I2NSF   L. Xia
Internet Draft   J. Strassner
Intended status: Standard Track   D.Zhang
                                                                  Huawei
                                                                   K. Li
                                                                 Alibaba
                                                               C. Basile
                                                                 A. Lioy
                                                                  PoliTO
                                                                D. Lopez
                                                                     TID
                                                                E. Lopez
                                                                Fortinet
                                                             N. BOUTHORS
                                                                  Qosmos
                                                             Luyuan Fang
                                                               Microsoft

Expires: December 2017   November 1, 2016

Information Model of NSFs Capabilities
draft-xibassnez-i2nsf-capability-00.txt

Status of this Memo

Copyright Notice

Abstract

This draft aims at defining the concept of capability. Capabilities are data that unambiguously characterize NSFs (Network Security Functions). Their correct definition will ease all the management and automation that can be. Moreover, it allows the definition of Interfaces to manage NSFs.

This draft is the first trial to merge two previous existing drafts on NSFs capabilities [I-D.draft-baspez-i2nsf-capabilities-00] and on the capability interface [I-D.draft-xia-i2nsf-capability-interface-im-06]. Further work will be needed to homogenize all the concepts and incorporate the feedback that will result after its publication.

Table of Contents

1. Introduction

The rapid development of virtualized systems, along with the demand of security services in virtualized systems, requires advanced security protection in various scenarios. Examples include network devices in an enterprise network, User Equipment (UE) in a mobile

network, devices in the Internet of Things (IoT), or residential access users [I-D.draft-ietf-i2nsf-problem-and-use-cases].

Capabilities are precise information that characterize in an unambiguous way (in a given virtualized system) what a NSF can do in terms of security policy enforcement and how a Controller can interact with it in order to enforce a security policy. Even if the aim is a general of capabilities, the unambiguity of capabilities is only assured in a given virtualized system.

According to [I-D.draft-ietf-i2nsf-framework], there are two types of I2NSF interfaces available for security rules provisioning:

o Interface between I2NSF clients and a security controller (Client Facing Interface): This is a service-oriented interface, whose main objective is to define a communication channel over which information defining security services controlling client's specific flow can be requested. This enables security information to be exchanged between various applications (e.g., OpenStack, or various BSS/OSS components) and other components (e.g., security controllers). The design goal of the service interface is to decouple the security service in the application layer from various kinds of security devices and their device-specific security functions.

Interface between NSFs (e.g., firewall, intrusion prevention, or anti-virus) and a security controller (NSFs Facing Interface): This interface is independent of how the NSFs are implemented (e.g., run in Virtual Machines (VMs) or physical appliances). The NSFs Facing Interface is used to decouple the security management scheme from the set of NSFs and their various implementations for this scheme, so as to specify and monitor a number of flow based security policies to individual NSFs. According to the definition in [I-D.draft-ietf-i2nsf-framework], NSFs Facing Interface includes sub-interfaces of Capability Interface, Registration Interface and Monitoring Interface. This document proposes the information model design for the first two interfaces (Capability and Registration), the Monitoring Interface information model is discussed in [I-D.draft-zhang-i2nsf-info-model-monitoring].

This document is organized as follows: Section 3 discusses the design principles for NSF capabilities and related ECA model. Section 4 further describes design principles for I2NSF capability information model. Section 5 provides more details about the design of network security capability. Section 6 presents further information on specific aspects of the information model.

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 references to [I-D.draft-ietf-i2nsf-terminology] for more specific security related and I2NSF scoped terminology definitions.

2.1. Terminology

AAA -Access control, Authorization, Authentication

ACL - Access Control List

AD - Active Directory

ANSI - American National Standards Institute

DDoS - Distributed Deny of Services

FW - Firewall

I2NSF - Interface to Network Security Functions

INCITS - International Committee for Information Technology Standards

IoT - Internet of Things

IPS - Intrusion Prevention System

LDAP - Lightweight Directory Access Protocol

NAT - Network Address Translation

NBI - North-bound Interface

NIST - National Institute of Standard Technology

NSF - Network Security Function

RBAC - Role Based Access Control

UE - User Equipment

URL - Uniform/Universal Resource Locator

VM - Virtual Machine

WAF - Web Application Firewall

3. Management of NSFs: Design Principles towards a model of Capabilities

Some basic principles for security capabilities and the systems that have to manage them need to be considered:

- o Flexibility: each security capability should be an independent function, with minimum overlap or dependency to other capabilities. This enables each security capability to be utilized and assembled together freely. More importantly, changes to one capability will not affect other capabilities;

- o High level of abstraction: each capability should be associated to a unified interface to make it programmable; this in turn provides a standardized ability to describe and report its processing results and corresponding statistics information. Furthermore, it facilitates the multi-vendor interoperability;

- o Automation: The system must have the ability to auto-discover, auto-negotiate, and auto-update security capabilities. These features are especially useful for the management of a large number of NSFs. They are essential to add smart services (refinement, analysis, capability reasoning, optimization) on top of the virtual layer;

- o Scalability: the management system must 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, the security capability must 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, a set of abstract and vendor-neutral capabilities with standard interfaces is needed together with a model of capabilities that allows to unambiguously determine what NSFs can do in terms of security policy enforcement. The security controller can compare the requirements of clients to the set of capabilities that are currently available in order to choose 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. This

also facilitates the customization of the functionality of the selected NSFs by setting the parameters of their interfaces.

Furthermore, when an unknown threat (e.g., zero-day exploits, unknown malware, and APTs) is reported by a network security device, new capabilities may be created, and/or existing capabilities may be updated (e.g., signature and algorithm), to correspond to the new functionality provided by the NSF to handle the threat. The new capabilities are provided from different vendors after their analysis of the new threats and subsequent installation of the functions required to report on (and possibly mitigate) the threat. New capabilities may be sent to and stored in a centralized repository, or stored separately in a local repository. In either cases, a standard interface is needed during this automated update process.

In defining the capabilities of a NSF, the "Event-Condition-Action" (ECA) policy rule set model in [I-D.draft-ietf-i2nsf-framework] should be here as the basis for the design:

o 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 policy rules for I2NSF, it is used to determine whether the Condition clause of the 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);

o Condition: 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 make a decision. When used in the context of policy rules for I2NSF, it is used to determine whether or not the set of Actions in that Policy Rule can be executed or not. The following are exemplary types of conditions:

  ? Packet content values: Refer to the kind of information or attributes acquired directly from the packet headers or payloads that can be used in the security policy. It can be any fields or attributes in the packet L2/L3/L4 header, or special segment of bytes in the packet payload;

  ? Context values: Refer to the context information for the received packets. It can be (and not limited to):

  * User: The user (or user group) information to which a network flow is associated. A user has many attributes, such as name, id, password, authentication mode, and so on. The combination of name and id (where id could be a password, a certificate, or other means of identifying the user) is often used in the security policy to identify the user. For example, if an NSF is aware of the IP (or MAC) address associated with the user, the NSF can use a pre-defined or dynamically learned name-address association to enforce the security functions for this given user (or user group);

  * Schedule: Time or time range when packet or flow is received;

  * Region: The geographic location where network traffic is received;

  * Target: The target indicates the entity to which the security services are applied. This can be a service, application, or device. A service is identified by the protocol type and/or port number. An application is a computer program for a specific task or purpose. It provides additional semantics (e.g., dependencies between services) for matching traffic. A device is a managed entity that is connected to the network. The attributes that can identify a device include type (e.g., router, switch, pc) and operating system (e.g., Windows, Linux, or Android), as well as the device's owner;

  * State: It refers to various states to which the network flow is associated. It can be either the TCP session state (e.g., new, established, related, invalid, or untracked), the session AAA state (e.g., authenticated but not authorized), or the access mode of the device (e.g., wireline, wireless, or cellular; these could be augmented with additional attributes, such as the type of VPN that is being used);

  * Direction: the direction of the network flow.

o Action: NSFs provide security functions by executing various Actions, which at least includes:

  ? Ingress actions, such as pass, drop, mirroring, etc;

? Egress actions, such as invoke signaling, tunnel encapsulation, packet forwarding and/or transformation;

? Applying a specific Functional Profile or signature - e.g., an IPS Profile, a signature file, an anti-virus file, or a URL filtering file. It is one of the key properties that determine the effectiveness of the NSF, and is mostly vendor-specific today. One goal of I2NSF is to standardize the form and functional interface of those security capabilities while supporting vendor-specific implementations of each. However, it is important to properly model the parts that are related to the action (what is enforced) and the conditions (on what it is enforced).

The above ECA ruleset is very general and easily extensible, thus can avoid any potential constraints which could limit the implementation of the network security control capability.

4. Information Model

An information model will be developed in order to describe in an abstract and vendor independent manner all the aspects related to the capabilities of NSFs. A detailed information model will be designed in the next versions of this draft as a consequence of the discussions, feedback, and extensions that will originate after the publication of this draft. In this version of the draft, we present a simplified view that only highlights the most important concepts.

The I2NSF capability interface is in charge of controlling and managing the NSFs by means of the information about the capabilities each NSF owns. This is done using the following approach:

1) User of the capability interface selects the set of capabilities required to meet the needs of the application;

2) A management entity uses the information model to match chosen capabilities to NSFs, independent of vendor;

3) A management entity takes the above information and creates or uses vendor-specific data models to install the NSFs identified by the chosen capabilities;

4) Control and monitoring can then begin.

This assumes that an external model, or set of models, is used to define the concept of an ECA Policy Rule and its components (e.g., Event, Condition, and Action objects).

Since Capabilities are unambiguous only within the same management system, and are not inherent characteristics that differentiate objects, it is also assumed that an external model (or set of models) will define a generic metadata concept.

The Capability is a general abstract concept. Currently, the most promising approach for defining the information model of the Capabilities uses the sub-classing to define non-overlapping and independent concepts. For instance, the Capability model that will be presented in the next sections that abstractly determines the security policies that a NSF can enforce does not overlap with an independent model that specifies how a NSF can be contacted by the controller (i.e., the protocols and secure channels) and when (i.e., the events to which it reacts). Capabilities are sub-classed from an appropriate class in the external metadata model.

The capability interface is used for advertising, creating, selecting and managing a set of specific security capabilities independent of the type and vendor of device that contains the NSF. That is, the user of the capability interface does not care whether the NSF is virtualized or hosted in a physical device, the vendor of the NSF, and which set of entities the NSF is communicating with (e.g., a firewall or an IPS). Instead, the user only cares about the set of capabilities that the NSF has, such as packet filtering or deep packet inspection. The overall structure is illustrated in the figure below:

```
+-------------------------+ 0..n        0..n +---------------+
|                         |/ \              \|   External    |
| External ECA Info Model + A ----------------+   Metadata    |
|                         |\ /  Aggregates   /|  Info Model   |
+-------------------------+      Metadata     +------+--------+
                                                    / \
                                                     |
                                                     |
                                                     |
                                                +----+-------+
                                                | Capability |
                                                | Sub-Model  |
                                                +------------+
```

Figure 1. The Overall I2NSF Information Model Design

This draft defines the Capability sub-Model shown in Figure 1. This model assumes that another, generic, information model for defining ECA policy rules (which includes a specific one for the CA part of ECA policy rules) exists outside of I2NSF.

It also assumes that Capabilities are modeled as metadata, since a Capability is something that describes and/or prescribes functionality about an object, but is not an inherent part of that object. Hence, the Security Capability Sub-Model extends the generic external metadata model.

Both of these external models could, but do not have to, draw from the SUPA model [I-D.draft-ietf-supa-generic-policy-info-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 I2NSF to reuse this generic model for different purposes.

It is assumed that the external ECA Information Model has the ability to aggregate metadata. Capabilities are then sub-classed from an appropriate class in the external Metadata Information Model; this enables the ECA objects to use the existing aggregation between them and Metadata to add Metadata to appropriate ECA objects.

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

5. Capabilities for security policy enforcement

At present, a variety of 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 are intended to describe the potentiality that Network Security Functions (NSFs) have for security policy enforcement purposes. Security Capabilities are abstract concepts that are independent of the actual security control that will implement them. However, every NSF will be associated to the capabilities it owns. Security Capabilities are required to allow differentiating among network functions. It would be a market enabler having a way to substitute a NSF with an equivalent one (i.e., having the same functionality). Moreover, Security Capabilities are very useful to reason about generic functions,

which may be needed at design time. That is, it is not needed to refer to a specific product when designing the network; rather the functions characterized by their capabilities are considered.

Therefore, we have developed another model where Security Capabilities determine what a security control can do in terms of conditions, actions, resolution strategies, external data, if it supports default action, etc. That is, Security Capabilities define without any ambiguity the actions a function can do in term of security policy enforcement. The Security Capability model is built on a predefined general policy model. The type of policies that a NSF can enforce is obtained by customizing the general policy model with the Security Capability information.

The Capability Model has been preliminarily validated by verifying that is allows the correct description of several existing security controls.

5.1. The CA Policy Model

The starting point of the design of our capability model is a simple observation. As human beings, we all understand immediately each other when we refer to security controls by just naming their category. For instance, experts agree on what is a NAT, a filtering control, or a VPN concentrator. Network security experts unequivocally refer to "packet filters" as stateless devices able to allow or deny packets forwarding based on conditions on source and destination IP addresses, source and destination ports, and IP protocol type fields [Alshaer]. Moreover, it is known that packet filter rules are prioritized and it is possible to specify a default action. More precisely, packet filters implement the First Matching Rule (FMR) or Last Matching Rule (LMR) resolution strategies.

However, we feel the need for more information in case of other devices, like stateful firewalls or application layer filters. These devices filter packets or communications, but there are differences among products in the packets and communications that they can categorize and the states they maintain. 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, apply anti-reply protections, and authenticate peers.

The purpose is to build a model of security capabilities that allow automatic management of virtualized systems, where intelligent components are able to properly identify and manage NSFs, and allow NSFs to properly declare their functionality so that they can be used in the correct way.

5.2. Geometric Model of CA Policies

We refer in this draft to the policy model defined in [Bas12] as geometric model, which is summarized here. Policies are specified by means of a set of rules in the "if condition then action" format [RFC3198]. Rules are formed by a condition clause and an action clause. This model can be further extended to support events, that is, in the Event-Condition-Action paradigms. However, for our purpose, the geometric model will only be used to define the CA part of the ECA model that we have selected as reference.

All the actions available to the security function are well known and organized in an action set A.

For filtering controls, the enforceable actions are either Allow and Deny, thus A={Allow,Deny}. For channel protection controls, they may be informally written as "enforce confidentiality", "enforce data authentication and integrity", and "enforce confidentiality and data authentication and integrity". However, these actions need to be instantiated to the technology used, for instance AH-transport mode and ESP-transport mode (and combinations thereof) are a more precise and univocal definition of channel protection actions.

Conditions are typed predicates concerning a given selector. A selector describes the values that a protocol field may take, e.g., the IP source selector is the set of all possible IP addresses, and it may also refer to the part of the packet where the values come from, e.g., the IP source selector refers to the IP source field in the IP header. Geometrically, a condition is the subset of its selector for which it evaluates to true. A condition on a given selector matches a packet if the value of the field referred to by the selector belongs to the condition. For instance, in Figure 2 the conditions are s1 <= S1 (read as s1 subset of or equal to S1) and s2 <= S2 (s1 of or equal to S2), both s1 and s2 match the packet x1, while only s2 matches x2.

```
       S2 ^ Destination port
          |
          |
          |      x2
          +......o
          |      .
          |      .
        --+.............+------------------------------------+
        | |      .      |                                    |
        s |      .      |                                    |
        e |      .      |            (rectangle)             |
        g |      .      |        condition clause (c)        |
        m |      .      |   here the action a is applied     |
        e |      .      |                                    |
        n |      .      |             x1=point=packet        |
        t +.............|.............o                      |
        | |      .      |             .                      |
        --+.............+------------------------------------+
          |      .      .             .                      .
          |      .      .             .                      .
          |      .      .             .                      .
          |      .      .             .                      .
          |      .      .             .                      .
          |      .      .             .                      .
    +------------+------+-------------+----------------------+------>
          |             |---- segment = condition in S1 -----|     S1
          +                                                 IP source
```

Figure 2: Geometric representation of a rule r=(c,a) that matches x1 but does not match x2.

To consider conditions in different selectors, the decision space is extended using the Cartesian product because distinct selectors refer to different fields, possibly from different protocol headers. Hence, given a policy-enabled element that allows the definition of conditions on the selectors S1, S2,..., Sm (where m is the number of Selectors available at the security control we want to model), its selection space is:

S=S1 X S2 X ... X Sm

To consider conditions in different selectors, the decision space is extended using the Cartesian product because distinct selectors refer to different fields, possibly from different protocol headers.

Accordingly, the condition clause c is a subset of S:

c = s1 X s2 X ... X sm <= S1 X S2 X ... X Sm = S

S represents the totality of the packets that are individually selectable by the security control to model when we use it to enforce a policy. Unfortunately, not all its subsets are valid condition clauses: only hyper-rectangles or union of hyper-rectangles (as they are Cartesian product of conditions) are valid. This is an intrinsic constraint of the policy languages as they specify rules by defining a condition for each selector. Languages that allow specification of conditions as relations over more fields are modeled by the geometric model as more complex geometric shapes determined by the equations. However, the algorithms to compute intersections are much more sophisticated than intersection hyper-rectangles. Figure 1 graphically represents a condition clause c in a two-dimensional selection space.

In the geometric model, a rule is expressed as r=(c,a), where c <= S (the condition clause is a subset of the selection space), and the action a belongs to A. Rule condition clauses matche a packet (rules matche a packet), if all the conditions forming the clauses match the packet: in Figure 1, the rule with condition clause c matches the packet x1 but not x2.

The rule set R is composed of n rules ri=(ci,ai).

The decision criteria for the action to apply when a packet matches two or more rules is abstracted by means of the resolution strategy RS: Pow(R) -> A, where Pow(R) is the power set of rules in R.

Formally, given a set of rules, the resolution strategy maps all the possible subsets of rules to an action a in A. When no rule matches a packet, the security controls may select the default action d in A, if they support one.

Resolution strategies may use, besides intrinsic rule data (i.e., condition clause and action clause), also ``external data'' associated to each rule, such as priority, identity of the creator, and creation time. Formally, every rule ri is associated by means of the function e(.) to:

```
e(ri) = (ri,f1(ri),f2(ri),...)
```

where E={fj:R -> Xj} (j=1,2,...) is the set that includes all the functions that map rules to external attributes in Xj. However, E, e, and all the Xj are determined by the resolution strategy used.

A policy is thus a function p: S -> A that connects each point of the selection space to an action taken from the action set A according to the rules in R. By also assuming RS(0)=d (where 0 is the empty-set) and RS(ri)=ai, the policy p can be described with this formula

```
p(x)=RS(match{R(x)}).
```

Therefore, in the geometric model, a policy is completely defined by the 4-tuple (R,RS,E,d): the rule set R, the resolution function RS, the set E of mappings to the external attributes, and the default action d.

Note that, the geometric model also supports ECA paradigms by simply modeling events like an additional selector.

5.3. Condition Types

After having analyzed the literature and the existing security controls, we have categorized the types of selectors in 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 a 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 conditions

```
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 instance

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

is not a valid condition, even if protocol types map to integers.

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 are well represented using a range-based selector (e.g., 1024-65535). As an example

```
source_port = 80

source_port < 1024

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

are valid conditions.

We include in the range-based selectors all 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 frequent at the application layer, where data are often represented as strings of text. The regex-based selector type also includes as sub-case the 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,

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

matches all the URLs that contain a domain, subdomain named website and the ones whose path contain the string .website.

```
MIME_type = video/*
```

matches all the MIME objects whose type is video.

Finally, we introduce the idea of custom check selectors. For instance the malware analysis looks for specific patterns and returns a Boolean value is an example of custom check selector, if

the logic of checking is not seen (nor really interesting) from the outside.

In order to be properly used by high-level policy based processed (like reasoning systems, refinement systems) these custom check selector need at least to be described as black-boxes, that is, the list of fields that they process (inputs) in order to return the Boolean verdict (output).

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

5.4. Model of Capabilities for Policy Specification and Enforcement Purposes

Our model of capabilities is based on actions and traffic classification features. Indeed, the need for enforcing one of the actions that a security control can apply to packets/flows is the main reason to use a security control. Moreover, security controls have classification features that permit the identification of the target packets/flows of the actions enforced, i.e., the selectors presented in Section 5.2. A security manager decides for a specific security control depending on the actions and classification features. If the security control can enforce the needed actions and has the classification features needed to identify the packets flows required by a policy, then the security control is capable of enforcing the policy. Our refinement model needs to know NSFs capabilities to perform its operations.

However, security controls may 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. We have ignored, to simplify this presentation, options to generate configurations that may have better performance, like the use of chains or ad hoc structures [Taylor]. Adding support to these forms of optimization is certainly feasible with a limited effort but it was outside the scope of this paper, that is, to show that adding security awareness to NFV management and orchestration features is possible. It is one of the task for future work.

Capabilities can be used for two purposes: describing generic security functions, and describing specific products. With the term generic security function (GNSF) we denote known classes of security functions. The idea is to have generic components whose behavior is as well understood as for the network components (i.e., a switch is

a switch and we know to use it even if it may have some vendor-specific functions). These generic functions can be substituted by any product that owns the required capability at instantiation time.

We have analyzed several classes of NSFs to prove the validity of our approach. We found the common features and defined a set of generic NSFs, including packet filter, URL filter, HTTP filter, VPN gateway, anti-virus, anti-malware, content filter, monitoring, anonymity proxy that will be described in a data model TBD.

Moreover, we have also categorized common extensions of the generic NSFs, packet filters that may decide based on time information. Moreover, some other packet filters add stateful features at ISO/OSI layer 4.

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

5.5. Algebra of capabilities

Our capabilities are defined by a 4-tuple, that is every NSF N will be associated to a 4-tuple (Ac; Cc; RSc; Dc) such that:

```
   (Ac; Cc; RSc; Dc) <= (XAC; XCC; XRSC; XDC)= K
```

where

- o XAC is the set of all the supported actions, that is, the set of all the actions supported by at least one of the existing NSFs;

- o Ac <= XAC is a set of actions that determine the actions actually available at the NSF N;

- o XCC is set of all the supported conditions types, that is, the set of all the conditions that can be to specify rules in at least one of the existing NSFs;

- o Cc <= XCC is the sef of conditions actually available at the NSF N;

- o XRSC is the set of all the existing resolutions strategies,

- o RSc <= XCC is the set of resolution strategies that can be used to specify solve conflicts of multiple matching rules at the NSF N;

- o XDC={F} U XAC, is the set of all the existing actions plus a dummy symbol F, a placeholder value that can be used to indicate that the default action can be freely selected by the policy editor; and

- o Dc <= XDC, Dc may be {F}, to indicate that the default action can be freely selected by the policy editor, thus it can vary in every policy, or an explicit action {a<=XAC} to indicate that the default action is fixed and the policy editor will not have the possibility to choice it.

Given cap1=(Ac1,Cc1,RSc1,def1) and cap2=(Ac2,Cc2,RSc2,def2), we define two operations that are useful to manipulate capabilities:

- o capability addition: cap1+cap2 = (Ac1 U Ac2, Cc1 U Cc2, RSc1,def1)

- o capability subtraction: cap_1-cap_2 = (Ac1\Ac2,Cc1\Cc2,RSc1,def1)

Note that addition and subtraction do not alter the resolution strategy and the default action method, as our main intent was to model addition of modules.

As an example, a generic packet filter that supports the first matching rule resolution strategies, allows the explicit specification of default actions and also supports time-based conditions. The description of its capabilities is the following:

```
Apf = {Allow, Deny}

Cpf= {IPsrc,IPdst,Psrc,Pdst,protType}

Ctime = {timestart,days,datestart,datestop}

cap_pf=(Apf; Cpf; {FMR}; F)

cap_pf+time=cap_pf + Ctime
```

By abuse of notation, we have written cap_pf+time=cap_pf + Ctime to shorten the more correct expression cap_pf+time=cap_pf +(;Ctime;;).

5.6. Examples of NSFs Categories

As an example of the application of the general capability model, we present in the next sections three examples of common categories: network security, content security, and attack mitigation.

5.6.1. Network Security

Network security is a category that describes the inspecting and processing of network traffic based on pre-defined security policies.

The inspecting portion may be thought of as a packet-processing engine that inspects packets traversing networks, either directly or in context to flows with which the packet is associated. From the perspective of packet-processing, implementations differ in the depths of packet headers and/or payloads they can inspect, the various flow and context states they can maintain, and the actions that can be applied to the packets or flows.

5.6.2. Content Security

Content security is another category of security capabilities applied to application layer. Through detecting the contents carried over the traffic in application layer, these capabilities can realize various security functions, such as defending against intrusion, inspecting virus, filtering malicious URL or junk email, blocking illegal web access or malicious data retrieval.

Generally, each type of threat in the application layer has a set of unique characteristics, and requires handling with a set of specific methods. Thus, these NSFs will be characterized by their own security capabilities.

5.6.3. Attack Mitigation

This category of security capabilities is used to detect and mitigate various types of network attacks. Today's common network attacks can be classified into the following sets, and each set further consists of a number of specific attacks:

o DDoS attacks:

?Network layer DDoS attacks: Examples include SYN flood, UDP flood, ICMP flood, IP fragment flood, IPv6 Routing header attack, and IPv6 duplicate address detection attack;

?Application layer DDoS attacks: Examples include http flood, https flood, cache-bypass http floods, WordPress XML RPC floods, ssl DDoS.

o Single-packet attack:

?Scanning and sniffing attacks: IP sweep, port scanning, etc

?malformed packet attacks: Ping of Death, Teardrop, etc

?special packet attacks: Oversized ICMP, Tracert, IP timestamp option packets, etc

Each type of network attack has its own network behaviors and packet/flow characteristics. Therefore, each type of attack needs a special security function, which is advertised as a capability, for detection and mitigation.

Overall, the implementation and management of this category of security capabilities of attack mitigation control is very similar to content security control. A standard interface, through which the security controller can choose and customize the given security capabilities according to specific requirements, is essential.

6. Information Sub-Model for Network Security Capabilities

The purpose of the Capability Framework Information Sub-Model is to define the concept of a Capability from an external metadata model, and enable Capabilities to be aggregated to appropriate objects. In the following sections we will present the cases of Network Security, Content Security, and Attack Mitigation sub-models.

6.1. Information Sub-Model for Network Security

The purpose of the Network Security Information Sub-Model is to define how network traffic is defined and determine if one or more network security features need to be applied to the traffic or not. Its basic structure is shown in the following figure:

```
                                       +---------------------+
     +---------------+                 |                     |
     |               |/ \             \| A Common Superclass |
     | ECAPolicyRule + A -------------+   for ECA Objects   |
     |               |\ /             /|                     |
     +-------+-------+                 +---------+-----------+
            / \                                 / \
             |                                   |
             |                                   |
   (subclasses to define Network          (subclasses of Event,
     Security ECA Policy Rules,       Condition, and Action Objects
      such as InspectTraffic)         for Network Security Control)
```

Figure 3. Network Security Information Sub-Model Overview

In the above figure, the ECAPolicyRule, along with the Event, Condition, and Action Objects, are defined in the external ECA Info Model. The Network Security Sub-Model extends both to define security-specific ECA policy rules, as well as Events, Conditions, and Actions.
An I2NSF Policy Rule is a special type of Policy Rule that is in event-condition-action (ECA) form. It consists of the Policy Rule, components of a Policy Rule (e.g., events, conditions, and actions), and optionally, metadata. It can be applied to both uni-directional and bi-directional traffic across the NSF.

Each rule is triggered by one or more events. If the set of events evaluates to true, then a set of conditions are evaluated and, if true, enable a set of actions to be executed.

An example of an I2NSF Policy Rule is, in pseudo-code:

```
IF <event-clause> is TRUE
   IF <condition-clause> is TRUE
      THEN execute <action-clause>
   END-IF
END-IF
```

In the above example, the Event, Condition, and Action portions of a Policy Rule are all **Boolean Clauses**.

6.1.1. Network Security Policy Rule Extensions

Figure 3 shows an example of more detailed design of the ECA Policy Rule subclasses that are contained in the Network Security Information Sub-Model, which just illustrates how more specific Network Security Policies are inherited and extended from the

SecurityECAPolicyRule class. Any new kinds of specific Network Security Policy can be created by following the same pattern of class design as below.

```
                             +---------------+
                             |    External   |
                             | ECAPolicyRule |
                             +-------+-------+
                                    / \
                                     |
                                     |
                        +------------+----------+
                        | SecurityECAPolicyRule |
                        +------------+----------+
                                     |
                                     |
           +----+-----+--------+-----+----+---------+---------+--- ...
           |          |        |          |         |         |
           |          |        |          |         |         |
    +------+-------+  |  +-----+-------+  |  +------+------+  |
    |Authentication|  |  |  Accounting |  |  |ApplyProfile |  |
    |ECAPolicyRule |  |  |ECAPolicyRule|  |  |ECAPolicyRule|  |
    +--------------+  |  +-------------+  |  +-------------+  |
                      |                   |                   |
               +------+------+     +------+------+     +--------------+
               |Authorization|     |   Traffic   |     |ApplySignature|
               |ECAPolicyRule|     | Inspection  |     |ECAPolicyRule |
               +-------------+     |ECAPolicyRule|     +--------------+
                                   +-------------+
```

Figure 4. Network Security Info Sub-Model ECAPolicyRule Extensions

The SecurityECAPolicyRule is the top of the I2NSF ECA Policy Rule hierarchy. It inherits from the (external) generic ECA Policy Rule to define Security ECA Policy Rules. The SecurityECAPolicyRule contains all of the attributes, methods, and relationships defined in its superclass, and adds additional concepts that are required for Network Security (these will be defined in the next version of this draft). The six SecurityECAPolicyRule subclasses extend the SecurityECAPolicyRule class to represent six different types of Network Security ECA Policy Rules. It is assumed that the (external) generic ECAPolicyRule class defines basic information in the form of attributes, such as an unique object ID, as well as a description and other basic, but necessary, information.

It is assumed that the (external) generic ECA Policy Rule is abstract; the SecurityECAPolicyRule is also abstract. This enables data model optimizations to be made while making this information model detailed but flexible and extensible.

The SecurityECAPolicyRule defines network security policy as a container that aggregates Event, Condition, and Action objects, which are described in Section 6.1.3, 6.1.4, and 6.1.5, respectively. Events, Conditions, and Actions can be generic or security-specific. Section 4.6 defines the concept of default security Actions.

Brief class descriptions of these six ECA Policy Rules are provided in the Appendix A.

6.1.2. Network Security Policy Rule Operation

Network security policy consists of a number of more granular ECA Policy Rules formed from the information model described above. In simpler cases, where the Event and Condition clauses remain unchanged, then network security control may be performed by calling additional network security actions. Network security policy examines and performs basic processing of the traffic as follows:

1. For a given SecurityECAPolicyRule (which can be generic or specific to security, such as those in Figure 3), the NSF evaluates the Event clause. It may use security Event objects to do all or part of this evaluation, which are defined in section 4.3.3. If the Event clause evaluates to TRUE, then the Condition clause of this SecurityECAPolicyRule is evaluated; otherwise, execution of this SecurityECAPolicyRule is stopped, and the next SecurityECAPolicyRule (if one exists) is evaluated;

2. The Condition clause is then evaluated. It may use security Condition objects to do all or part of this evaluation, which are defined in section 4.3.4. If the Condition clause evaluates to TRUE, then the set of Actions in this SecurityECAPolicyRule MUST be executed. This is defined as "matching" the SecurityECAPolicyRule; otherwise, execution of this SecurityECAPolicyRule is stopped, and the next SecurityECAPolicyRule (if one exists) is evaluated;

3. If none of the SecurityECAPolicyRules are matched, then the NSF denies the traffic by default;

4. If the traffic matches a rule, the NSF performs the defined Actions on the traffic. It may use security Action objects to do all or part of this execution, which are defined in section 4.3.5. If the action is "deny", the NSF blocks the traffic. If the basic action is permit or mirror, the NSF firstly performs that function, and then checks whether certain other security capabilities are referenced in the rule. If yes, go to step 5. If no, the traffic is permitted;

5. If other security capabilities (e.g., Anti-virus or IPS) are referenced in the SecurityECAPolicyRule, and the Action defined in the rule is permit or mirror, the NSF performs the referenced security capabilities.

Metadata attached to the SecurityECAPolicyRule MAY be used to control how the SecurityECAPolicyRule is evaluated. This is called a Policy Rule Evaluation Strategy. For example, one strategy is to match and execute the first SecurityECAPolicyRule, and then exit without executing any other SecurityECAPolicyRules (even if they matched). In contrast, a second strategy is to first collect all SecurityECAPolicyRules that matched, and then execute them according to a pre-defined order (e.g., the priority of each SecurityECAPolicyRule).

One policy or rule can be applied multiple times to different managed objects (e.g., links, devices, networks, VPNS). This not only guarantees consistent policy enforcement, but also decreases the configuration workload.

6.1.3. Network Security Event Sub-Model

Figure 10 shows a more detailed design of the Event subclasses that are contained in the Network Security Information Sub-Model.

```
                                     +---------------------+
        +---------------+ 1..n   1..n|                     |
        |               |/ \        \| A Common Superclass |
        | ECAPolicyRule + A ---------+   for ECA Objects   |
        |               |\ /        /|                     |
        +---------------+            +-----------+---------+
                                                / \
                                                 |
                                                 |
                         +--------------+--------+------+
                         |              |               |
                         |              |               |
                   +-----+----+  +------+------+  +-----+-----+
                   | An Event |  | A Condition |  | An Action |
                   |   Class  |  |    Class    |  |   Class   |
                   +-----+----+  +-------------+  +-----------+
                        / \
                         |
                         |
                         |
             +-----------+---+----------------+--------------+-- ...
             |               |                |              |
             |               |                |              |
     +-------+----+ +--------+-----+ +--------+-----+ +------+-----+
     |UserSecurity| |    Device    | |    System    | |TimeSecurity|
     |   Event    | | SecurityEvent| | SecurityEvent| |     Event  |
     +------------+ +--------------+ +--------------+ +------------+
```

Figure 5. Network Security Info Sub-Model Event Class Extensions

The four Event classes shown in Figure 5 extend the (external) generic Event class to represent Events that are of interest to Network Security. It is assumed that the (external) generic Event class defines basic Event information in the form of attributes, such as a unique event ID, a description, as well as the date and time that the event occurred.

The following are assumptions that define the functionality of the generic Event class. If desired, these could be defined as attributes in a SecurityEvent class (which would be a subclass of the generic Event class, and a superclass of the four Event classes shown in Figure 10). However, this makes it harder to use any generic Event model with the I2NSF events. Assumptions are:

- The generic Event class is abstract
- All four SecurityEvent subclasses are concrete
- The generic Event class uses the composite pattern, so individual Events as well as hierarchies of Events are available (the four subclasses in Figure 10 would be subclasses of the Atomic Event)
- The generic Event class has a mechanism to uniquely identify the source of the Event
- The generic Event class has a mechanism to separate header information from its payload
- The generic Event class has a mechanism to attach zero or more metadata objects to it

Brief class descriptions are provided in the following sub-sections.

6.1.3.1. UserSecurityEvent Class Description

The purpose of this class is to represent Events that are initiated by a user, such as logon and logoff Events. Information in this Event may be used as part of a test to determine if the Condition clause in this ECA Policy Rule should be evaluated or not. Examples include user identification data and the type of connection used by the user.

The UserSecurityEvent class defines the following attributes:

6.1.3.1.1. The usrSecEventContent Attribute

This is a mandatory string that contains the content of the UserSecurityEvent. The format of the content is specified in the usrSecEventFormat class attribute, and the type of Event is defined in the usrSecEventType class attribute. An example of the usrSecEventContent attribute is the string "hrAdmin", with the usrSecEventFormat set to 1 (GUID) and the usrSecEventType attribute set to 5 (new logon).

6.1.3.1.2. The usrSecEventFormat Attribute

This is a mandatory non-negative enumerated integer, which is used to specify the data type of the usrSecEventContent attribute. The content is specified in the usrSecEventContent class attribute, and the type of Event is defined in the usrSecEventType class attribute. An example of the usrSecEventContent attribute is the string "hrAdmin", with the usrSecEventFormat attribute set to 1 (GUID) and the usrSecEventType attribute set to 5 (new logon). Values include:

0: unknown
1: GUID (Generic Unique IDentifier)
2: UUID (Universal Unique IDentifier)
3: URI (Uniform Resource Identifier)
4: FQDN (Fully Qualified Domain Name)
5: FQPN (Fully Qualified Path Name)

6.1.3.1.3. The usrSecEventType Attribute

This is a mandatory non-negative enumerated integer, which is used to specify the type of Event that involves this user. The content and format are specified in the usrSecEventContent and usrSecEventFormat class attributes, respectively. An example of the usrSecEventContent attribute is the string "hrAdmin", with the usrSecEventFormat attribute set to 1 (GUID) and the usrSecEventType attribute set to 5 (new logon). Values include:

0: unknown
1: new user created
2: new user group created
3: user deleted
4: user group deleted
5: user logon
6: user logoff
7: user access request
8: user access granted
9: user access violation

6.1.3.2. DeviceSecurityEvent Class Description

The purpose of a DeviceSecurityEvent is to represent Events that provide information from the Device that are important to I2NSF Security. Information in this Event may be used as part of a test to determine if the Condition clause in this ECA Policy Rule should be evaluated or not. Examples include alarms and various device statistics (e.g., a type of threshold that was exceeded), which may signal the need for further action.

The DeviceSecurityEvent class defines the following attributes:

6.1.3.2.1. The devSecEventContent Attribute

This is a mandatory string that contains the content of the DeviceSecurityEvent. The format of the content is specified in the devSecEventFormat class attribute, and the type of Event is defined

in the devSecEventType class attribute. An example of the devSecEventContent attribute is "alarm", with the devSecEventFormat attribute set to 1 (GUID), the devSecEventType attribute set to 5 (new logon).

6.1.3.2.2. The devSecEventFormat Attribute

This is a mandatory non-negative enumerated integer, which is used to specify the data type of the devSecEventContent attribute. Values include:

0:	unknown
1:	GUID (Generic Unique IDentifier)
2:	UUID (Universal Unique IDentifier)
3:	URI (Uniform Resource Identifier)
4:	FQDN (Fully Qualified Domain Name)
5:	FQPN (Fully Qualified Path Name)

6.1.3.2.3. The devSecEventType Attribute

This is a mandatory non-negative enumerated integer, which is used to specify the type of Event that was generated by this device. Values include:

0:	unknown
1:	communications alarm
2:	quality of service alarm
3:	processing error alarm
4:	equipment error alarm
5:	environmental error alarm

Values 1-5 are defined in X.733. Additional types of errors may also be defined.

6.1.3.2.4. The devSecEventTypeInfo[0..n] Attribute

This is an optional array of strings, which is used to provide additional information describing the specifics of the Event generated by this Device. For example, this attribute could contain probable cause information in the first array, trend information in the second array, proposed repair actions in the third array, and additional information in the fourth array.

6.1.3.2.5. The devSecEventTypeSeverity Attribute

This is a mandatory non-negative enumerated integer, which is used to specify the perceived severity of the Event generated by this Device. Values include:

0:	unknown
1:	cleared
2:	indeterminate
3:	critical
4:	major
5:	minor
6:	warning

Values 1-6 are from X.733.

6.1.3.3. SystemSecurityEvent Class Description

The purpose of a SystemSecurityEvent is to represent Events that are detected by the management system, instead of Events that are generated by a user or a device. Information in this Event may be used as part of a test to determine if the Condition clause in this ECA Policy Rule should be evaluated or not. Examples include an event issued by an analytics system that warns against a particular pattern of unknown user accesses, or an Event issued by a management system that represents a set of correlated and/or filtered Events.

The SystemSecurityEvent class defines the following attributes:

6.1.3.3.1. The sysSecEventContent Attribute

This is a mandatory string that contains the content of the SystemSecurityEvent. The format of the content is specified in the sysSecEventFormat class attribute, and the type of Event is defined in the sysSecEventType class attribute. An example of the sysSecEventContent attribute is the string "sysadmin3", with the sysSecEventFormat attribute set to 1 (GUID), and the sysSecEventType attribute set to 2 (audit log cleared).

6.1.3.3.2. The sysSecEventFormat Attribute

This is a mandatory non-negative enumerated integer, which is used to specify the data type of the sysSecEventContent attribute. Values include:

0: unknown
1: GUID (Generic Unique IDentifier)
2: UUID (Universal Unique IDentifier)
3: URI (Uniform Resource Identifier)
4: FQDN (Fully Qualified Domain Name)
5: FQPN (Fully Qualified Path Name)

6.1.3.3.3. The sysSecEventType Attribute

This is a mandatory non-negative enumerated integer, which is used to specify the type of Event that involves this device. Values include:

0: unknown
1: audit log written to
2: audit log cleared
3: policy created
4: policy edited
5: policy deleted
6: policy executed

6.1.3.4. TimeSecurityEvent Class Description

The purpose of a TimeSecurityEvent is to represent Events that are temporal in nature (e.g., the start or end of a period of time). Time events signify an individual occurrence, or a time period, in which a significant event happened. Information in this Event may be used as part of a test to determine if the Condition clause in this ECA Policy Rule should be evaluated or not. Examples include issuing an Event at a specific time to indicate that a particular resource should not be accessed, or that different authentication and authorization mechanisms should now be used (e.g., because it is now past regular business hours).

The TimeSecurityEvent class defines the following attributes:

6.1.3.4.1. The timeSecEventPeriodBegin Attribute

This is a mandatory DateTime attribute, and represents the beginning of a time period. It has a value that has a date and/or a time component (as in the Java or Python libraries).

6.1.3.4.2. The timeSecEventPeriodEnd Attribute

This is a mandatory DateTime attribute, and represents the end of a time period. It has a value that has a date and/or a time component (as in the Java or Python libraries). If this is a single Event occurence, and not a time period when the Event can occur, then the timeSecEventPeriodEnd attribute may be ignored.

6.1.3.4.3. The timeSecEventTimeZone Attribute

This is a mandatory string attribute, and defines the time zone that this Event occurred in using the format specified in ISO8601.

6.1.4. Network Security Condition Sub-Model

Figure 6 shows a more detailed design of the Condition subclasses that are contained in the Network Security Information Sub-Model.

```
                                     +---------------------+
        +---------------+ 1..n  1..n |                     |
        |               |/ \        \| A Common Superclass |
        | ECAPolicyRule+ A ----------+   for ECA Objects   |
        |               |\ /        /|                     |
        +-------+-------+            +-----------+---------+
                                                / \
                                                 |
                                                 |
                       +--------------+----------+----+
                       |              |               |
                       |              |               |
                 +-----+----+  +------+------+  +-----+-----+
                 | An Event |  | A Condition |  | An Action |
                 |   Class  |  |    Class    |  |   Class   |
                 +----------+  +------+------+  +-----------+
                                     / \
                                      |
                                      |
           +--------+----------+------+---+---------+--------+--- ...
           |        |          |          |         |        |
           |        |          |          |         |        |
     +-----+-----+  |  +-------+-------+  |  +------+------+  |
     |   Packet  |  |  | PacketPayload |  |  |    Target   |  |
     |  Security |  |  |    Security   |  |  |  Security   |  |
     | Condition |  |  |   Condition   |  |  |  Condition  |  |
     +-----------+  |  +---------------+  |  +-------------+  |
                    |                     |                   |
             +------+-------+  +----------+------+  +--------+-------+
             | UserSecurity |  | SecurityContext |  | GenericContext |
             |   Condition  |  |    Condition    |  |    Condition   |
             +--------------+  +-----------------+  +----------------+
```

Figure 6. Network Security Info Sub-Model Condition Class Extensions

The six Condition classes shown in Figure 6 extend the (external) generic Condition class to represent Conditions that are of interest to Network Security. It is assumed that the (external) generic Condition class is abstract, so that data model optimizations may be defined. It is also assumed that the generic Condition class defines basic Condition information in the form of attributes, such as a unique object ID, a description, as well as a mechanism to attach zero or more metadata objects to it. While this could be defined as attributes in a SecurityCondition class (which would be a subclass

of the generic Condition class, and a superclass of the six Condition classes shown in Figure 11), this makes it harder to use any generic Condition model with the I2NSF conditions.

Brief class descriptions are provided in the following sub-sections.

6.1.4.1. PacketSecurityCondition

The purpose of this Class is to represent packet header information that can be used as part of a test to determine if the set of Policy Actions in this ECA Policy Rule should be executed or not. This class is abstract, and serves as the superclass of more detailed conditions that involve different types of packet formats. Its subclasses are shown in Figure 7, and are defined in the following sections.

```
                       +-------------------------+
                       | PacketSecurityCondition |
                       +------------+------------+
                                   / \
                                    |
                                    |
           +---------+----------+---+-----+----------+
           |         |          |         |          |
           |         |          |         |          |
  +--------+-------+ | +--------+-------+ | +--------+-------+
  | PacketSecurity | | | PacketSecurity | | | PacketSecurity |
  |  MACCondition  | | | IPv4Condition  | | | IPv6Condition  |
  +----------------+ | +----------------+ | +----------------+
                     |                    |
            +--------+-------+   +--------+-------+
            |  TCPCondition  |   |  UDPCondition  |
            +----------------+   +----------------+
```

Figure 7. Network Security Info Sub-Model PacketSecurityCondition Class Extensions

6.1.4.1.1. PacketSecurityMACCondition

The purpose of this Class is to represent packet MAC packet header information that can be used as part of a test to determine if the set of Policy Actions in this ECA Policy Rule should be executed or not. This class is concrete, and defines the following attributes:

6.1.4.1.1.1. The pktSecCondMACDest Attribute

This is a mandatory string attribute, and defines the MAC destination address (6 octets long).

6.1.4.1.1.2. The pktSecCondMACSrc Attribute

This is a mandatory string attribute, and defines the MAC source address (6 octets long).

6.1.4.1.1.3. The pktSecCondMAC8021Q Attribute

This is an optional string attribute, and defines the 802.1Q tag value (2 octets long). This defines VLAN membership and 802.1p priority values.

6.1.4.1.1.4. The pktSecCondMACEtherType Attribute

This is a mandatory string attribute, and defines the EtherType field (2 octets long). Values up to and including 1500 indicate the size of the payload in octets; values of 1536 and above define which protocol is encapsulated in the payload of the frame.

6.1.4.1.1.5. The pktSecCondMACTCI Attribute

This is an optional string attribute, and defines the Tag Control Information. This consists of a 3 bit user priority field, a drop eligible indicator (1 bit), and a VLAN identifier (12 bits).

6.1.4.1.2. PacketSecurityIPv4Condition

The purpose of this Class is to represent packet IPv4 packet header information that can be used as part of a test to determine if the set of Policy Actions in this ECA Policy Rule should be executed or not. This class is concrete, and defines the following attributes:

6.1.4.1.2.1. The pktSecCondIPv4SrcAddr Attribute

This is a mandatory string attribute, and defines the IPv4 Source Address (32 bits).

6.1.4.1.2.2. The pktSecCondIPv4DestAddr Attribute

This is a mandatory string attribute, and defines the IPv4 Destination Address (32 bits).

6.1.4.1.2.3. The pktSecCondIPv4ProtocolUsed Attribute

This is a mandatory string attribute, and defines the protocol used in the data portion of the IP datagram (8 bits).

6.1.4.1.2.4. The pktSecCondIPv4DSCP Attribute

This is a mandatory string attribute, and defines the Differentiated Services Code Point field (6 bits).

6.1.4.1.2.5. The pktSecCondIPv4ECN Attribute

This is an optional string attribute, and defines the Explicit Congestion Notification field (2 bits).

6.1.4.1.2.6. The pktSecCondIPv4TotalLength Attribute

This is a mandatory string attribute, and defines the total length of the packet (including header and data) in bytes (16 bits).

6.1.4.1.2.7. The pktSecCondIPv4TTL Attribute

This is a mandatory string attribute, and defines the Time To Live in seconds (8 bits).

6.1.4.1.3. PacketSecurityIPv6Condition

The purpose of this Class is to represent packet IPv6 packet header information that can be used as part of a test to determine if the set of Policy Actions in this ECA Policy Rule should be executed or not. This class is concrete, and defines the following attributes:

6.1.4.1.3.1. The pktSecCondIPv6SrcAddr Attribute

This is a mandatory string attribute, and defines the IPv6 Source Address (128 bits).

6.1.4.1.3.2. The pktSecCondIPv6DestAddr Attribute

This is a mandatory string attribute, and defines the IPv6 Destination Address (128 bits).

6.1.4.1.3.3. The pktSecCondIPv6DSCP Attribute

This is a mandatory string attribute, and defines the Differentiated Services Code Point field (6 bits). It consists of the six most significant bits of the Traffic Class field in the IPv6 header.

6.1.4.1.3.4. The pktSecCondIPv6ECN Attribute

This is a mandatory string attribute, and defines the Explicit Congestion Notification field (2 bits). It consists of the two least significant bits of the Traffic Class field in the IPv6 header.

6.1.4.1.3.5. The pktSecCondIPv6FlowLabel Attribute

This is a mandatory string attribute, and defines an IPv6 flow label. This, in combination with the Source and Destination Address fields, enables efficient IPv6 flow classification by using only the IPv6 main header fields (20 bits).

6.1.4.1.3.6. The pktSecCondIPv6PayloadLength Attribute

This is a mandatory string attribute, and defines the total length of the packet (including the fixed and any extension headers, and data) in bytes (16 bits).

6.1.4.1.3.7. The pktSecCondIPv6NextHeader Attribute

This is a mandatory string attribute, and defines the type of the next header (e.g., which extension header to use) (8 bits).

6.1.4.1.3.8. The pktSecCondIPv6HopLimit Attribute

This is a mandatory string attribute, and defines the maximum number of hops that this packet can traverse (8 bits).

6.1.4.1.4. PacketSecurityTCPCondition

The purpose of this Class is to represent packet TCP packet header information that can be used as part of a test to determine if the set of Policy Actions in this ECA Policy Rule should be executed or not. This class is concrete, and defines the following attributes:

6.1.4.1.4.1. The pktSecCondTPCSrcPort Attribute

This is a mandatory string attribute, and defines the Source Port (16 bits).

6.1.4.1.4.2. The pktSecCondTPCDestPort Attribute

This is a mandatory string attribute, and defines the Destination Port (16 bits).

6.1.4.1.4.3. The pktSecCondTPCSeqNum Attribute

This is a mandatory string attribute, and defines the sequence number (32 bits).

6.1.4.1.4.4. The pktSecCondTPCFlags Attribute

This is a mandatory string attribute, and defines the nine Control bit flags (9 bits).

6.1.4.1.5. PacketSecurityUDPCondition

The purpose of this Class is to represent packet UDP packet header information that can be used as part of a test to determine if the set of Policy Actions in this ECA Policy Rule should be executed or not. This class is concrete, and defines the following attributes:

6.1.4.1.5.1. The pktSecCondUDPSrcPort Attribute

This is a mandatory string attribute, and defines the UDP Source Port (16 bits).

6.1.4.1.5.2. The pktSecCondUDPDestPort Attribute

This is a mandatory string attribute, and defines the UDP Destination Port (16 bits).

6.1.4.1.5.3. The pktSecCondUDPLength Attribute

This is a mandatory string attribute, and defines the length in bytes of the UDP header and data (16 bits).

6.1.4.2. PacketPayloadSecurityCondition

The purpose of this Class is to represent packet payload data that can be used as part of a test to determine if the set of Policy Actions in this ECA Policy Rule should be executed or not. Examples include a specific set of bytes in the packet payload.

6.1.4.3. TargetSecurityCondition

The purpose of this Class is to represent information about different targets of this policy (i.e., entities to which this policy rule should be applied), which can be used as part of a test to determine if the set of Policy Actions in this ECA Policy Rule should be executed or not. Examples include whether the targeted entities are playing the same role, or whether each device is administered by the same set of users, groups, or roles.

This Class has several important subclasses, including:

a. ServiceSecurityContextCondition is the superclass for all information that can be used in an ECA Policy Rule that specifies data about the type of service to be analyzed (e.g., the protocol type and port number)

b. ApplicationSecurityContextCondition is the superclass for all information that can be used in a ECA Policy Rule that specifies data that identifies a particular application (including metadata, such as risk level)

c. DeviceSecurityContextCondition is the superclass for all information that can be used in a ECA Policy Rule that specifies data about a device type and/or device OS that is being used

6.1.4.4. UserSecurityCondition

The purpose of this Class is to represent data about the user or group referenced in this ECA Policy Rule that can be used as part of a test to determine if the set of Policy Actions in this ECA Policy Rule should be evaluated or not. Examples include the user or group id used, the type of connection used, whether a given user or group is playing a particular role, or whether a given user or group has failed to login a particular number of times.

6.1.4.5. SecurityContextCondition

The purpose of this Class is to represent security conditions that are part of a specific context, which can be used as part of a test to determine if the set of Policy Actions in this ECA Policy Rule should be evaluated or not. Examples include testing to determine if a particular pattern of security-related data have occurred, or if the current session state matches the expected session state.

6.1.4.6. GenericContextSecurityCondition

The purpose of this Class is to represent generic contextual information in which this ECA Policy Rule is being executed, which can be used as part of a test to determine if the set of Policy Actions in this ECA Policy Rule should be evaluated or not. Examples include geographic location and temporal information.

6.1.5. Network Security Action Sub-Model

Figure 7 shows a more detailed design of the Action subclasses that are contained in the Network Security Information Sub-Model.

```
                                     +---------------------+
        +---------------+ 1..n  1..n |                     |
        |               |/ \        \| A Common Superclass |
        | ECAPolicyRule+ A ----------+   for ECA Objects   |
        |               |\ /        /|                     |
        +---------------+            +-----------+---------+
                                                / \
                                                 |
                                                 |
                         +--------------+--------+------+
                         |              |               |
                         |              |               |
                   +-----+----+  +------+------+  +-----+-----+
                   | An Event |  | A Condition |  | An Action |
                   |   Class  |  |    Class    |  |   Class   |
                   +----------+  +-------------+  +-----+-----+
                                                       / \
                                                        |
                                                        |
          +------------+-------------+------------------+-------- ...
          |            |             |                  |
          |            |             |                  |
     +----+----+  +----+---+  +------+-------+  +-------+--------+
     | Ingress |  | Egress |  | ApplyProfile |  | ApplySignature |
     | Action  |  | Action |  |    Action    |  |     Action     |
     +---------+  +--------+  +--------------+  +----------------+
```

Figure 7. Network Security Info Sub-Model Action Extensions

The four Action classes shown in Figure 7 extend the (external) generic Action class to represent Actions that perform a Network Security Control function. Brief class descriptions are provided in the following sub-sections.

6.1.5.1. IngressAction

The purpose of this Class is to represent actions performed on packets that enter an NSF. Examples include pass, drop, mirror traffic.

6.1.5.2. EgressAction

The purpose of this Class is to represent actions performed on packets that exit an NSF. Examples include pass, drop, mirror traffic, signal, encapsulate.

6.1.5.3. ApplyProfileAction

The purpose of this Class is to represent applying a profile to packets to perform content security and/or attack mitigation control.

6.1.5.4. ApplySignatureAction

The purpose of this Class is to represent applying a signature file to packets to perform content security and/or attack mitigation control.

6.2. Information Model for Content Security Control

The block for content security control is composed of a number of security capabilities, while each one aims for protecting against a specific type of threat in the application layer.

Following figure shows a basic structure of the information model:

```
                   +-----------------------------------+
                   |                                   |
                   |                                   |
                   |     Anti-Virus                    |
                   |     Intrusion Prevention          |
                   |     URL Filtering                 |
                   |     File Blocking                 |
                   |     Data Filtering                |
                   |     Application Behavior Control  |
                   |     Mail Filtering                |
                   |     Packet Capturing              |
                   |     File Isolation                |
                   |     ...                           |
                   |                                   |
                   |                                   |
                   |                                   |
                   |                                   |
                   |              Information model    |
                   |              for content security |
                   |              control              |
                   +-----------------------------------+
```

Figure 8. The basic structure of information model for content security control

The detailed description about the standard interface and the parameters for all the security capabilities of this category are TBD.

6.3. Information Model for Attack Mitigation Control

The block for attack mitigation control is composed of a number of security capabilities, while each one aims for mitigating a specific type of network attack.

Following figure shows a basic structure of the information model:

Please view in a fixed-width font such as Courier.

```
+---------------------------------------------------+
|                                                   |
| +---------------------+    +---------------+     |
| |Attack mitigation    |    | General Shared|     |
| |capabilites:         |    | Parameters:   |     |
| | SYN flood,          |    |               |     |
| | UDP flood,          |    |               |     |
| | ICMP flood,         |    |               |     |
| | IP fragment flood,  |    |               |     |
| | IPv6 related attacks|    |               |     |
| | HTTP flood,         |    |               |     |
| | HTTPS flood,        |    |               |     |
| | DNS flood,          |    |               |     |
| | DNS amplification,  |    |               |     |
| | SSL DDoS,           |    |               |     |
| | IP sweep,           |    |               |     |
| | Port scanning,      |    |               |     |
| | Ping of Death,      |    |               |     |
| | Oversized ICMP      |    |               |     |
| |                     |    |               |     |
| | ...                 |    |               |     |
| |                     |    |               |     |
| +---------------------+    +---------------+     |
|                                                   |
|                            Information model      |
|                            for attack mitigation|
|                            control                |
+---------------------------------------------------+
```

Figure 9. The basic structure of information model for attack mitigation control

The detailed description about the standard interface and the general shared parameters for all the security capabilities of this category are TBD.

7. Security Considerations

The security capability policy information sent to NSFs should be protected by the secure communication channel, to ensure the confidentiality and integrity. In another side, the NSFs and security controller can all be faked, which lead to undesirable results, i.e., security policy leakage from security controller, faked security controller sending false information to mislead the NSFs. The mutual authentication is essential to protected against

this kind of attack. The current mainstream security technologies (i.e., TLS, DTLS, IPSEC, X.509 PKI) can be employed appropriately to provide the above security functionalities.

In addition, to defend against the DDoS attack caused by the security controller sending too much configuration messages to the NSFs, the rate limiting or similar mechanisms should be considered in NSF, whether in advance or just in the process of DDoS attack.

8. IANA Considerations

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

9. References

9.1. 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>.

9.2. 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-problem-and-use-cases] Hares, S., et.al., "I2NSF Problem Statement and Use cases", Work in Progress, February, 2016.

[I-D.draft-ietf-i2nsf-framework] Lopez, E., et.al., "Framework for Interface to Network Security Functions", Work in Progress, October, 2016.

[I-D.draft-ietf-i2nsf-terminology] Hares, S., et.al., "Interface to Network Security Functions (I2NSF) Terminology", Work in Progress, April, 2016

[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, June, 2016.

[I-D.draft-baspez-i2nsf-capabilities-00] Basile C., Lopez D. R., "A Model of Security Capabilities for Network Security Functions", July 2016

[I-D.draft-xia-i2nsf-capability-interface-im-06] Xia L., et al., "Information Model of Interface to Network Security Functions Capability Interface", June 2016

[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.

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

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

10. Acknowledgments

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

Appendix A.

Six exemplary policy rules of Network Security Capability are introduced in this Appendix to clarify how to create different kinds of specific ECA policy rules.

Note that there is a common pattern that defines how these ECAPolicyRules operate; this simplifies their implementation. All of these six ECA Policy Rules are concrete classes.

In addition, none of these six subclasses define attributes. This enables them to be viewed as simple object containers, and hence, applicable to a wide variety of content. It also means that the content of the function (e.g., how an entity is authenticated, what specific traffic is inspected, or which particular signature is applied) is defined solely by the set of events, conditions, and actions that are contained by the particular subclass. This enables the policy rule, with its aggregated set of events, conditions, and actions, to be treated as a reusable object.

A.1. AuthenticationECAPolicyRule Class Definition

The purpose of an AuthenticationECAPolicyRule is to define an ECA Policy Rule that can verify whether an entity has an attribute of a specific value.

This class does NOT define the authentication method used. This is because this would effectively "enclose" this information within the AuthenticationECAPolicyRule. This has two drawbacks. First, other entities that need to use information from the Authentication class(es) could not; they would have to associate with the AuthenticationECAPolicyRule class, and those other classes would not likely be interested in the AuthenticationECAPolicyRule. Second, the evolution of new authentication methods should be independent of the AuthenticationECAPolicyRule; this cannot happen if the Authentication class(es) are embedded in the AuthenticationECAPolicyRule. Hence, this document recommends the following design:

```
                                                     +-----------------+
    +----------------+ 1..n                    1...n |                 |
    |                |/ \  HasAuthenticationMethod  \| Authentication  |
    | Authentication + A ----------+-----------------+     Method      |
    | ECAPolicyRule  |\ /          ^                /|                 |
    |                |             |                 +-----------------+
    +----------------+             |
```

```
                                     |
                        +------------+------------+
                        | AuthenticationRuleDetail |
                        +------------+------------+
                                    / \ 0..n
                                     |
                                     | PolicyControlsAuthentication
                                     |
                                    / \
                                     A
                                    \ / 0..n
                          +----------+-------------+
                          | ManagementECAPolicyRule |
                          +-------------------------+
```

Figure 10. Modeling Authentication Mechanisms

This document only defines the AuthenticationECAPolicyRule; all other classes, and the aggregations, are defined in an external model. For completeness, descriptions of how the two aggregations are used are below.

Figure 10 defines an aggregation between the AuthenticationECAPolicyRule and an externalAuthenticationMethod class (which is likely a superclass for different types of authentication mechanisms). This decouples the implementation of authentication mechanisms from how authentication mechanisms are used.

Since different AuthenticationECAPolicyRules can use different authentication mechanisms in different ways, the aggregation is realized as an association class. This enables the attributes and methods of the association class (i.e., AuthenticationRuleDetail) to be used to define how a given AuthenticationMethod is used by a particular AuthenticationECAPolicyRule.

Similarly, the PolicyControlsAuthentication aggregation defines policies to control the configuration of the AuthenticationRuleDetail association class. This enables the entire authentication process to be managed by ECAPolicyRules.

Note: a data model MAY choose to collapse this design into a more efficient implementation. For example, a data model could define two attributes for the AuthenticationECAPolicyRule class, called (for example) authenticationMethodCurrent and authenticationMethodSupported, to represent the HasAuthenticationMethod aggregation and its association class. The

former is a string attribute that defines the current authentication method used by this AuthenticationECAPolicyRule, while the latter defines a set of authentication methods, in the form of an authentication capability, which this AuthenticationECAPolicyRule can advertise.

A.2. AuthorizationECAPolicyRuleClass Definition

The purpose of an AuthorizationECAPolicyRule is to define an ECA Policy Rule that can determine whether access to a resource should be given and, if so, what permissions should be granted to the entity that is accessing the resource.

This class does NOT define the authorization method(s) used. This is because this would effectively "enclose" this information within the AuthorizationECAPolicyRule. This has two drawbacks. First, other entities that need to use information from the Authorization class(es) could not; they would have to associate with the AuthorizationECAPolicyRule class, and those other classes would not likely be interested in the AuthorizationECAPolicyRule. Second, the evolution of new authorization methods should be independent of the AuthorizationECAPolicyRule; this cannot happen if the Authorization class(es) are embedded in the AuthorizationECAPolicyRule. Hence, this document recommends the following design:

```
                                                    +---------------+
    +----------------+ 1..n                   1...n |               |
    |                |/ \   HasAuthorizationMethod \| Authorization |
    | Authorization  + A ----------+----------------+     Method    |
    | ECAPolicyRule  |\ /          ^               /|               |
    |                |             |                +---------------+
    +----------------+             |
                                   |
                      +------------+------------+
                      | AuthorizationRuleDetail |
                      +------------+------------+
                                  / \ 0..n
                                   |
                                   | PolicyControlsAuthorization
                                   |
                                  / \
                                   A
                                  \ / 0..n
                        +----------+-------------+
                        | ManagementECAPolicyRule |
                        +------------------------+
```

Figure 11. Modeling Authorization Mechanisms

This document only defines the AuthorizationECAPolicyRule; all other classes, and the aggregations, are defined in an external model. For completeness, descriptions of how the two aggregations are used are below.

Figure 11 defines an aggregation between the AuthorizationECAPolicyRule and an external AuthorizationMethod class (which is likely a superclass for different types of authorization mechanisms). This decouples the implementation of authorization mechanisms from how authorization mechanisms are used.

Since different AuthorizationECAPolicyRules can use different authorization mechanisms in different ways, the aggregation is realized as an association class. This enables the attributes and methods of the association class (i.e., AuthorizationRuleDetail) to be used to define how a given AuthorizationMethod is used by a particular AuthorizationECAPolicyRule.

Similarly, the PolicyControlsAuthorization aggregation defines policies to control the configuration of the AuthorizationRuleDetail association class. This enables the entire authorization process to be managed by ECAPolicyRules.

Note: a data model MAY choose to collapse this design into a more efficient implementation. For example, a data model could define two attributes for the AuthorizationECAPolicyRule class, called (for example) authorizationMethodCurrent and authorizationMethodSupported, to represent the HasAuthorizationMethod aggregation and its association class. The former is a string attribute that defines the current authorization method used by this AuthorizationECAPolicyRule, while the latter defines a set of authorization methods, in the form of an authorization capability, which this AuthorizationECAPolicyRule can advertise.

A.3. AccountingECAPolicyRuleClass Definition

The purpose of an AccountingECAPolicyRule is to define an ECA Policy Rule that can determine which information to collect, and how to collect that information, from which set of resources for the purpose of trend analysis, auditing, billing, or cost allocation [RFC2975] [RFC3539].

This class does NOT define the accounting method(s) used. This is because this would effectively "enclose" this information within the AccountingECAPolicyRule. This has two drawbacks. First, other entities that need to use information from the Accounting class(es) could not; they would have to associate with the AccountingECAPolicyRule class, and those other classes would not likely be interested in the AccountingECAPolicyRule. Second, the evolution of new accounting methods should be independent of the AccountingECAPolicyRule; this cannot happen if the Accounting class(es) are embedded in the AccountingECAPolicyRule. Hence, this document recommends the following design:

```
                                                  +-------------+
    +----------------+ 1..n                 1...n |             |
    |                |/ \   HasAccountingMethod  \| Accounting  |
    |  Accounting    + A ----------+--------------+   Method    |
    | ECAPolicyRule  |\ /          ^             /|             |
    |                |             |              +-------------+
    +----------------+             |
                                   |
                        +----------+-----------+
                        | AccountingRuleDetail |
                        +----------+-----------+
                                  / \ 0..n
                                   |
                                   | PolicyControlsAccounting
                                   |
                                  / \
                                   A
                                  \ / 0..n
                        +----------+--------------+
                        | ManagementECAPolicyRule |
                        +-------------------------+
```

Figure 12. Modeling Accounting Mechanisms

This document only defines the AccountingECAPolicyRule; all other classes, and the aggregations, are defined in an external model. For completeness, descriptions of how the two aggregations are used are below.

Figure 12 defines an aggregation between the AccountingECAPolicyRule and an external AccountingMethod class (which is likely a superclass for different types of accounting mechanisms). This decouples the implementation of accounting mechanisms from how accounting mechanisms are used.

Since different AccountingECAPolicyRules can use different accounting mechanisms in different ways, the aggregation is realized as an association class. This enables the attributes and methods of the association class (i.e., AccountingRuleDetail) to be used to define how a given AccountingMethod is used by a particular AccountingECAPolicyRule.

Similarly, the PolicyControlsAccounting aggregation defines policies to control the configuration of the AccountingRuleDetail association class. This enables the entire accounting process to be managed by ECAPolicyRules.

Note: a data model MAY choose to collapse this design into a more efficient implementation. For example, a data model could define two attributes for the AccountingECAPolicyRule class, called (for example) accountingMethodCurrent and accountingMethodSupported, to represent the HasAccountingMethod aggregation and its association class. The former is a string attribute that defines the current accounting method used by this AccountingECAPolicyRule, while the latter defines a set of accounting methods, in the form of an authorization capability, which this AccountingECAPolicyRule can advertise.

A.4. TrafficInspectionECAPolicyRuleClass Definition

The purpose of a TrafficInspectionECAPolicyRule is to define an ECA Policy Rule that, based on a given context, can determine which traffic to examine on which devices, which information to collect from those devices, and how to collect that information.

This class does NOT define the traffic inspection method(s) used. This is because this would effectively "enclose" this information within the TrafficInspectionECAPolicyRule. This has two drawbacks. First, other entities that need to use information from the TrafficInspection class(es) could not; they would have to associate with the TrafficInspectionECAPolicyRule class, and those other classes would not likely be interested in the TrafficInspectionECAPolicyRule. Second, the evolution of new traffic inspection methods should be independent of the TrafficInspectionECAPolicyRule; this cannot happen if the TrafficInspection class(es) are embedded in the TrafficInspectionECAPolicyRule. Hence, this document recommends the following design:

```
                                                   +------------------+
   +-------------------+1..n                   1..n|                  |
   |                   |/ \ HasTrafficInspection  \|      Traffic     |
   | TrafficInspection + A ----------+-------------+ InspectionMethod |
   |   ECAPolicyRule   |\ /          ^           / |                  |
   |                   |             |             +------------------+
   +-------------------+             |
                                     |
                        +------------+------------+
                        | TrafficInspectionDetail |
                        +------------+------------+
                                    / \ 0..n
                                     |
                                     | PolicyControlsTrafficInspection
                                     |
                                    / \
                                     A
                                    \ / 0..n
                          +----------+--------------+
                          | ManagementECAPolicyRule |
                          +-------------------------+
```

Figure 13. Modeling Traffic Inspection Mechanisms

This document only defines the TrafficInspectionECAPolicyRule; all other classes, and the aggregations, are defined in an external model. For completeness, descriptions of how the two aggregations are used are below.

Figure 13 defines an aggregation between the TrafficInspectionECAPolicyRule and an external TrafficInspection class (which is likely a superclass for different types of traffic inspection mechanisms). This decouples the implementation of traffic inspection mechanisms from how traffic inspection mechanisms are used.

Since different TrafficInspectionECAPolicyRules can use different traffic inspection mechanisms in different ways, the aggregation is realized as an association class. This enables the attributes and methods of the association class (i.e., TrafficInspectionDetail) to be used to define how a given TrafficInspectionMethod is used by a particular TrafficInspectionECAPolicyRule.

Similarly, the PolicyControlsTrafficInspection aggregation defines policies to control the configuration of the TrafficInspectionDetail association class. This enables the entire traffic inspection process to be managed by ECAPolicyRules.

Note: a data model MAY choose to collapse this design into a more efficient implementation. For example, a data model could define two attributes for the TrafficInspectionECAPolicyRule class, called (for example) trafficInspectionMethodCurrent and trafficInspectionMethodSupported, to represent the HasTrafficInspectionMethod aggregation and its association class. The former is a string attribute that defines the current traffic inspection method used by this TrafficInspectionECAPolicyRule, while the latter defines a set of traffic inspection methods, in the form of a traffic inspection capability, which this TrafficInspectionECAPolicyRule can advertise.

A.5. ApplyProfileECAPolicyRuleClass Definition

The purpose of an ApplyProfileECAPolicyRule is to define an ECA Policy Rule that, based on a given context, can apply a particular profile to specific traffic. The profile defines the security capabilities for content security control and/or attack mitigation control; these will be described in sections 4.4 and 4.5, respectively.

This class does NOT define the set of Profiles used. This is because this would effectively "enclose" this information within the ApplyProfileECAPolicyRule. This has two drawbacks. First, other entities that need to use information from the Profile class(es) could not; they would have to associate with the ApplyProfileECAPolicyRule class, and those other classes would not likely be interested in the ApplyProfileECAPolicyRule. Second, the evolution of new Profile classes should be independent of the ApplyProfileECAPolicyRule; this cannot happen if the Profile class(es) are embedded in the ApplyProfileECAPolicyRule. Hence, this document recommends the following design:

```
                                                        +-------------+
       +-------------------+ 1..n                  1..n |             |
       |                   |/ \  ProfileApplied        \|             |
       | ApplyProfile      + A -----------+-------------+   Profile   |
       |   ECAPolicyRule   |\ /           ^            /|             |
       |                   |              |             +-------------+
       +-------------------+              |
                                          |
                             +------------+---------+
                             | ProfileAppliedDetail |
                             +------------+---------+
                                         / \ 0..n
                                          |
                                          |
         PolicyControlsProfileApplication |
                                          |
                                         / \
                                          A
                                         \ / 0..n
                               +----------+--------------+
                               | ManagementECAPolicyRule |
                               +-------------------------+
```

Figure 14. Modeling Profile ApplicationMechanisms

This document only defines the ApplyProfileECAPolicyRule; all other classes, and the aggregations, are defined in an external model. For completeness, descriptions of how the two aggregations are used are below.

Figure 14 defines an aggregation between the ApplyProfileECAPolicyRule and an external Profile class (which is likely a superclass for different types of Profiles). This decouples the implementation of Profiles from how Profiles are used.

Since different ApplyProfileECAPolicyRules can use different Profiles in different ways, the aggregation is realized as an association class. This enables the attributes and methods of the association class (i.e., ProfileAppliedDetail) to be used to define how a given Profileis used by a particular ApplyProfileECAPolicyRule.

Similarly, the PolicyControlsProfileApplication aggregation defines policies to control the configuration of the ProfileAppliedDetail association class. This enables the application of Profiles to be managed by ECAPolicyRules.

Note: a data model MAY choose to collapse this design into a more efficient implementation. For example, a data model could define two attributes for the ApplyProfileECAPolicyRuleclass, called (for example) profileAppliedCurrent and profileAppliedSupported, to represent the ProfileApplied aggregation and its association class. The former is a string attribute that defines the current Profile used by this ApplyProfileECAPolicyRule, while the latter defines a set of Profiles, in the form of a Profile capability, which this ApplyProfileECAPolicyRule can advertise.

A.6. ApplySignatureECAPolicyRuleClass Definition

The purpose of an ApplySignatureECAPolicyRule is to define an ECA Policy Rule that, based on a given context, can determine which Signature object (e.g., an anti-virus file, or aURL filtering file, or a script) to apply to which traffic. The Signature object defines the security capabilities for content security control and/or attack mitigation control; these will be described in sections 4.4 and 4.5, respectively.

This class does NOT define the set of Signature objects used. This is because this would effectively "enclose" this information within the ApplySignatureECAPolicyRule. This has two drawbacks. First, other entities that need to use information from the Signature object class(es) could not; they would have to associate with the ApplySignatureECAPolicyRule class, and those other classes would not likely be interested in the ApplySignatureECAPolicyRule. Second, the evolution of new Signature object classes should be independent of the ApplySignatureECAPolicyRule; this cannot happen if the Signature object class(es) are embedded in the ApplySignatureECAPolicyRule. Hence, this document recommends the following design:

```
                                                  +-------------+
     +---------------+ 1..n                  1..n |             |
     |               |/ \   SignatureApplied     \|             |
     | ApplySignature+ A ----------+--------------+  Signature  |
     | ECAPolicyRule |\ /          ^             /|             |
     |               |             |             +-------------+
     +---------------+             |
                                   |
                      +------------+-----------+
                      | SignatureAppliedDetail |
                      +------------+-----------+
                                  / \ 0..n
                                   |
                                   | PolicyControlsSignatureApplication
                                   |
                                  / \
                                   A
                                  \ / 0..n
                        +----------+--------------+
                        | ManagementECAPolicyRule |
                        +-------------------------+
```

Figure 15. Modeling Sginature Application Mechanisms

This document only defines the ApplySignatureECAPolicyRule; all other classes, and the aggregations, are defined in an external model. For completeness, descriptions of how the two aggregations are used are below.

Figure 15 defines an aggregation between the ApplySignatureECAPolicyRule and an external Signature object class (which is likely a superclass for different types of Signature objects). This decouples the implementation of signature objects from how Signature objects are used.

Since different ApplySignatureECAPolicyRules can use different Signature objects in different ways, the aggregation is realized as an association class. This enables the attributes and methods of the association class (i.e., SignatureAppliedDetail) to be used to define how a given Signature object is used by a particular ApplySignatureECAPolicyRule.

Similarly, the PolicyControlsSignatureApplication aggregation defines policies to control the configuration of the

SignatureAppliedDetail association class. This enables the application of the Signature object to be managed by policy.

Note: a data model MAY choose to collapse this design into a more efficient implementation. For example, a data model could define two attributes for the ApplySignatureECAPolicyRule class, called (for example) signature signatureAppliedCurrent and signatureAppliedSupported, to represent the SignatureApplied aggregation and its association class. The former is a string attribute that defines the current Signature object used by this ApplySignatureECAPolicyRule, while the latter defines a set of Signature objects, in the form of a Signature capability, which this ApplySignatureECAPolicyRule can advertise.

Authors' Addresses

Liang Xia (Frank)
Huawei

101 Software Avenue, Yuhuatai District
Nanjing, Jiangsu 210012
China

Email: Frank.xialiang@huawei.com

John Strassner
Huawei
Email: John.sc.Strassner@huawei.com

DaCheng Zhang
Huawei

Email: dacheng.zhang@huawei.com

Kepeng Li
Alibaba

Email: kepeng.lkp@alibaba-inc.com

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

Antonio Lioy
Politecnico di Torino
Corso Duca degli Abruzzi, 34
Torino, 10129
Italy
Email: lioy@polito.it

Diego R. Lopez
Telefonica I+D
Zurbaran, 12
Madrid,   28010
Spain

Phone: +34 913 129 041
Email: diego.r.lopez@telefonica.com

Edward Lopez
Fortinet
899 Kifer Road
Sunnyvale, CA 94086
Phone: +1 703 220 0988

EMail: elopez@fortinet.com

Nicolas BOUTHORS
Qosmos

Email: Nicolas.BOUTHORS@qosmos.com

Luyuan Fang
Microsoft
15590 NE 31st St
Redmond, WA 98052
Email: lufang@microsoft.com