment without Management System

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.

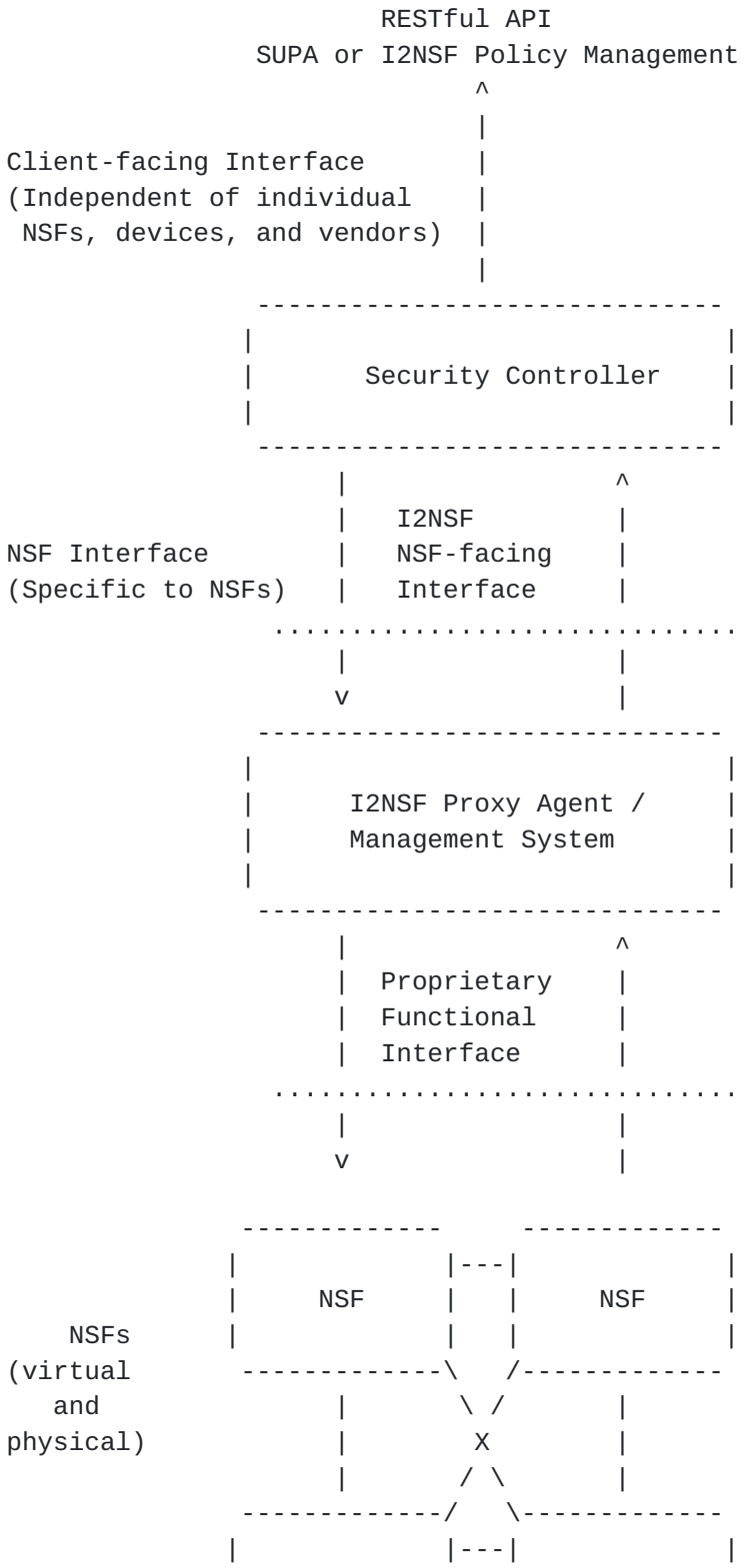

Figure 2: Deployment with Management System or I2NSF Proxy Agent

As mentioned above, these models discussed here don't affect the definition of Client-Facing interface, they do give an overall context for defining a Security Policy interface based on abstraction. This can help in implementing a Security Controller.

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 Security Admin'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 their business needs and objectives.

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 or required for all use-cases, but should be a good starting point for a wide variety of use-cases in Service Provider and Enterprise networks.

A specific implementation may not support all these requirements but in order to define a base set of requirements which would works for most use-cases, this document will make an attempt to classify these requirements in three categories:

MUST: This means, the requirement must be supported by Client-Facing interface.

RECOMMENDED: This means, we recommend that Client-Facing interface support this requirement since it might be applicable to large number of use-cases but some vendor may choose to omit if their focus is only certain market segments.

MAY: This means, the requirement is not mandatory for Client-Facing interface but may be needed for specific use-cases.

4.1. Requirement for Multi-Tenancy in Client-Facing interface

An organization may have internal tenants and might want a framework wherein each tenant manages its own Security Policies with isolation from other tenants. This requirement may be applicable to Service Providers and Large Enterprises so we classify this requirement in RECOMMENDED category. If an implement does not support this requirement, it must support a default implicit tenant created by Security Controller that owns all the Security Policies.

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 able to manage its own Security Policies without affecting other tenants.

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.

The following objects are needed to fulfill this requirement:

Policy-Tenant: An entity that owns and manages Security Policies applied to its own asset and 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-Facing interface

A Security Admin must be authenticated and authorized in order to manage Security Policies. We classify this requirement in MUST category since without proper authentication and authorization, the security posture of entire organization can be easily compromised.

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 [RFC6749], 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-Facing interface

A tenant in organization may have multiple users with each user given certain privileges. Some user such as "Admin" may have all the permission but other may have limited permissions. We classify this requirement in RECOMMENDED category since it aligns with Multi-Tenancy requirement. If this requirement is not supported, a default privilege must be assigned to all the users.

The following objects are needed to fulfill this requirement:

Policy-Authorization-Role: Defines the permissions assigned to a user such as creating and managing policies on specified resources. A user may not be allowed to change existing policies but only view them.

4.4. Requirement to protect Client-Facing interface from attacks

The interface must be protections against attacks from malicious clients or a client impersonator. Potential attacks could come from Botnets, hosts infected with virus or some unauthorized entities. This requirement is highly RECOMMENDED since it may not be needed if the entire framework is deployed in very controlled environment. But if needed, we recommend that Security Controller uses a out-of-band communication channel for Client-Facing interface. In addition,it is also recommended that traffic Client-Facing interface communication 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-Facing interface from misconfiguration

There must be protections from mis-configured clients. System and policy parameters 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. We consider this to be a MUST requirement but implementation aspects would depend upon each individual API communication.

4.6. Requirement to manage policy lifecycle with rich set of controls

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 multiple different criteria such as Security Admin's manual intervention or some external event. We consider requirement listed here to be a MUST for wide variety of use-cases.

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 threat detection. 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 activated by Security Admin manually using a client interface such as GUI or CLI.

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 configured but gets activated or deactivated upon specified timing calendar with Security Policy definition.

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 Security Admin 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". This requirement is the fundamental

building block of Client-Facing interface; so making it a MUST requirement. But object defined here may not support all use-cases and may not be required by everyone so it is left up to vendor whether all or partial set of these object is supported.

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 "Metadata Driven Groups". The metadata is a tag associated with endpoint information such as User, Application, or Device. The mapping from metadata 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 endpoint.

Client-Facing interface must support Endpoint Groups as a target for a Security Policy. The following metadata driven groups MAY be used for configuring Security Polices:

User-Group: This group identifies a set of users based on a tag or static information such as user-names. The tag identifying users, 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 device information. The tag identifying the devices, 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 application names. The tag identifying applications, 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 locations. Tag may correspond 1:1 to location. The tag identifying locations 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 Policy Rules is a central component of any Security Policy but rule requirements may vary based on use-cases and it is hard to define a complete set that works for everyone. In order to build a rich interface, we are going to take a different approach; we will define the building block of rules and let Security Admin build rules using these construct so that Security Policies meet their requirements:

Segmentation policies : This set of policies create rules for communication between two Endpoint Groups. An organization may restrict certain communication between a set of user and applications for example. The segmentation policy may be a micro-segmentation rule between components of complex applications or related to hybrid cloud deployment based on location.

Threat policies: This set of policies creates rules to prevent communication with externally or internally identified threats. The threats may be well knows such as threat feeds from external sources or dynamically identified by using specialty devices in the network.

Governance and Compliance policies: This set of policies creates rules to implement business requirement such as controlling access to internal or external resources for meeting regulatory compliance or business objectives.

In order to build a generic rule engine to satisfy diverse set of Policy Rules, we propose following objects:

In order to build a generic rule engine to satisfy diverse set of Policy Rules, we propose following objects:

Source Policy Endpoint Group: A source target of the Policy Rule. This may be special object "ALL" if all groups meet this criteria.

Destination Policy Endpoint Group: A destination target of the Policy Rule. This may be a special object "ALL", if all groups meet this criteria.

Direction: By default rules are applied in either direction but this object can be used to make rule definition uni-directional.

Threat Group: An object that represents a set of static or dynamic threats such as Botnet, GeoIP, URL feeds or virus and malware signatures detected dynamically. This object can be used as source or destination target in a rule.

Match Condition: An object that represents a set of allowed interactions. It could be as simple as group of application names or L4 ports allowed between two Endpoint Groups. It could very well that all traffic is allowed between two groups.

Exceptions: In order to truly build rules which are Security Admin and built with user semantics, we should allow to specify exceptions to the match criteria. This will greatly simplify Security Admin's task. E.g., we could build a rule that allows all traffic between two groups except a particular application or threat source.

Actions: Action is what makes rule and Policy work. The Action is defined in details in next section. We RECOMMEND that there be a one-to-one mapping between rule and action otherwise if multiple rules are associated with one action, it may be a difficult to manage Security Policy lifecycle as they evolve.

4.9. Requirement to express rich set of Policy Actions

Security Admin must be able to configure a variety of actions for a given Policy Rule. Typically, Security Policy specifies a simple action of "deny" or "permit" if a particular condition is matched. Although this may be enough for most use-cases, the I2NSF Client-Facing interface must provide a more comprehensive set of actions so that the interface can be used effectively across various security needs.

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 is useful for threat remediation purposes. If a security breach or infection point is detected, a Security Admin would like to isolate for purpose of remediation or controlling attack surface.

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, IPS, IDS, honeypot, or VPN, etc.

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 for consistent policy enforcement

As proposed in this document that the Client-Facing interface MUST be built using higher-level "User-Constructs" that are independent of network design and implementations. In order to achieve this, it becomes important that Security Controller functionality becomes more complex that keep track of various objects that are used to express Security Policies. The Security Controller MUST evaluate the Security Policies whenever these objects and network topology change to make sure that Security Policy is consistently enforced as expressed.

Although this document does not specify how Security Controller achieve this and any implementation challenges. It is assumed that once Security Controller uses Client-Facing interface to accept Security Policies; it would maintain the security posture as per the Security Policies during all changes in network or Endpoints and other building blocks of the framework.

An event must be logged by Security Controller when a Security Policy is updated due to changes in it's building blocks such as Endpoint Group contents or the Security Policy is moved from one enforcement point to another because the Endpoint has moved in the network. This may help in debugging and auditing for compliance reasons. The Security Admin may optionally receive notifications if supported and desired.

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 a newly expressed Security Policy could conflict with existing Security Policies applied to a set of Policy Endpoint Groups. This MUST be detected and Security Admin be allowed for manual correction if needed.

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 framework in a network may require the use of a specialty device such as honeypot, behavioral analytic, or SIEM for threat detection; the device may provide threat information such as threat feeds, virus signatures, and malicious file hashes.

The Client-Facing interface must allow Security Admin to include these devices under Security Controller's Client-Facing interface so that a Security Policy could be expressed using information from such devices; basically it allows ability to integrate third part devices into the Security Policy framework.

4.14. Requirement to collect telemetry data

One of the most important aspect of security is to have visibility into the network. As threats become more sophisticated, Security Admin must be able to gather different types of telemetry data from various NSFs 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.

Client-Facing interface must provide a set of telemetry data available to Security Admin from Security Controller. The Security Admin should be able to subscribe and receive to this data set.

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 important since Security Controller could be deployed by using multiple componenets and different pieces may come from different vendors; it is difficult to isolate and debug issues without ablility to track each component's operational behavior. Even if the vendor is same for all the components, it is hard to imagine that all pieces would be released in lock step by the vendor.

Without API versioning, it is hard to debug and figure out issues when deploying Security Controller and its components built overtime across multiple release cycles. Although API versioning does not guarantee that Security Controller would always work but it helps in debugging if the problem is caused by an API mismatch.

5.2. API Extensibility

Abstraction and standardization of Client-Facing interface is of tremendous value to Security Admins 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 SHALL be a way to extend existing APIs or to create a new API that addresses specific vendors's new NSF functionality.

5.3. APIs and Data Model Transport

The APIs for interface SHALL be derived from the YANG based data model. The data model for Client-Facing 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 and Monitoring

Client-Facing interface must allow ability to collect various alarms, events, statistics about enforcement and policy violations from NSFs in the network. The events and alarms may be associated with a specific policy or associated with operating conditions of a specific NSF in general. The statistics may be a measure of potential Security Policy violations or general data that reflect operational behavior of a NSF. The events, alarms and statistics may also be used as an input to automate Security Policy lifecycle management.

5.5. Affinity

Client-Facing interface must allow Security Admin to pass any additional metadata that a user may want to provide with a Security Policy e.g., if the policy needs to be enforced by a very highly secure NSF with Trusted Platform Module (TPM) chip. Another example would be, if a policy can not be enforced by a multi-tenant NSF. This would Security Admin control on operating environment

5.6. Test Interface

Client-Facing interface must support 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 existing policies or

if there are enough resources and capability to enforce this policy. The test interface would provide a mechanism to Security Admin where policies could be tested in the actual environment before enforcement.

6. Security Considerations

Client-Facing interface to Security controller must be protected to make sure that entire security posture is not compromised. This draft mandates that interface must have proper authentication and authorization control mechanisms to ward off malicious attacks. The draft does not specify a particular mechanism as different organization may have different needs based on their specific deployment environment and moreover new methods may evolve to better suit contemporary requirements.

Authentication and authorization alone may not be sufficient for Client-Facing interface; the interface API must be validated for proper input 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.

We recommend that all attack surface must be examined with careful consideration of the operating environment and available industry best practices must be used such as process and standards to protect security controller against malicious or inadvertent attacks.

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-framework]
Lopez, D., Lopez, E., Dunbar, L., Strassner, J., and R. Kumar, "Framework for Interface to Network Security Functions", draft-ietf-i2nsf-framework-06 (work in progress), July 2017.

[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-16 (work in progress), May 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