I2NSF
Internet Draft
Intended status: Standard Track
Expires: January 02, 2019

L. Xia
J. Strassner
Huawei
C. Basile
PoliTO
D. Lopez
TID

July 02, 2018

Information Model of NSFs Capabilities
draft-ietf-i2nsf-capability-02.txt

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), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on January 02, 2019.

Copyright Notice

Abstract

This draft defines the concept of an NSF (Network Security Function) capability, as well as its information model. Capabilities are a set of features that are available from a managed entity, and are represented as data that unambiguously characterizes an NSF. Capabilities enable management entities to determine the set of features from available NSFs that will be used, and simplify the management of NSFs.

Table of Contents

1. Introduction

The rapid development of virtualized systems requires advanced security protection in various scenarios. Examples include network

devices in an enterprise network, User Equipment in a mobile network, devices in the Internet of Things, or residential access users [RFC8192].

NSFs produced by multiple security vendors provide various security capabilities to customers. Multiple NSFs can be combined together to provide security services over the given network traffic, regardless of whether the NSFs are implemented as physical or virtual functions.

Security Capabilities describe the functions that Network Security Functions (NSFs) are available to provide for security policy enforcement purposes. Security Capabilities are independent of the actual security control mechanisms that will implement them.

Every NSF SHOULD be described with the set of capabilities it offers. Security Capabilities enable security functionality to be described in a vendor-neutral manner. That is, it is not needed to refer to a specific product or technology when designing the network; rather, the functions characterized by their capabilities are considered. Security Capabilities are a market enabler, providing a way to define customized security protection by unambiguously describing the security features offered by a given NSF.

This document is organized as follows. Section 2 defines conventions and acronyms used. Section 3 discusses the design principles for I2NSF capability information model, the related ECA model, and provides detailed information model design of I2NSF network security capability.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

This document uses terminology defined in [I-D.draft-ietf-i2nsf-terminology] for security related and I2NSF scoped terminology.

2.1. Acronyms

I2NSF - Interface to Network Security Functions

NSF - Network Security Function

DNF - Disjunctive Normal Form

3. Capability Information Model Design

A Capability Information Model (CapIM) is a formalization of the functionality that an NSF advertises. This enables the precise specification of what an NSF can do in terms of security policy enforcement, so that computer-based tasks can unambiguously refer to, use, configure, and manage NSFs. Capabilities MUST be defined in a vendor- and technology-independent manner (e.g., regardless of the differences among vendors and individual products).

Humans are able to refer to categories of security controls and understand each other. For instance, security experts agree on what is meant by the terms "NAT", "filtering", and "VPN concentrator". As a further example, network security experts unequivocally refer to "packet filters" as stateless devices able to allow or deny packet forwarding based on various conditions (e.g., source and destination IP addresses, source and destination ports, and IP protocol type fields) [Alshaer].

However, more information is required in case of other devices, like stateful firewalls or application layer filters. These devices filter packets or communications, but there are differences in the packets and communications that they can categorize and the states they maintain. Humans deal with these differences by asking more questions to determine the specific category and functionality of the device. Machines can follow a similar approach, which is commonly referred to as question-answering [Hirschman] [Galitsky]. In this context, the CapIM and the derived Data Models provide important and rich information sources.

Analogous considerations can be applied for channel protection protocols, where we all understand that they will protect packets by means of symmetric algorithms whose keys could have been negotiated with asymmetric cryptography, but they may work at different layers and support different algorithms and protocols. To ensure protection, these protocols apply integrity, optionally confidentiality, anti-reply protections, and authenticate peers.

The CapIM is intended to clarify these ambiguities by providing a formal description of NSF functionality. The set of functions that are advertised MAY be restricted according to the privileges of the user or application that is viewing those functions. I2NSF Capabilities enable unambiguous specification of the security capabilities available in a (virtualized) networking environment,

and their automatic processing by means of computer-based techniques.

This includes enabling the security controller to properly identify and manage NSFs, and allow NSFs to properly declare their functionality, so that they can be used in the correct way.

3.1. Design Principles and ECA Policy Model Overview

This document defines an information model for representing NSF capabilities. Some basic design principles for security capabilities and the systems that manage them are:

- Independence: each security capability SHOULD be an independent function, with minimum overlap or dependency on other capabilities. This enables each security capability to be utilized and assembled together freely. More importantly, changes to one capability SHOULD NOT affect other capabilities. This follows the Single Responsibility Principle [Martin] [OODSRP].

- Abstraction: each capability MUST be defined in a vendor-independent manner.

- Advertisement: A dedicated, well-known interface MUST be used to advertise and register the capabilities of each NSF. This same interface MUST be used by other I2NSF Components to determine what Capabilities are currently available to them.

- Execution: a dedicated, well-known interface MUST be used to configure and monitor the use of a capability. This provides a standardized ability to describe its functionality, and report its processing results. This facilitates multi-vendor interoperability.

- Automation: the system MUST have the ability to auto-discover, auto-negotiate, and auto-update its security capabilities (i.e., without human intervention). These features are especially useful for the management of a large number of NSFs. They are essential for adding smart services (e.g., refinement, analysis, capability reasoning, and optimization) to the security scheme employed. These features are supported by many design patterns, including the Observer Pattern [OODOP], the Mediator Pattern [OODMP], and a set of Message Exchange Patterns [Hohpe].

- Scalability: the management system SHOULD have the capability to scale up/down or scale in/out. Thus, it can meet various performance requirements derived from changeable network traffic or service requests. In addition, security capabilities that are affected by scalability changes SHOULD support reporting statistics to the security controller to assist its decision on whether it needs to invoke scaling or not.

Based on the above principles, this document defines a capability model that enables an NSF to register (and hence advertise) its set of capabilities that other I2NSF Components can use. These capabilities MAY have their access control restricted by policy; this is out of scope for this document. The set of capabilities provided by a given set of NSFs unambiguously define the security offered by the set of NSFs used. The security controller can compare the requirements of users and applications to the set of capabilities that are currently available in order to choose which capabilities of which NSFs are needed to meet those requirements. Note that this choice is independent of vendor, and instead relies specifically on the capabilities (i.e., the description) of the functions provided.

Furthermore, when an unknown threat (e.g., zero-day exploits and unknown malware) is reported by an NSF, new capabilities may be created, and/or existing capabilities may be updated (e.g., by updating its signature and algorithm). This results in enhancing the existing NSFs (and/or creating new NSFs) to address the new threats. New capabilities may be sent to and stored in a centralized repository, or stored separately in a vendor's local repository. In either case, a standard interface facilitates the update process. This document specifies a metadata model that MAY be used to further describe and/or prescribe the characteristics and behavior of the I2NSF capability model. For example, in this case, metadata could be used to describe the updating of the capability, and prescribe the particular version that an implementation should use. This initial version of the model covers and has been validated to describe NSFs that are designed with a set of capabilities (which covers most of the existing NSFs). Checking the behavior of the model with systems that change capabilities dynamically at runtime has been extensively explored (e.g., impact on automatic registration).

The "Event-Condition-Action" (ECA) policy model in [RFC8329] is used as the basis for the design of the capability model; definitions of all I2NSF policy-related terms are also defined in [I-D.draft-ietf-i2nsf-terminology]. The following three terms define the structure and behavior of an I2NSF imperative policy rule:

- Event: An Event is defined as any important occurrence in time of a change in the system being managed, and/or in the environment of the system being managed. When used in the context of I2NSF Policy Rules, it is used to determine whether the Condition clause of the I2NSF Policy Rule can be evaluated or not. Examples of an I2NSF Event include time and user actions (e.g., logon, logoff, and actions that violate an ACL).

- Condition: A condition is defined as a set of attributes, features, and/or values that are to be compared with a set of known attributes, features, and/or values in order to determine whether or not the set of Actions in that (imperative) I2NSF Policy Rule can be executed or not. Examples of I2NSF Conditions include matching attributes of a packet or flow, and comparing the internal state of an NSF to a desired state.

- Action: An action is used to control and monitor aspects of flow-based NSFs when the event and condition clauses are satisfied. NSFs provide security functions by executing various Actions. Examples of I2NSF Actions include providing intrusion detection and/or protection, web and flow filtering, and deep packet inspection for packets and flows.

An I2NSF Policy Rule is made up of three Boolean clauses: an Event clause, a Condition clause, and an Action clause. This structure is also called an ECA (Event-Condition-Action) Policy Rule. A Boolean clause is a logical statement that evaluates to either TRUE or FALSE. It may be made up of one or more terms; if more than one term is present, then each term in the Boolean clause is combined using logical connectives (i.e., AND, OR, and NOT).

An I2NSF ECA Policy Rule has the following semantics:

```
IF <event-clause> is TRUE

   IF <condition-clause> is TRUE

      THEN execute <action-clause> [constrained by metadata]

   END-IF

END-IF
```

Technically, the "Policy Rule" is really a container that aggregates the above three clauses, as well as metadata. Aggregating metadata enables business logic to be used to prescribe behavior. For example, suppose a particular ECA Policy Rule contains three actions (A1, A2, and A3, in that order). Action A2 has a priority of 10; actions A1 and A3 have no priority specified. Then, metadata may be used to restrict the set of actions that can be executed when the event and condition clauses of this ECA Policy Rule are evaluated to be TRUE; two examples are: (1) only the first action (A1) is executed, and then the policy rule returns to its caller, or (2) all actions are executed, starting with the highest priority.

The above ECA policy model is very general and easily extensible.

3.2. Relation with the External Information Model

Note: the symbology used from this point forward is taken from section 3.3 of [I-D.draft-ietf-supa-generic-policy-info-model].

The I2NSF NSF-Facing Interface is used to select and manage the NSFs using their capabilities. This is done using the following approach:

1) Each NSF registers its capabilities with the management system through a dedicated interface, and hence, makes its capabilities available to the management system;

2) The security controller compares the needs of the security service with the set of capabilities from all available NSFs that it manages using the CapIM;

3) The security controller uses the CapIM to select the final set of NSFs to be used;

4) The security controller takes the above information and creates or uses one or more data models from the CapIM to manage the NSFs;

5) Control and monitoring can then begin.

This assumes that an external information model is used to define the concept of an ECA Policy Rule and its components (e.g., Event, Condition, and Action objects). This enables I2NSF Policy Rules [I-D.draft-ietf-i2nsf-terminology] to be subclassed from an external information model.

The external ECA Information Model supplies at least a set of objects that represent a generic ECA Policy Rule, and a set of objects that represent Events, Conditions, and Actions that can be

aggregated by the generic ECA Policy Rule. This enables appropriate I2NSF Components to reuse this generic model for different purposes, as well as specialize it (i.e., create new model objects) to represent concepts that are specific to I2NSF and/or an application that is using I2NSF.

It is assumed that the external ECA Information Model also has the ability to aggregate metadata. This enables metadata to be used to prescribe and/or describe characteristics and behavior of the ECA Policy Rule. Specifically, Capabilities are subclassed from this external metadata model. If the desired Capabilities are already defined in the CapIM, then no further action is necessary. Otherwise, new Capabilities SHOULD be defined either by defining new classes that can wrap existing classes using the decorator pattern [Gamma] or by another mechanism (e.g., through subclassing); the parent class of the new Capability SHOULD be either an existing CapIM metadata class or a class defined in the external metadata information model. In either case, the ECA objects can use the existing aggregation between them and the Metadata class to add metadata to appropriate ECA objects.

Detailed descriptions of each portion of the information model are given in the following sections.

3.3. I2NSF Capability Information Model Theory of Operation

Capabilities are typically used to represent NSF functions that can be invoked. Capabilities are objects, and hence, can be used in the event, condition, and/or action clauses of an I2NSF ECA Policy Rule.

The I2NSF CapIM refines a predefined (and external) metadata model; the application of I2NSF Capabilities is done by refining a predefined (and external) ECA Policy Rule information model that defines how to use, manage, or otherwise manipulate a set of capabilities. In this approach, an I2NSF Policy Rule is a container that is made up of three clauses: an event clause, a condition clause, and an action clause. When the I2NSF policy engine receives a set of events, it matches those events to events in active ECA Policy Rules. If the event matches, then this triggers the evaluation of the condition clause of the matched I2NSF Policy Rule. The condition clause is then evaluated; if it matches, then the set of actions in the matched I2NSF Policy Rule MAY be executed. The operation of each of these clauses MAY be affected by metadata that is aggregated by either the ECA Policy Rule and/or by each clause, as well as the selected resolution strategy.

Condition clauses are logical formulas that combine one or more conditions that evaluate to a Boolean (i.e., true or false) result. The values in a condition clause are built on values received or owned by the NSF. For instance, the condition clause 'ip source == 1.2.3.4' is true when the IP address is equal to 1.2.3.4. Two or more conditions require a formal mechanism to represent how to operate on each condition to produce a result. For the purposes of this document, every condition clause MUST be expressed in either conjunctive or disjunctive normal form. Informally, conjunctive normal form expresses a clause as a set of sub-clauses that are logically ANDed together, where each sub-clause contains only terms that use OR and/or NOT operators). Similarly, disjunctive normal form is a set of sub-clauses that are logically ORed together, where each sub-clause contains only terms that use AND and/or NOT operators.

This document defines additional important extensions to both the external ECA Policy Rule model and the external Metadata model that are used by the I2NSF CapIM; examples include resolution strategy, external data, and default actions. All these extensions come from the geometric model defined in [Bas12]. A more detailed description is provided in Appendix E; a summary of the important points of this geometric model follows.

Formally, given a set of actions in an I2NSF Policy Rule, the resolution strategy maps all the possible subsets of actions to an outcome. In other words, the resolution strategy is included in an I2NSF Policy to decide how to evaluate all the actions from the matching I2NSF Policy Rule.

Some concrete examples of resolution strategy are:

o First Matching Rule (FMR)

o Last Matching Rule (LMR)

o Prioritized Matching Rule (PMR) with Errors (PMRE)

o Prioritized Matching Rule with No Errors (PMRN)

In the above, a PMR strategy is defined as follows:

1. Order all actions by their Priority (highest is first, no priority is last); actions that have the same priority may be appear in any order in their relative location.

2. For PMRE: if any action fails to execute properly, temporarily stop execution of all actions. Invoke the error handler of the failed action. If the error handler is able to recover from the error, then continue execution of any remaining actions; else, terminate execution of the ECA Policy Rule.

3. For PMRN: if any action fails to execute properly, stop execution of all actions. Invoke the error handler of the failed action, but regardless of the result, execution of the ECA Policy Rule MUST be terminated.

Regardless of the resolution strategy, when no rule matches a packet, a default action MAY be executed.

Resolution strategies may use, besides intrinsic rule data (i.e., event, condition, and action clauses), "external data" associated to each rule, such as priority, identity of the creator, and creation time. Two examples of this are attaching metadata to the policy action and/or policy rule, and associating the policy rule with another class to convey such information.

3.3.1. I2NSF Capability Information Model

Figure 1 below shows one example of an external model. This is a simplified version of the MEF Policy model [PDO]. For our purposes:

- MCMPolicyObject is an abstract class, and is derived from MCMManagedEntity [MCM]

- MCMPolicyStructure is an abstract superclass for building different types of Policy Rules (currently, for I2NSF, only imperative (i.e., ECA) Policy Rules are considered)

- An I2NSFECAPolicyRule could be subclassed from MCMECAPolicyRule

- I2NSF Events, Conditions, and Actions could be subclasses from MCMPolicyEvent, MCMPolicyCondition, and MCMPolicyAction

- MCMMetaData is aggregated by MCMEntity, which is the superclass of MCMManagedEntity. So all Policy objects may aggregate MCMMetaData

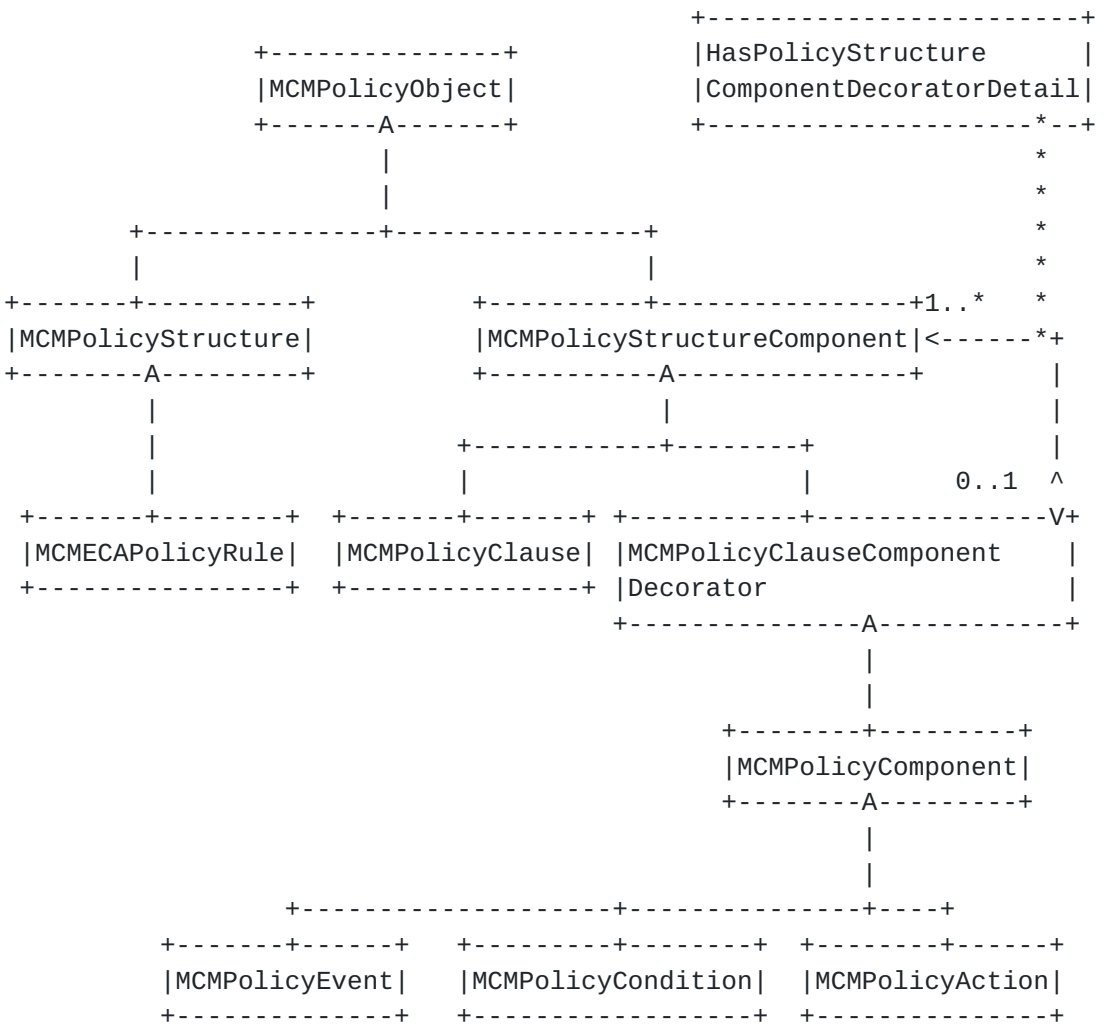

Figure 1 Exemplary External Information Model (from the MEF)

The CapIM model uses the Decorator Pattern [Gamma]. The decorator pattern enables a base object to be "wrapped" by zero or more decorator objects. The Decorator MAY attach additional characteristics and behavior, in the form of attributes at runtime in a transparent manner without requiring recompilation and/or redeployment. This is done by using composition instead of inheritance. Objects can "wrap" (more formally, extend the interface

of) an object. In essence, a new object can be built out of pre-existing objects.

The Decorator Pattern is applied to allow NSF instances to aggregate I2NSFSecurityCapability instances. By means of this aggregation, an NSF can be associated to the functions it provides in terms of security policy enforcement, both at specification time (i.e., when a vendor provides a new NSF), statically, when a NSF is added to a (virtualized) networking environment, and dynamically, during network operations. Figure 2 shows an NSF aggregating zero or more SecurityCapabilities. This may be thought of as an NSF possessing (or defining) zero or more Security Capabilities. This "possession" (or "definition") is represented in UML as an aggregated, called HasSecurityCapability. The hasSecurityCapabilityDetail is an association class that allows NSF instances to aggregate I2NSFSecurityCapability instances. An NSF MAY be described by 0 or more SecurityCapabilities.

Since there can be many types of NSF that have many different types of I2NSFSecurityCapabilities, the definition of a SecurityCapability must be done using the context of an NSF. This is realized by an association class in UML. HasSecurityCapabilityDetail is an association class. This yields the following design:

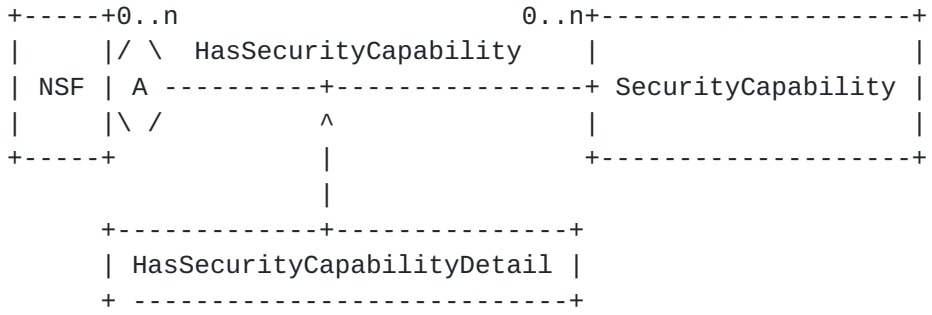

Figure 2 Defining SecurityCapabilities of an NSF

This enables the HasSecurityCapabilityDetail association class to be the target of a Policy Rule. That is, the HasSecurityCapabilityDetail class has attributes and methods that define which I2NSFSecurityCapabilities of this NSF are visible and can be used [MCM].

3.3.2. The SecurityCapability class

The SecurityCapability class defines the concept of metadata that define security-related capabilities. It is subclassed from an appropriate class of an external metadata information

model.Subclasses of the SecurityCapability class can be used to answer the following questions:

- What are the events that are caught by the NSF to trigger the condition clause evaluation (Event subclass)?
- What kind of condition clauses can be specified on the NSF to define valid rules? This question splits into two questions: (1) what are the conditions that can be specified (Condition subclass), and (2) how to build a valid condition clause from a set of individual conditions (ClauseEvaluation class).
- What are the actions that the NSF can enforce (Action class)?
- How to define a correct policy on the NSF?

3.3.3. I2NSF Condition Clause Operator Types

After having analyzed the literature and some existing NSFs, the types of selectors are categorized as exact-match, range-based, regex-based, and custom-match [Bas15][Lunt].

Exact-match selectors are (unstructured) sets: elements can only be checked for equality, as no order is defined on them. As an example, the protocol type field of the IP header is an unordered set of integer values associated to protocols. The assigned protocol numbers are maintained by the IANA (http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml).

In this selector, it is only meaningful to specify condition clauses that use either the "equals" or "not equals operators":

```
proto = tcp, udp       (protocol type field equals to TCP or UDP)

proto != tcp           (protocol type field different from TCP)
```

No other operators are allowed on exact-match selectors. For example, the following is an invalid condition clause, even if protocol types map to integers:

```
proto < 62             (invalid condition)
```

Range-based selectors are ordered sets where it is possible to naturally specify ranges as they can be easily mapped to integers. As an example, the ports in the TCP protocol may be represented

using a range-based selector (e.g., 1024-65535). For example, the following are examples of valid condition clauses:

```
source_port = 80

source_port < 1024

source_port < 30000 && source_port >= 1024
```

We include, in range-based selectors, the category of selectors that have been defined by Al-Shaer et al. as "prefix-match" [Alshaer]. These selectors allow the specification of ranges of values by means of simple regular expressions. The typical case is the IP address selector (e.g., 10.10.1.*). There is no need to distinguish between prefix match and range-based selectors as 10.10.1.* easily maps to [10.10.1.0, 10.10.1.255].

Another category of selector types includes the regex-based selectors, where the matching is performed by using regular expressions. This selector type is used frequently at the application layer, where data are often represented as strings of text. The regex-based selector type also includes string-based selectors, where matching is evaluated using string matching algorithms (SMA) [Cormen]. Indeed, for our purposes, string matching can be mapped to regular expressions, even if in practice SMA are much faster. For instance, Squid (http://www.squid-cache.org/), a popular Web caching proxy that offers various access control capabilities, allows the definition of conditions on URLs that can be evaluated with SMA (e.g., dstdomain) or regex matching (e.g., dstdom_regex).

As an example, the condition clause:

```
URL = *.website.*
```

matches all the URLs that contain a subdomain named website and the ones whose path contain the string ".website.". As another example, the condition clause:

```
MIME_type = video/*
```

matches all MIME objects whose type is video.

Finally, the idea of a custom check selector is introduced. For instance, malware analysis can look for specific patterns, and returns a Boolean value if the pattern is found or not.

In order to be properly used by high-level policy-based processing systems (such as reasoning systems and policy translation systems), these custom check selectors can be modeled as black-boxes (i.e., a function that has a defined set of inputs and outputs for a particular state), which provide an associated Boolean output.

More examples of custom check selectors will be presented in the next versions of the draft. Some examples are already present in Section 6.

3.3.4. Capability Selection and Usage

Capability selection and usage are based on the set of security traffic classification and action features that an NSF provides; these are defined by the capability model. If the NSF has the classification features needed to identify the packets/flows required by a policy, and can enforce the needed actions, then that particular NSF is capable of enforcing the policy.

NSFs may also have specific characteristics that automatic processes or administrators need to know when they have to generate configurations, like the available resolution strategies and the possibility to set default actions.

The capability information model can be used for two purposes: describing the features provided by generic security functions, and describing the features provided by specific products. The term Generic Network Security Function (GNSF) refers to the classes of security functions that are known by a particular system. The idea is to have generic components whose behavior is well understood, so that the generic component can be used even if it has some vendor-specific functions. These generic functions represent a point of interoperability, and can be provided by any product that offers the required capabilities. GNSF examples include packet filter, URL filter, HTTP filter, VPN gateway, anti-virus, anti-malware, content filter, monitoring, and anonymity proxy; these will be described later in a revision of this draft as well as in an upcoming data model contribution.

The next section will introduce the algebra to compose the information model of capability registration, defined to associate NSFs to capabilities and to check whether a NSF has the capabilities needed to enforce policies.

3.3.5. Capability Algebra

We introduce a Capability Algebra to ensure that the actions of different policy rules do not conflict with each.

Formally, two I2NSF Policy Rules conflict with each other if:

- the event clauses of each evaluate to TRUE
- the condition clauses of each evaluate to TRUE
- the action clauses affect the same object in different ways

For example, if we have two Policy Rules in the same Policy:

R1: During 8am-6pm, if traffic is external, then run through FW
R2: During 7am-8pm, conduct anti-malware investigation

There is no conflict between R1 and R2, since the actions are different. However, consider these two rules:

R3: During 8am-6pm, John gets GoldService
R4: During 10am-4pm, FTP from all users gets BronzeService

R3 and R4 are now in conflict, between the hours of 10am and 4pm, because the actions of R3 and R4 are different and apply to the same user (i.e., John).

Let us define the concept of a "matched" policy rule as one in which its event and condition clauses both evaluate to true. Then, the behavior of the Policy Rule, as specified by the CapIM, is defined by a 6-tuple {Ac, Cc, Ec, RSc, Dc, EVc}, where:

- Ac is the set of Actions currently available from the NSF;
- Cc is the set of Capabilities currently available from the NSF;
- Ec is the set of Events that an NSF can catch. Note that for NSF (e.g., a packet filter) that are not able to react to events, this set will be empty;
- RSc is the set of Resolution Strategies that can be used to specify how to resolve conflicts that occur between the actions of the same or different policy rules that are matched and contained in this particular NSF;

o Dc defines the notion of a Default action. This action can be either an explicit action that has been chosen {a}, or a set of actions {F}, where F is a dummy symbol (i.e., a placeholder value) that can be used to indicate that the default action can be freely selected by the policy editor. This is denoted as {F} U {a}.

EVc defines the set of Condition Clause Evaluation Rules that can be used at the NSF to decide when the condition clause is true given the result of the evaluation of the individual conditions. Before introducing the rest of the capability model, we will introduce the symbols that we will use to represent set operations:

o "U" is the union operation, A U B returns a new set that includes all the elements in A and all the elements in B

o "\" is the set minus operation, A \ B returns all the elements that are in A but not in B.

Given two sets of capabilities, denoted as cap1=(Ac1,Cc1, Ec1,RSc1,Dc1,EVc1) and cap2=(Ac2,Cc2,Ec2,RSc2,Dc2,EVc2) two set operations are defined for manipulating capabilities:

o capability addition: cap1+cap2 = {Ac1 U Ac2, Cc1 U Cc2, Ec1 U Ec2, RSc1 U RSc2, Dc1 U DC2, EVc1 U EVc2}

o capability subtraction: cap_1-cap_2 = {Ac1 \ Ac2, Cc1 \ Cc2, Ec1 \ Ec2, RSc1 U RSc2, Dc1 U DC2, EVc1 U EVc2}

In the above formulae, "U" is the set union operator and "\" is the set difference operator.

The addition and subtraction of capabilities are defined as the addition (set union) and subtraction (set difference) of both the capabilities and their associated actions. Note that the Resolution Strategies and Default Actions are added in both cases.

As an example, assume that a packet filter capability, Cpf, is defined. Further, assume that a second capability, called Ctime, exists, and that it defines time-based conditions. Suppose we need to construct a new generic packet filter, Cpfgen, that adds time-based conditions to Cpf. Conceptually, this is simply the addition of the Cpf and Ctime capabilities, as follows:

```
Apf   =  {Allow, Deny}
Cpf   =  {IPsrc,IPdst,Psrc,Pdst,protType}
Epf   =  {}
```

```
RSpf  =  {FMR}
Dpf   =  {A1}
EVpf  =  {DNF}


Atime =  {Allow, Deny, Log}
Ctime =  {timestart, timeend, datestart, datestop}
Etime =  {}
RStime = {LMR}
Dtime  = {A2}
EVtime = {}
```

Then, Cpfgen is defined as:

```
Cpfgen = {Apf U Atime, Cpf U Ctime, Epf U Etime, RSpf U RStime,
      Dpf U Time, EVpf U EVtime}
       = {Allow, Deny, Log},
         {{IPsrc,IPdst,Psrc,Pdst,protType} U {timestart, timeend,
           datestart, datestop}}
         {}
         {FMR, LMR}
         {A1, A2}
         {DNF}
```

In other words, Cpfgen provides three actions (Allow, Deny, Log), filters traffic based on a 5-tuple that is logically ANDed with a time period, can use either FMR or LMR (but obviously not both), and can provide either A1 or A2 (but again, not both) as a default action. In any case, multiple conditions will be processed with DNF when evaluating the condition clause.

4. IANA Considerations

TBD

5. References

5.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2234] Crocker, D. and Overell, P.(Editors), "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium and Demon Internet Ltd., November 1997.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC5511] Farrel, A., "Routing Backus-Naur Form (RBNF): A Syntax Used to Form Encoding Rules in Various Routing Protocol Specifications", RFC 5511, April 2009.

[RFC3198] Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J., and S. Waldbusser, "Terminology for Policy-Based Management", RFC 3198, DOI 10.17487/RFC3198, November 2001, <http://www.rfc-editor.org/info/rfc3198>.

[RFC8192] Hares, S., Lopez, D., Zarny, M., Jacquenet, C., Kumar, R., and J. Jeong, "Interface to Network Security Functions (I2NSF): Problem Statement and Use Cases", RFC 8192, DOI 10.17487/RFC8192, July 2017, <https://www.rfc-editor.org/info/rfc8192>.

[RFC8329] Lopez, D., Lopez, E., Dunbar, L., Strassner, J. and R. Kumar, "Framework for Interface to Network Security Functions", RFC 8329, February 2018.

5.2. Informative References

[INCITS359 RBAC] NIST/INCITS, "American National Standard for Information Technology - Role Based Access Control", INCITS 359, April, 2003

[I-D.draft-ietf-i2nsf-terminology] Hares, S., et.al., "Interface to Network Security Functions (I2NSF) Terminology", Work in Progress, January, 2018

[I-D.draft-ietf-supa-generic-policy-info-model] Strassner, J., Halpern, J., Coleman, J., "Generic Policy Information Model for Simplified Use of Policy Abstractions (SUPA)", Work in Progress, May, 2017.

[Alshaer] Al Shaer, E. and H. Hamed, "Modeling and management of firewall policies", 2004.

[Bas12] Basile, C., Cappadonia, A., and A. Lioy, "Network-Level Access Control Policy Analysis and Transformation", 2012.

[Bas15] Basile, C. and A. Lioy, "Analysis of application-layer filtering policies with application to HTTP", 2015.

[Cormen] Cormen, T., "Introduction to Algorithms", 2009.

[Galitsky] Galitsky, B. and Pampapathi, R., "Can many agents answer questions better than one", First Monday, 2005; http://dx.doi.org/10.5210/fm.v10i1.1204

[Gamma] Gamma, E., Helm, R. Johnson, R., Vlissides, J., "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Nov, 1994. ISBN 978-0201633610

[Hirschman] Hirschman, L., and Gaizauskas, R., "Natural Language Question Answering: The View from Here", Natural Language Engineering 7:4, pgs 275-300, Cambridge University Press, 2001

[Hohpe] Hohpe, G. and Woolf, B., "Enterprise Integration Patterns", Addison-Wesley, 2003, ISBN 0-32-120068-3

[Lunt] van Lunteren, J. and T. Engbersen, "Fast and scalable packet classification", 2003.

[Martin] Martin, R.C., "Agile Software Development, Principles, Patterns, and Practices", Prentice-Hall, 2002, ISBN: 0-13-597444-5

[MCM] MEF, "MEF Core Model", Technical Specification MEF X, April 2018

[OODMP] http://www.oodesign.com/mediator-pattern.html

[OODSOP] http://www.oodesign.com/observer-pattern.html

[OODSRP] http://www.oodesign.com/single-responsibility-principle.html

[PDO] MEF, "Policy Driven Orchestration", Technical Specification MEF Y, January 2018

[Taylor] Taylor, D. and J. Turner, "Scalable packet classification using distributed crossproducting of field labels", 2004.

6. Acknowledgments

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Cataldo Basile
Politecnico di Torino
Corso Duca degli Abruzzi, 34
Torino, 10129
Italy
Email: cataldo.basile@polito.it

Liang Xia (Frank)
Huawei
101 Software Avenue, Yuhuatai District
Nanjing, Jiangsu 210012
China
Email: Frank.xialiang@huawei.com

John Strassner
Huawei
2330 Central Expressway
Santa Clara, CA 95050 USA
Email: John.sc.Strassner@huawei.com

Diego R. Lopez
Telefonica I+D
Zurbaran, 12
Madrid, 28010
Spain
Phone: +34 913 129 041
Email: diego.r.lopez@telefonica.com