I2NSF
Internet-Draft
Intended status: Standard Track
Expires: January 5, 2018

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

July 3, 2017

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

## Abstract

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

## Status of this Memo

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

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

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

This Internet-Draft will expire on January 5, 2018.

Table of Contents

Table of Contents (continued)

Table of Contents (continued)



## 1. Introduction

The rapid development of virtualized systems requires advanced security protection in various scenarios. Examples include network devices in an enterprise network, User Equipment in a mobile network, devices in the Internet of Things, or residential access users [I-D.draft-ietf-i2nsf-problem-and-use-cases].

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

Security Capabilities describe the set of network security-related features that are available to use for security policy enforcement purposes. Security Capabilities are independent of the actual security control mechanisms that will implement them. Every NSF registers the set of Capabilities it offers. Security Capabilities are a market enabler, providing a way to define customized security protection by unambiguously describing the security features offered by a given NSF. Moreover, Security Capabilities enable security functionality to be described in a vendor-neutral manner. That is, it is not required to refer to a specific product when designing the network; rather, the functionality characterized by their Capabilities are considered.

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

- Interface between I2NSF users and applications, and a security controller (Consumer-Facing Interface): this is a service-oriented interface that provides a communication channel between consumers of NSF data and services and the network operator's security controller. This enables security information to be exchanged between various applications (e.g., OpenStack, or various BSS/OSS components) and the security controller. The design goal of the Consumer-Facing Interface is to decouple the specification of security services from their implementation.

- o Interface between NSFs (e.g., firewall, intrusion prevention, or anti-virus) and the security controller (NSF-Facing Interface): The NSF-Facing Interface is used to decouple the security management scheme from the set of NSFs and their various implementations for this scheme, and is independent of how the NSFs are implemented (e.g., run in Virtual Machines or physical appliances). This document defines an object-oriented information model for network security, content security, and attack mitigation Capabilities, along with associated I2NSF Policy objects.

This document is organized as follows. Section 2 defines conventions and acronyms used. Section 3 discusses the design principles for the I2NSF Capability information model and related policy model objects. Section 4 defines the structure of the information model, which describes the policy and capability objects design; details of the model elements are contained in the appendices.

## 2. Conventions used in this document

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

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

### 2.1. Acronyms

| | |
|---|---|
| AAA: | Access control, Authorization, Authentication |
| ACL: | Access Control List |
| (D)DoD: | (Distributed) Denial of Service (attack) |
| ECA: | Event-Condition-Action |
| FMR: | First Matching Rule (resolution strategy) |
| FW: | Firewall |
| GNSF: | Generic Network Security Function |
| HTTP: | HyperText Transfer Protocol |
| I2NSF: | Interface to Network Security Functions |
| IPS: | Intrusion Prevention System |
| LMR: | Last Matching Rule (resolution strategy) |
| MIME: | Multipurpose Internet Mail Extensions |
| NAT: | Network Address Translation |
| NSF: | Network Security Function |
| RPC: | Remote Procedure Call |
| SMA: | String Matching Algorithm |
| URL: | Uniform Resource Locator |
| VPN: | Virtual Private Network |

## 3. Information Model Design

The starting point of the design of the Capability information model is the categorization of types of security functions. For instance, experts agree on what is meant by the terms "IPS", "Anti-Virus", and "VPN concentrator". Network security experts unequivocally refer to "packet filters" as stateless devices able to allow or deny packet forwarding based on various conditions (e.g., source and destination IP addresses, source and destination ports, and IP protocol type fields) [Alshaer].

However, more information is required in case of other devices, like stateful firewalls or application layer filters. These devices filter packets or communications, but there are differences in the packets and communications that they can categorize and the states they maintain. Analogous considerations can be applied for channel protection protocols, where we all understand that they will protect packets by means of symmetric algorithms whose keys could have been negotiated with asymmetric cryptography, but they may work at different layers and support different algorithms and protocols. To ensure protection, these protocols apply integrity, optionally confidentiality, anti-reply protections, and authenticate peers.

### 3.1. Capability Information Model Overview

This document defines a model of security Capabilities that provides the foundation for automatic management of NSFs. This includes enabling the security controller to properly identify and manage NSFs, and allow NSFs to properly declare their functionality, so that they can be used in the correct way.

Some basic design principles for security Capabilities and the systems that have to manage them are:

- Independence: each security Capability should be an independent function, with minimum overlap or dependency on other Capabilities. This enables each security Capability to be utilized and assembled together freely. More importantly, changes to one Capability will not affect other Capabilities. This follows the Single Responsibility Principle [Martin] [OODSRP].
- Abstraction: each Capability should be defined in a vendor-independent manner, and associated to a well-known interface to provide a standardized ability to describe and report its processing results. This facilitates multi-vendor interoperability.
- Automation: the system must have the ability to auto-discover, auto-negotiate, and auto-update its security Capabilities (i.e., without human intervention). These features are especially useful for the management of a large number of NSFs. They are essential to add smart services (e.g., analysis,

refinement, Capability reasoning, and optimization) for the security scheme employed. These features are supported by many design patterns, including the Observer Pattern [OODOP], the Mediator Pattern [OODMP], and a set of Message Exchange Patterns [Hohpe].

- o Scalability: the management system must have the Capability to scale up/down or scale in/out. Thus, it can meet various performancerequirements derived from changeable network traffic or service requests. In addition, security Capabilities that are affected by scalability changes must support reporting statistics to the security controller to assist its decision on whether it needs to invoke scaling or not. However, this requirement is for information only, and is beyond the scope of this document.

Based on the above principles, a set of abstract and vendor-neutral Capabilities with standard interfaces is defined. This provides a Capability model that enables a set of NSFs that are required at a given time to be selected, as well as the unambiguous definition of the security offered by the set of NSFs used. The security controller can compare the requirements of users and applications to the set of Capabilities that are currently available in order to choose which 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. The security controller may also be able to customize the functionality of selected NSFs.

Furthermore, when an unknown threat (e.g., zero-day exploits and unknown malware) is reported by a NSF, new Capabilities may be created, and/or existing Capabilities may be updated (e.g., by updating its signature and algorithm). This results in enhancing existing NSFs (and/or creating new NSFs) to address the new threats. New Capabilities may be sent to and stored in a centralized repository, or stored separately in a vendor’s local repository. In either case, a standard interface facilitates the update process.

Note that most systems cannot dynamically create a new Capability without human interaction. This is an area for further study.

## 3.2. ECA Policy Model Overview

The "Event-Condition-Action" (ECA) policy model is used as the basis for the design of I2NSF Policy Rules; definitions of all I2NSF policy-related terms are also defined in [I-D.draft-ietf-i2nsf-terminology]:

- o Event: An Event is any important occurrence in time of a change in the system being managed, and/or in the environment of the system being managed. When used in the context of I2NSF Policy Rules, it is used to determine whether the Condition clause of the I2NSF Policy Rule can be evaluated or not.

Examples of an I2NSF Event include time and user actions (e.g., logon, logoff, and actions that violate an ACL).

- Condition: A condition is defined as a set of attributes, features, and/or values that are to be compared with a set of known attributes, features, and/or values in order to determine whether or not the set of Actions in that (imperative) I2NSF Policy Rule can be executed or not. Examples of I2NSF Conditions include matching attributes of a packet or flow, and comparing the internal state of an NSF to a desired state.
- Action: An action is used to control and monitor aspects of flow-based NSFs when the event and condition clauses are satisfied. NSFs provide security functions by executing various Actions. Examples of I2NSF Actions include providing intrusion detection and/or protection, web and flow filtering, and deep packet inspection for packets and flows.

An I2NSF Policy Rule is made up of three Boolean clauses: an Event clause, a Condition clause, and an Action clause. A Boolean clause is a logical statement that evaluates to either TRUE or FALSE. It may be made up of one or more terms; if more than one term, then a Boolean clause connects the terms using logical connectives (i.e., AND, OR, and NOT). It has the following semantics:

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

Technically, the "Policy Rule" is really a container that aggregates the above three clauses, as well as metadata.

The above ECA policy model is very general and easily extensible, and can avoid potential constraints that could limit the implementation of generic security Capabilities.

## 3.3. Relation with the External Information Model

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

The I2NSF NSF-Facing Interface is in charge of selecting and managing the NSFs using their Capabilities. This is done using the following approach:

1) Each NSF registers its Capabilities with the management system when it "joins", and hence makes its Capabilities available to the management system;
2) The security controller selects the set of Capabilities required to meet the needs of the security service from all available NSFs that it manages;

3) The security controller uses the Capability information model to match chosen Capabilities to NSFs, independent of vendor;
4) The security controller takes the above information and creates or uses one or more data models from the Capability information model to manage the NSFs;
5) Control and monitoring can then begin.

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

Capabilities are defined as classes (e.g., a set of objects that exhibit a common set of characteristics and behavior [I-D.draft-ietf-supa-generic-policy-info-model].

Each Capability is made up of at least one model element (e.g., attribute, method, or relationship) that differentiates it from all other objects in the system. Capabilities are, generically, a type of metadata (i.e., information that describes, and/or prescribes, the behavior of objects); hence, it is also assumed that an external information model is used to define metadata (preferably, in the form of a class hierarchy). Therefore, it is assumed that Capabilities are subclassed from an external metadata model.

The Capability sub-model 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 NSF-Facing Interface does not care whether the NSF is virtualized or hosted in a physical device, who the vendor of the NSF is, 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:

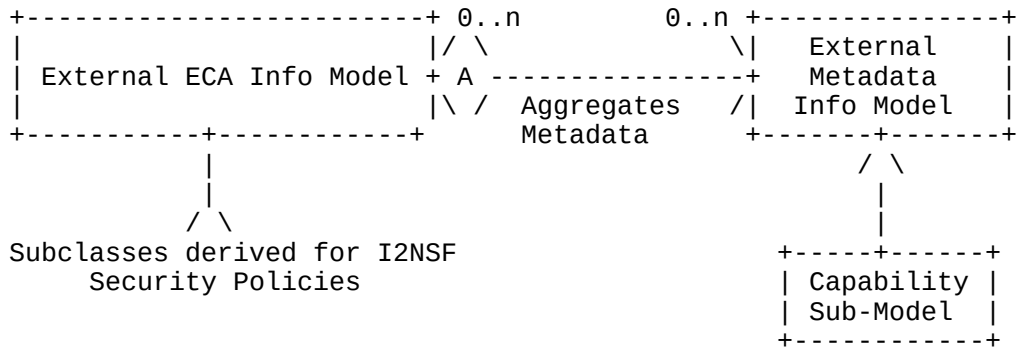


Figure 1. The Overall I2NSF Information Model Design

This draft defines a set of extensions to a generic, external, ECA Policy Model to represent various NSF ECA Security Policy Rules. It also defines the Capability Sub-Model; this enables ECA Policy Rules to control which Capabilities are seen by which actors, and used by the I2NSF system. Finally, it places requirements on what type of extensions are required to the generic, external, ECA information model and metadata models, in order to manage the lifecycle of I2NSF Capabilities.

Both of the external models shown in Figure 1 could, but do not have to, be based on the SUPA information model [I-D.draft-ietf-supa-generic-policy-info-model]. Note that classes in the Capability Sub-Model will inherit the AggregatesMetadata aggregation from the External Metadata Information Model.

The external ECA Information Model supplies at least a set of classes that represent a generic ECA Policy Rule, and a set of classes 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, as well as refine it (i.e., create new subclasses, or add attributes and relationships) to represent I2NSF-specific concepts.

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.

### 3.4. I2NSF Capability Information Model: Theory of Operation

Capabilities are typically used to represent NSF functions that can be invoked. Capabilities are objects, and hence, can be used in the event, condition, and/or action clauses of an I2NSF ECA Policy Rule. The I2NSF Capability information model refines a predefined metadata model; the application of I2NSF Capabilities is done by refining a predefined ECA Policy Rule information model that defines how to use, manage, or otherwise manipulate a set of Capabilities. In this approach, an I2NSF Policy Rule is a container that is made up of three clauses: an event clause, a condition clause, and an action clause. When the I2NSF policy engine receives a set of events, it matches those events to events in active ECA Policy Rules. If the event matches, then this triggers the evaluation of the condition clause of the matched I2NSF Policy Rule. The condition clause is then evaluated; if it matches, then the set of actions in the matched I2NSF Policy Rule MAY be executed.

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

Formally, given a set of actions in an I2NSF Policy Rule, the resolution strategy maps all the possible subsets of actions to an outcome. In other words, the resolution strategy is included in the I2NSF Policy Rule to decide how to evaluate all the actions in a particular I2NSF Policy Rule. This is then extended to include all possible I2NSF Policy Rules that can be applied in a particular scenario. Hence, the final action set from all I2NSF Policy Rules is deduced.

Some concrete examples of resolution strategy are the First Matching Rule (FMR) or Last Matching Rule (LMR) resolution strategies. When no rule matches a packet, the NSFs may select a default action, if they support one.

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

### 3.4.1. I2NSF Condition Clause Operator Types

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

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

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

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

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

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

Range-based selectors are ordered sets where it is possible to naturally specify ranges as they can be easily mapped to integers. As an example, the ports in the TCP protocol may be represented with a range-based selector (e.g., 1024-65535). As another example, the following are examples of valid condition clauses:

```
source_port = 80
source_port < 1024
source_port < 30000 && source_port >= 1024
```

We include, in range-based selectors, the category of selectors that have been defined by Al-Shaer et al. as "prefix-match" [Alshaer]. These selectors allow the specification of ranges of values by means of simple regular expressions. The typical case is the IP address selector (e.g., 10.10.1.*).

There is no need to distinguish between prefix match and range-based selectors; for example, the address range "10.10.1.*" maps to "[10.10.1.0,10.10.1.255]".

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

As an example, the condition clause:

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

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

```
"MIME_type = video/*"
```

matches all MIME objects whose type is video.

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

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

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

### 3.4.2. Capability Selection and Usage

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

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

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

The next section will introduce the algebra to define the information model of Capability registration. This associates NSFs to Capabilities, and checks whether a NSF has the Capabilities needed to enforce policies.

### 3.4.3. Capability Algebra

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

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

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

For example, if we have two Policies:

P1: During 8am-6pm, if traffic is external, then run through FW
P2: During 7am-8pm, conduct anti-malware investigation

There is no conflict between P1 and P2, since the actions are different. However, consider these two policies:

P3: During 8am-6pm, John gets GoldService
P4: During 10am-4pm, FTP from all users gets BronzeService

P3 and P4 are now in conflict, because between the hours of 10am and 4pm, the actions of P3 and P4 are different and apply to the same user (i.e., John).

Let us define the concept of a "matched" policy rule as one in which its event and condition clauses both evaluate to true. This enables the actions in this policy rule to be evaluated. Then, the conflict matrix is defined by a 5-tuple {Ac, Cc, Ec, RSc, Dc}, where:

- o Ac is the set of Actions currently available from the NSF;
- o Cc is the set of Conditions currently available from the NSF;
- o Ec is the set of Events the NSF is able to respond to. Therefore, the event clause of an I2NSF ECA Policy Rule that is written for an NSF will only allow a set of designated events in Ec. For compatibility purposes, we will assume that if Ec={} (that is, Ec is empty), the NSF only accepts CA policies.
- o RSc is the set of Resolution Strategies that can be used to specify how to resolve conflicts that occur between the actions of the same or different policy rules that are matched and contained in this particular NSF;
- o Dc defines the notion of a Default action that can be used to specify a predefined action when no other alternative action was matched by the currently executing I2NSF Policy Rule. An analogy is the use of a default statement in a C switch statement. This field of the Capability algebra can take the following values:
  - An explicit action (that has been predefined; typically, this means that it is fixed and not configurable), denoted as Dc ={a}. In this case, the NSF will always use the action as as the default action.
  - A set of explicit actions, denoted Dc={a1,a2, ...}; typically, this means that any **one** action can be used as the default action. This enables the policy writer to choose one of a predefined set of actions {a1, a2, ...} to serve as the default action.

- A fully configurable default action, denoted as Dc={F}. Here, F is a dummy symbol (i.e., a placeholder value) that can be used to indicate that the default action can be freely selected by the policy editor from the actions Ac available at the NSF. In other words, one of the actions Ac may be selected by the policy writer to act as the default action.
- No default action, denoted as Dc={}, for cases where the NSF does not allow the explicit selection of a default action.

```
*** Note to WG: please review the following paragraphs
*
*  Interesting Capability concepts that could be considered for a next
*  version of the Capability model and algebra include:
*
*     o Event clause representation (e.g., conjunctive vs. disjunctive
*       normal form for Boolean clauses)
*     o Event clause evaluation function, which would enable more
*       complex expressions than simple Boolean expressions to be used
*
*
*     o Condition clause representation (e.g., conjunctive vs.
*       disjunctive normal form for Boolean clauses)
*     o Condition clause evaluation function, which would enable more
*       complex expressions than simple Boolean expressions to be used
*     o Action clause evaluation strategies (e.g., execute first
*       action only, execute last action only, execute all actions,
*       execute all actions until an action fails)
*     o The use of metadata, which can be associated to both an I2NSF
*       Policy Rule as well as objects contained in the I2NSF Policy
*       Rule (e.g., an action), that describe the object and/or
*       prescribe behavior. Descriptive examples include adding
*       authorship information and defining a time period when an NSF
*       can be used to be defined; prescriptive examples include
*       defining rule priorities and/or ordering.
*
*  Given two sets of Capabilities, denoted as
*
*     cap1=(Ac1,Cc1,Ec1,RSc1,Dc1) and
*     cap2=(Ac2,Cc2,Ec2,RSc2,Dc2),
*
*  two set operations are defined for manipulating Capabilities:
*
*     o Capability addition:
*          cap1+cap2 = {Ac1 U Ac2, Cc1 U Cc2, Ec1 U Ec2, RSc1, Dc1}
*     o Capability subtraction:
*          cap1-cap2 = {Ac1 \ Ac2, Cc1 \ Cc2, Ec1 \ Ec2, RSc1, Dc1}
*
*  In the above formulae, "U" is the set union operator and "\" is the
*  set difference operator.
```

*  The addition and subtraction of Capabilities are defined as the
*  addition (set union) and subtraction (set difference) of both the
*  Capabilities and their associated actions. Note that **only** the
*  leftmost (in this case, the first matched policy rule) Resolution
*  Strategy and Default Action are used.
*
*  Note: actions, events, and conditions are **symmetric**. This means
*  that when two matched policy rules are merged, the resultant actions
*  and Capabilities are defined as the union of each individual matched
*  policy rule. However, both resolution strategies and default actions
*  are **asymmetric** (meaning that in general, they can **not** be
*  combined, as one has to be chosen). In order to simplify this, we
*  have chosen that the **leftmost** resolution strategy and the
*  **leftmost** default action are chosen. This enables the developer
*  to view the leftmost matched rule as the "base" to which other
*  elements are added.
*
*  As an example, assume that a packet filter Capability, Cpf, is
*  defined. Further, assume that a second Capability, called Ctime,
*  exists, and that it defines time-based conditions. Suppose we need
*  to construct a new generic packet filter, Cpfgen, that adds
*  time-based conditions to Cpf.
*
*
*  Conceptually, this is simply the addition of the Cpf and Ctime
*  Capabilities, as follows:
*     Apf   =  {Allow, Deny}
*     Cpf   =  {IPsrc,IPdst,Psrc,Pdst,protType}
*     Epf   =  {}
*     RSpf  =  {FMR}
*     Dpf   =  {A1}
*
*     Atime =  {Allow, Deny, Log}
*     Ctime =  {timestart, timeend, datestart, datestop}
*     Etime =  {}
*     RStime = {LMR}
*     Dtime =  {A2}
*
*  Then, Cpfgen is defined as:
*     Cpfgen = {Apf U Atime, Cpf U Ctime, Epf U Etime, RSpf, Dpf}
*            = {Allow, Deny, Log},
*              {{IPsrc, IPdst, Psrc, Pdst, protType} U
*               {timestart, timeend, datestart, datestop}},
*              {},
*              {FMR},
*              {A1}
*
*  In other words, Cpfgen provides three actions (Allow, Deny, Log),
*  filters traffic based on a 5-tuple that is logically ANDed with a
*  time period, and uses FMR; it provides A1 as a default action, and
*  it does not react to events.

* Note: We are investigating, for a next revision of this draft, the
* possibility to add further operations that do not follow the
* symmetric vs. asymmetric properties presented in the previous note.
* We are looking for use cases that may justify the complexity added
* by the availability of more Capability manipulation operations.
*
*** End Note to WG

### 3.5. Initial NSFs Capability Categories

The following subsections define three common categories of Capabilities: network security, content security, and attack mitigation. Future versions of this document may expand both the number of categories as well as the types of Capabilities within a given category.

#### 3.5.1. Network Security Capabilities

Network security is a category that describes the inspecting and processing of network traffic based on the use of 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 the context of 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.

#### 3.5.2. Content Security Capabilities

Content security is another category of security Capabilities applied to the application layer. Through analyzing traffic contents carried in, for example, the application layer, content security Capabilities can be used to identify various security functions that are required. These include defending against intrusion, inspecting for viruses, filtering malicious URL or junk email, blocking illegal web access, or preventing malicious data retrieval.

Generally, each type of threat in the content security category has a set of unique characteristics, and requires handling using a set of methods that are specific to that type of content. Thus, these Capabilities will be characterized by their own content-specific security functions.

### 3.5.3. Attack Mitigation Capabilities

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:

- 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, and ssl DDoS.
- o Single-packet attacks:
  - 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 set of Capabilities, for detection and mitigation. The implementation and management of this category of security Capabilities of attack mitigation control is very similar to the content security control category. A standard interface, through which the security controller can choose and customize the given security Capabilities according to specific requirements, is essential.

## 4. Information Sub-Model for Network Security Capabilities

The purpose of the Capability Information Sub-Model is to define the concept of a Capability, and enable Capabilities to be aggregated to appropriate objects. The following sections present the Network Security, Content Security, and Attack Mitigation Capability sub-models.

### 4.1. 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:

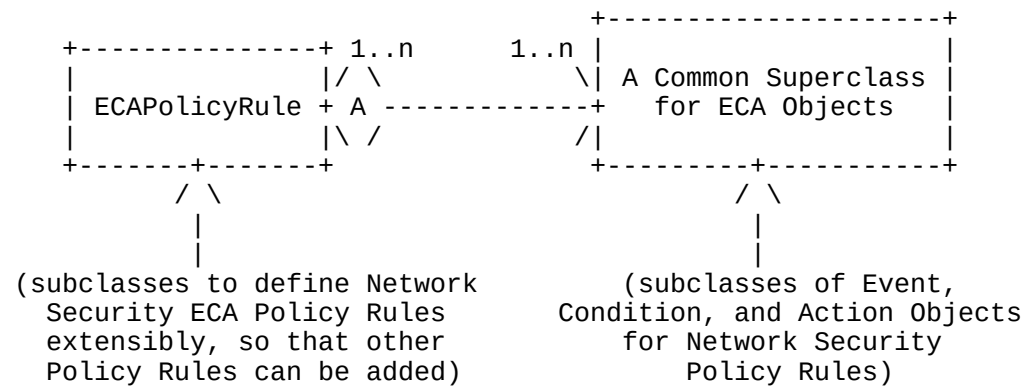


Figure 2. Network Security Information Sub-Model Overview

In the above figure, the ECAPolicyRule, along with the Event, Condition, and Action Objects, are defined in the external ECA Information Model. The Network Security Sub-Model extends all of these objects in order to define security-specific ECA Policy Rules, as well as extensions to the (generic) Event, Condition, and Action objects.

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, actions, and some extensions like resolution policy, default action and external data), and optionally, metadata. It can be applied to both uni- 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. This takes the following conceptual form:

```
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**. Hence, they can contain combinations of terms connected by the three logical connectives operators (i.e., AND, OR, NOT). An example is:

```
((SLA==GOLD) AND ((numPackets>burstRate) OR NOT(bwAvail<minBW)))
```

Note that Metadata, such as Capabilities, can be aggregated by I2NSF ECA Policy Rules.

#### 4.1.1. 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.

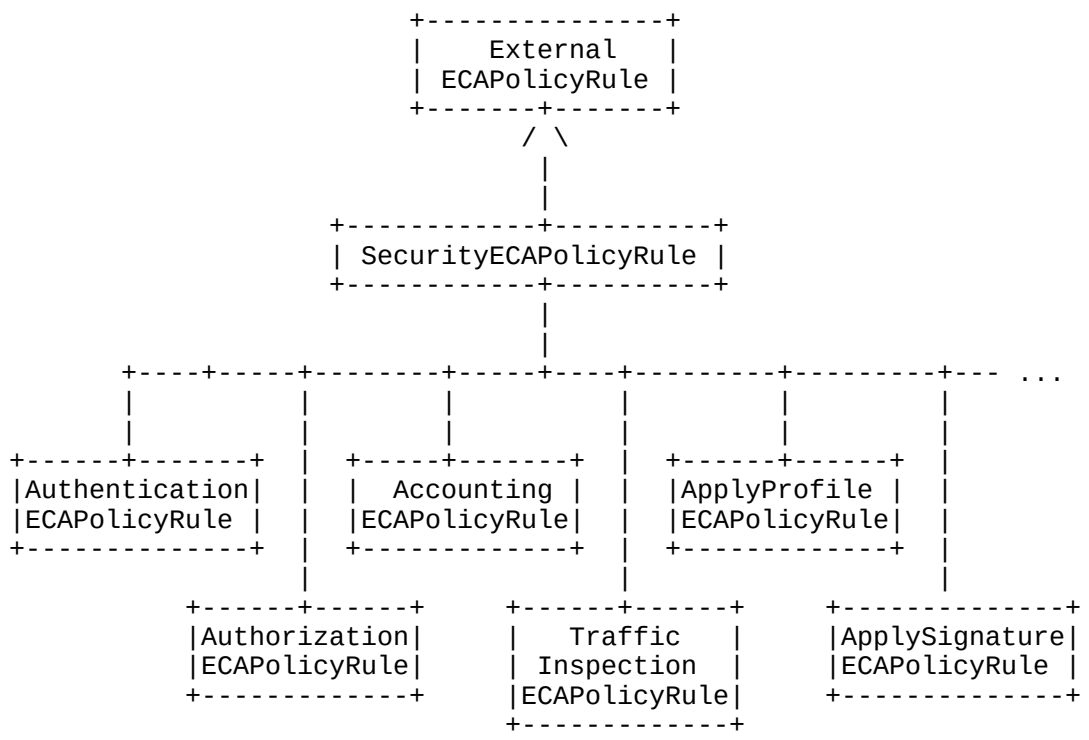


Figure 3. Network Security Info Sub-Model ECAPolicyRule Extensions

The SecurityECAPolicyRule is the top of the I2NSF ECA Policy Rule hierarchy. It inherits from the (external) generic ECA Policy Rule, and represents the specialization of this generic ECA Policy Rule to add security-specific 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 necessary information.

*** Note to WG
*
*  The design in Figure 3 represents the simplest conceptual design
*  for network security. An alternative model would be to use a
*  software pattern (e.g., the Decorator pattern); this would result
*  in the SecurityECAPolicyRule class being "wrapped" by one or more
*  of the six subclasses shown in Figure 3. The advantage of such a
*  pattern is to reduce the number of active objects at runtime, as
*  well as offer the ability to combine multiple actions of different
*  policy rules (e.g., inspect traffic and then apply a filter) into
*  one. The disadvantage is that it is a more complex software design.
*  The design team is requesting feedback from the WG regarding this.
*
*** End of Note to WG

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. For example, abstract classes may be collapsed into concrete classes.

The SecurityECAPolicyRule defines network security policy as a container that aggregates Event, Condition, and Action objects, which are described in Section 4.1.3, 4.1.4, and 4.1.5, respectively. Events, Conditions, and Actions can be generic or security-specific.

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

### 4.1.2. Network Security Policy Rule Operation

A Network Security Policy consists of one or more ECA Policy Rules formed from the information model described above. In simpler cases, where the Event and Condition clauses remain unchanged, then the action of one Policy Rule may invoke additional network security actions from other Policy Rules. Network security policy examines and performs basic processing of the traffic as follows:

1. The NSF evaluates the Event clause of a given SecurityECAPolicyRule (which can be generic or specific to security, such as those in Figure 3). It may use security Event objects to do all or part of this evaluation, which are defined in section 4.1.3. If the Event clause evaluates to TRUE, then the Condition clause of this SecurityECAPolicyRule is evaluated; otherwise, the 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 4.1.4. If the Condition clause evaluates to TRUE, it is defined as "matching" the SecurityECAPolicyRule; otherwise, execution of this SecurityECAPolicyRule is stopped, and the next SecurityECAPolicyRule (if one exists) is evaluated.
3. The set of actions to be executed are retrieved, and then the resolution strategy is used to define their execution order. This process includes using any optional external data associated with the SecurityECAPolicyRule.
4. Execution then takes one of the following three forms:
   a. If one or more actions is selected, then the NSF may perform those actions as defined by the resolution strategy. For example, the resolution strategy may only allow a single action to be executed (e.g., FMR or LMR), or it may allow all actions to be executed (optionally, in a particular order). In these and other cases, the NSF Capability MUST clearly define how execution will be done. It may use security Action objects to do all or part of this execution, which are defined in section 4.1.5. 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.
   b. If no actions are selected, and if a default action exists, then the default action is performed. Otherwise, no actions are performed.
   c. Otherwise, the traffic is denied.
5. If other security Capabilities (e.g., the conditions and/or actions implied by Anti-virus or IPS profile NSFs) are referenced in the action set of the SecurityECAPolicyRule, the NSF can be configured to use the referenced security Capabilities (e.g., check conditions or enforce actions). Execution then terminates.

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.

#### 4.1.3. Network Security Event Sub-Model

Figure 4 shows a more detailed design of the Event subclasses that are contained in the Network Security Information Sub-Model.

The four Event classes shown in Figure 4 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.

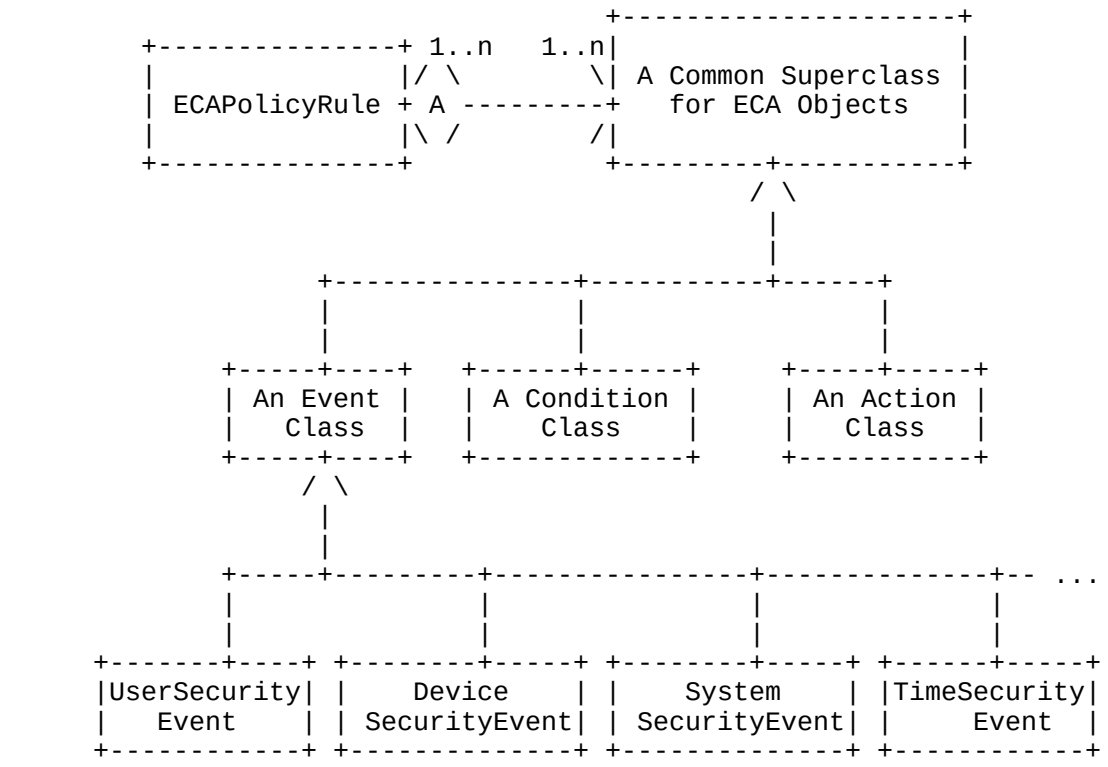


Figure 4. Network Security Info Sub-Model Event Class Extensions

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 4). However, this makes it harder to use any generic Event model with the I2NSF events. Assumptions are:

- 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 4 would be subclasses of the Atomic Event class); otherwise, a mechanism is needed to be able to group Events into a collection
- 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

*** Note to WG:
*
*  The design in Figure 4 represents the simplest conceptual design
*  design for describing Security Events. An alternative model would
*  be to use a software pattern (e.g., the Decorator pattern); this
*  would result in the SecurityEvent class being "wrapped" by one or
*  more of the four subclasses shown in Figure 4. The advantage of
*  such a pattern is to reduce the number of active objects at runtime,
*  as well as offer the ability to combine multiple events of different
*  types into one. The disadvantage is that it is a more complex
*  software design.
*
*** End of Note to WG

Brief class descriptions are provided in Appendix B.

#### 4.1.4. Network Security Condition Sub-Model

Figure 5 shows a more detailed design of the Condition subclasses that are contained in the Network Security Information Sub-Model. The six Condition classes shown in Figure 5 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 5), this makes it harder to use any generic Condition model with the I2NSF conditions.

*** Note to WG:
*
*  The design in Figure 5 represents the simplest conceptual design
*  for describing Security Conditions. An alternative model would be
*  to use a software pattern (e.g., the Decorator pattern); this would
*  result in the SecurityCondition class being "wrapped" by one or
*  more of the six subclasses shown in Figure 5. The advantage of such
*  a pattern is to reduce the number of active objects at runtime, as
*  well as offer the ability to combine multiple conditions of
*  different types into one. The disadvantage is that it is a more
*  complex software design.
*  The design team is requesting feedback from he WG regarding this.
*
*** End of Note to WG

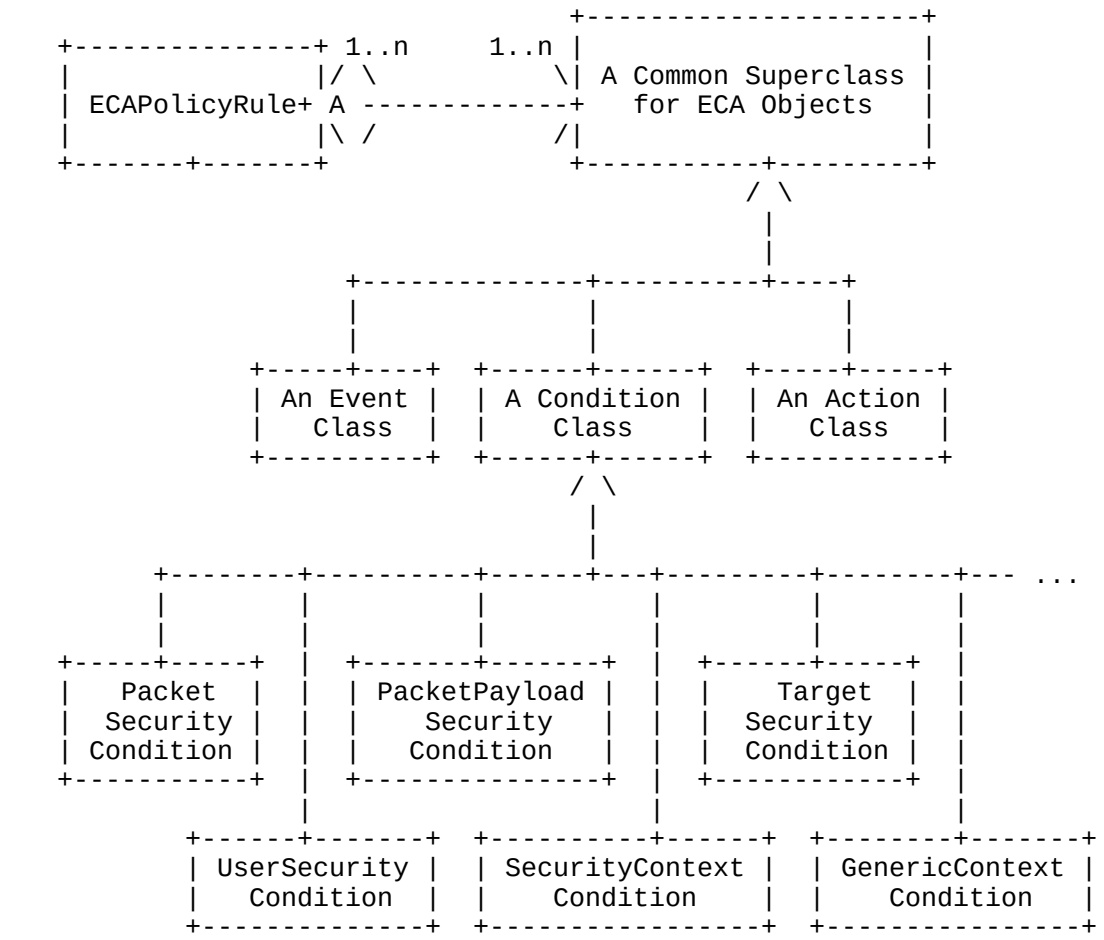


Figure 5. Network Security Info Sub-Model Condition Class Extensions

Brief class descriptions are provided in Appendix C.

### 4.1.5. Network Security Action Sub-Model

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

The four Action classes shown in Figure 6 extend the (external) generic Action class to represent Actions that perform a Network Security Control function.

The three Action classes shown in Figure 6 extend the (external) generic Action class to represent Actions that are of interest to Network Security. It is assumed that the (external) generic Action class is abstract, so that data model optimizations may be defined.

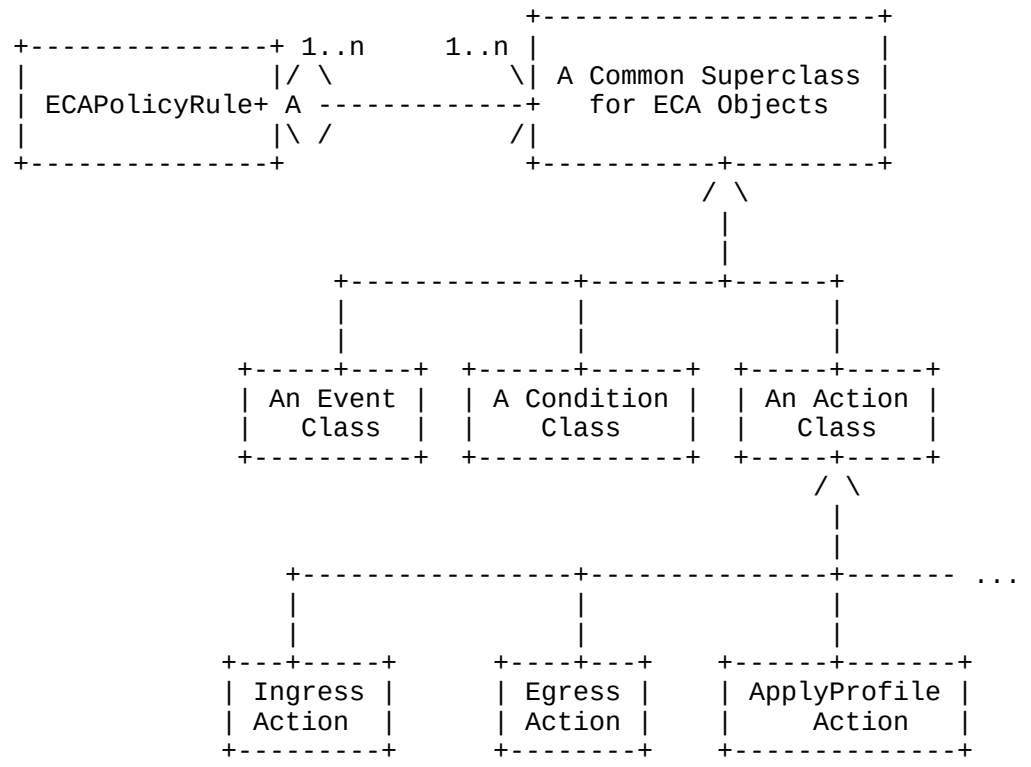


Figure 6. Network Security Info Sub-Model Action Extensions

It is also assumed that the generic Action class defines basic Action 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 SecurityAction class (which would be a subclass of the generic Action class, and a superclass of the six Action classes shown in Figure 6), this makes it harder to use any generic Action model with the I2NSF actions.

*** Note to WG
*  The design in Figure 6 represents the simplest conceptual design
*  for describing Security Actions. An alternative model would be to
*  use a software pattern (e.g., the Decorator pattern); this would
*  result in the SecurityAction class being "wrapped" by one or more
*  of the three subclasses shown in Figure 6. The advantage of such a
*  pattern is to reduce the number of active objects at runtime, as
*  well as offer the ability to combine multiple actions of different
*  types into one. The disadvantage is that it is a more complex
*  software design.
*  The design team is requesting feedback from the WG regarding this.
*** End of Note to WG

Brief class descriptions are provided in Appendix D.

## 4.2. Information Model for I2NSF Capabilities

The I2NSF Capability Model is made up of a number of Capabilities that represent various content security and attack mitigation functions. Each Capability protects against a specific type of threat in the application layer. This is shown in Figure 7.

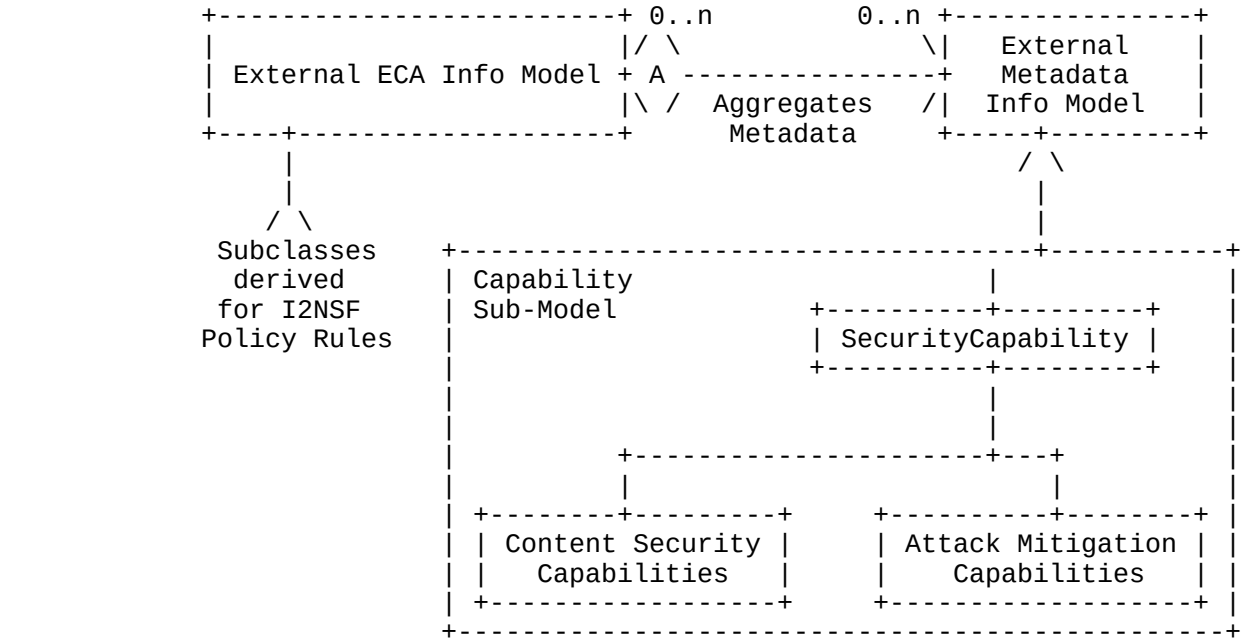


Figure 7. I2NSF Security Capability High-Level Model

Figure 7 shows a common I2NSF Security Capability class, called SecurityCapability. This enables us to add common attributes, relationships, and behavior to this class without affecting the design of the external metadata information model. All I2NSF Security Capabilities are then subclassed from the SecuritCapability class.

Note: the SecurityCapability class will be defined in the next version of this draft, after feedback from the WG is obtained.

## 4.3. Information Model for Content Security Capabilities

Content security is composed of a number of distinct security Capabilities; each such Capability protects against a specific type of threat in the application layer. Content security is a type of Generic Network Security Function (GNSF), which summarizes a well-defined set of security Capabilities, and was shown in Figure 7.

Figure 8 shows exemplary types of the content security GNSF.

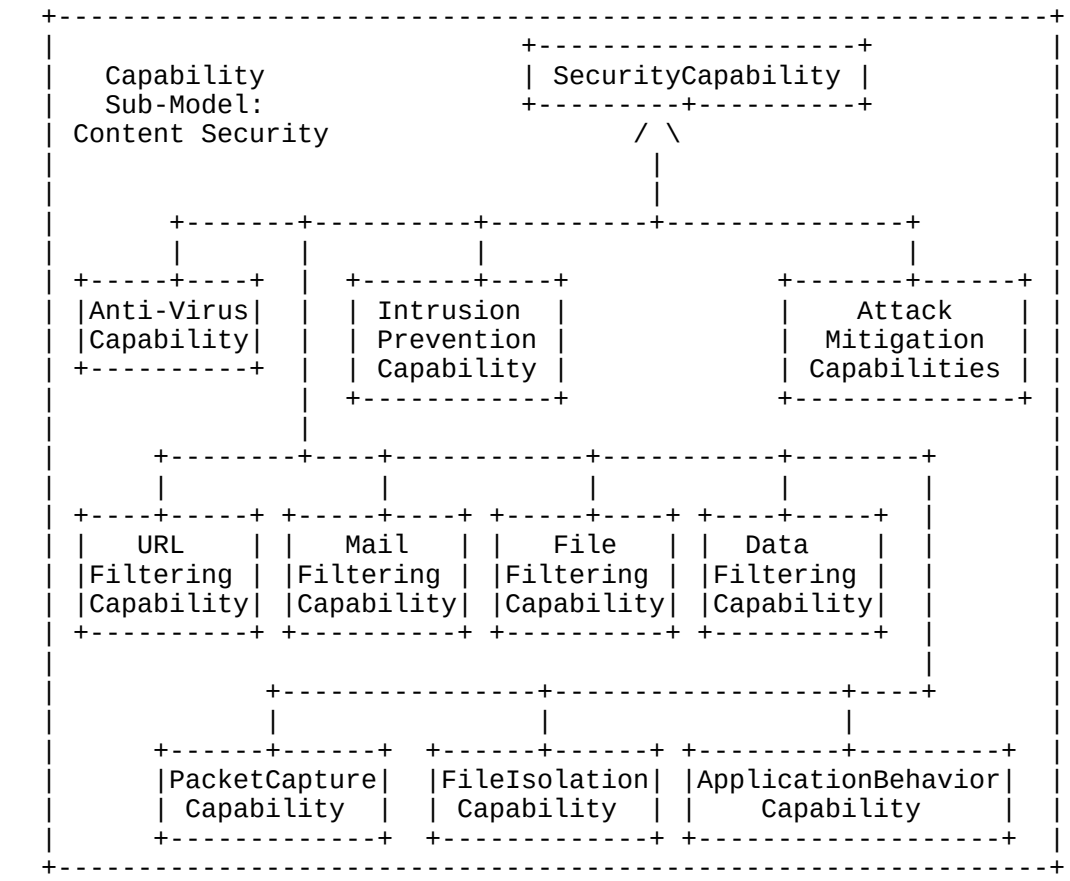


Figure 8. Network Security Capability Information Model

The detailed description about a standard interface, and the parameters for all the security Capabilities of this category, will be defined in a future version of this document.

## 4.4. Information Model for Attack Mitigation Capabilities

Attack mitigation is composed of a number of GNSFs; each one protects against a specific type of network attack. Attack Mitigation security is a type of GNSF, which summarizes a well-defined set of security Capabilities, and was shown in Figure 7. Figure 9 shows exemplary types of Attack Mitigation GNSFs.

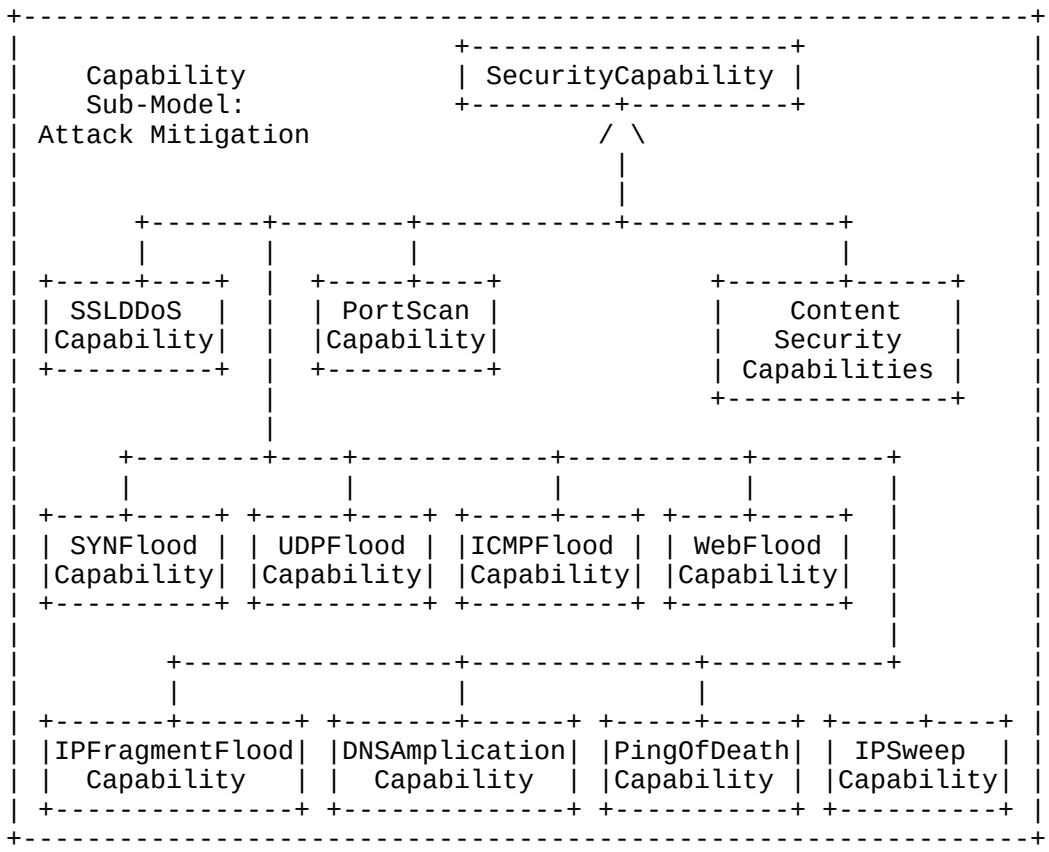


Figure 9. Attack Mitigation Capability Information Model

The detailed description about a standard interface, and the parameters for all the security Capabilities of this category, will be defined in a future version of this document.

## 5. Security Considerations

The security Capability policy information sent to NSFs should be protected by a secure communication channel, to ensure its confidentiality and integrity. Note that the NSFs and security controller can all be spoofed, which leads to undesirable results (e.g., security policy leakage from security controller, or a spoofed security controller sending false information to mislead the NSFs). Hence, mutual authentication MUST be supported to protected against this kind of threat. The current mainstream security technologies (i.e., TLS, DTLS, and IPSEC) can be employed to protect against the above threats.

In addition, to defend against DDoS attacks caused by a hostile security controller sending too many configuration messages to the NSFs, rate limiting or similar mechanisms should be considered.

## 6. IANA Considerations

TBD

## 7. Contributors

The following people contributed to creating this document, and are listed below in alphabetical order:

Antonio Lioy (Politecnico di Torino)
Dacheng Zhang (Huawei)
Edward Lopez (Fortinet)
Fulvio Valenza (Politecnico di Torino)
Kepeng Li (Alibaba)
Luyuan Fang (Microsoft)
Nicolas Bouthors (QoSmos)

## 8. References

### 8.1. Normative References

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

[RFC3539]
Aboba, B., and Wood, J., "Authentication, Authorization, and Accounting (AAA) Transport Profile", RFC 3539, June 2003.

## 8.2. Informative References

[RFC2975]
Aboba, B., et al., "Introduction to Accounting Management", RFC 2975, October 2000.

[I-D.draft-ietf-i2nsf-problem-and-use-cases]
Hares, S., et.al., "I2NSF Problem Statement and Use cases", draft-ietf-i2nsf-problem-and-use-cases-16, May 2017.

[I-D.draft-ietf-i2nsf-framework]
Lopez, E., et.al., "Framework for Interface to Network Security Functions", draft-ietf-i2nsf-framework-06, July, 2017.

[I-D.draft-ietf-i2nsf-terminology]
Hares, S., et.al., "Interface to Network Security Functions (I2NSF) Terminology", draft-ietf-i2nsf-terminology-03, March, 2017

[I-D.draft-ietf-supa-generic-policy-info-model]
Strassner, J., Halpern, J., van der Meer, S., "Generic Policy Information Model for Simplified Use of Policy Abstractions (SUPA)", draft-ietf-supa-generic-policy-info-model-03, May, 2017.

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

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

[Bas15]
Basile, C. and Lioy, A., "Analysis of application-layer filtering policies with application to HTTP", IEEE/ACM Transactions on Networking, Vol 23, Issue 1, February 2015.

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

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

[Lunt]
van Lunteren, J. and T. Engbersen, "Fast and scalable packet classification", IEEE Journal on Selected Areas in Communication, vol 21, Issue 4, September 2003.

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

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

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

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

## Appendix A. Network Security Capability Policy Rule Definitions

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

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 I2NSF ECA Policy Rule that can verify whether an entity has an attribute of a specific value. A high-level conceptual figure is shown below.

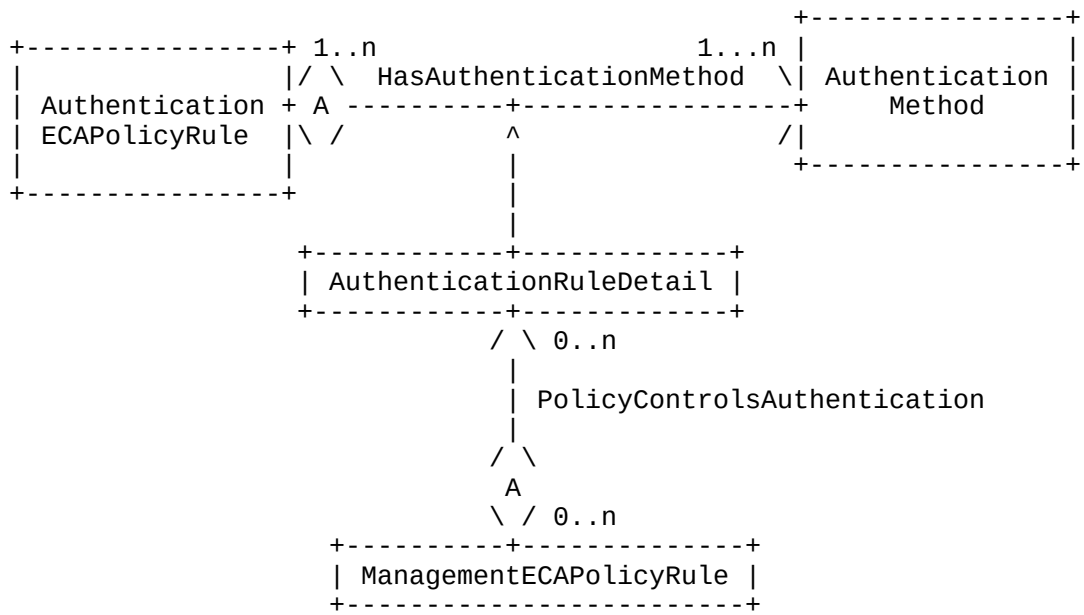


Figure 10. Modeling Authentication Mechanisms

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.

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 described below.

Figure 10 defines an aggregation between an external class, which defines one or more authentication methods, and an AuthenticationECAPolicyRule. This decouples the implementation of authentication mechanisms from how authentication mechanisms are managed and 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 Policy Rules to control the configuration of the AuthenticationRuleDetail association class. This enables the entire authentication process to be managed by ECA PolicyRules.

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 (e.g., called authenticationMethodCurrent and authenticationMethodSupported), to represent the HasAuthenticationMethod aggregation and its association class. The former would be a string attribute that defines the current authentication method used by this AuthenticationECAPolicyRule, while the latter would define 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 I2NSF 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:

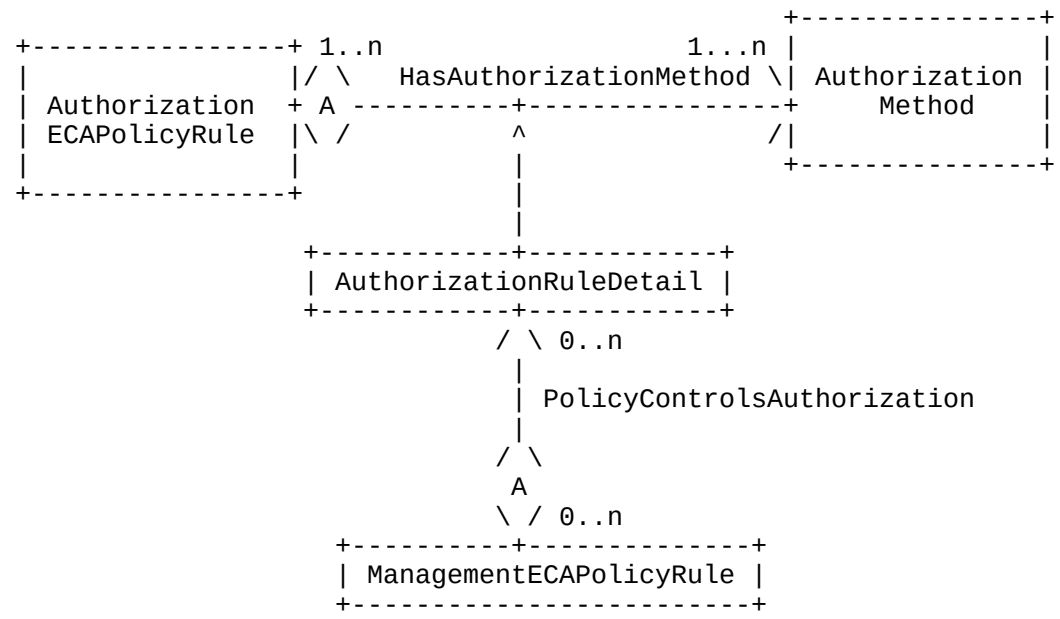


Figure 11. Modeling Authorization Mechanisms

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 described below.

Figure 11 defines an aggregation between the AuthorizationECAPolicyRule and an external class that defines one or more authorization methods. This decouples the implementation of authorization mechanisms from how authorization mechanisms are managed and 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 Policy Rules to control the configuration of the AuthorizationRuleDetail association class. This enables the entire authorization process to be managed by ECA PolicyRules.

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 I2NSF 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:

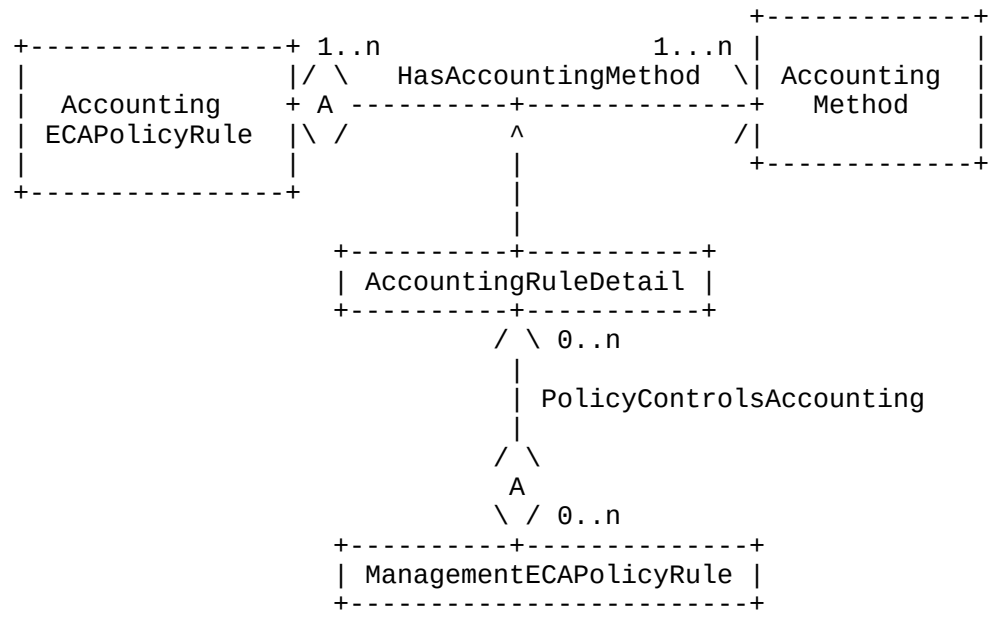


Figure 12.  Modeling Accounting Mechanisms

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 described below.

Figure 12 defines an aggregation between the AccountingECAPolicyRule and an external class that defines one or more accounting methods. This decouples the implementation of accounting mechanisms from how accounting mechanisms are managed and 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 Policy Rules to control the configuration of the AccountingRuleDetail association class. This enables the entire accounting process to be managed by ECA PolicyRules.

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 accounting Capability, which this AccountingECAPolicyRule can advertise.

### A.4. TrafficInspectionECAPolicyRuleClass Definition

The purpose of a TrafficInspectionECAPolicyRule is to define an I2NSF 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:

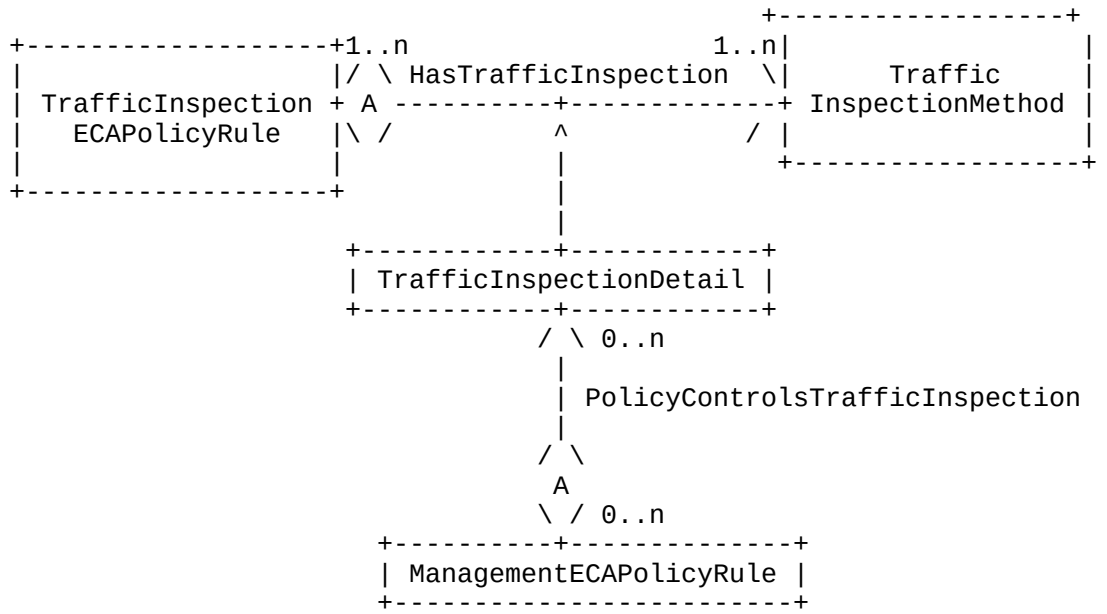


Figure 13. Modeling Traffic Inspection Mechanisms

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 described below.

Figure 13 defines an aggregation between the TrafficInspectionECAPolicyRule and an external class that defines one or more traffic inspection mechanisms. This decouples the implementation of traffic inspection mechanisms from how traffic inspection mechanisms are managed and 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 Policy Rules to control the configuration of the TrafficInspectionDetail association class. This enables the entire traffic inspection process to be managed by ECA PolicyRules.

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 I2NSF 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:

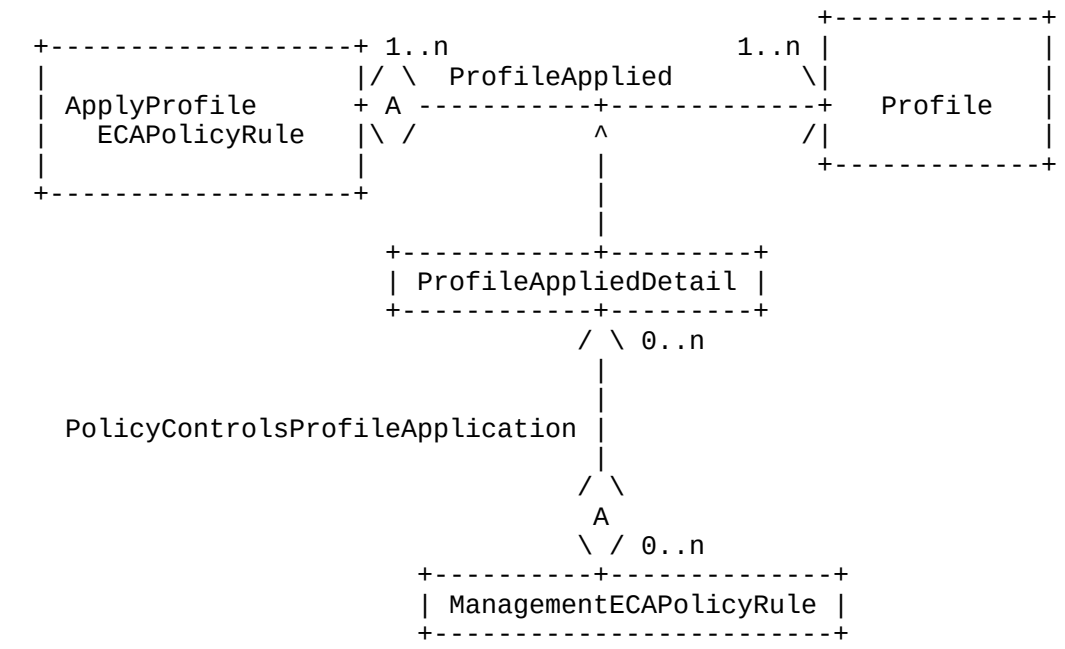


Figure 14. Modeling Profile ApplicationMechanisms

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 described below.

Figure 14 defines an aggregation between the ApplyProfileECAPolicyRule and an external Profile class. 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 Profile is 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 and used by ECA PolicyRules.

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 I2NSF 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:

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 described below.

Figure 15 defines an aggregation between the ApplySignatureECAPolicyRule and an external Signature object class. 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.

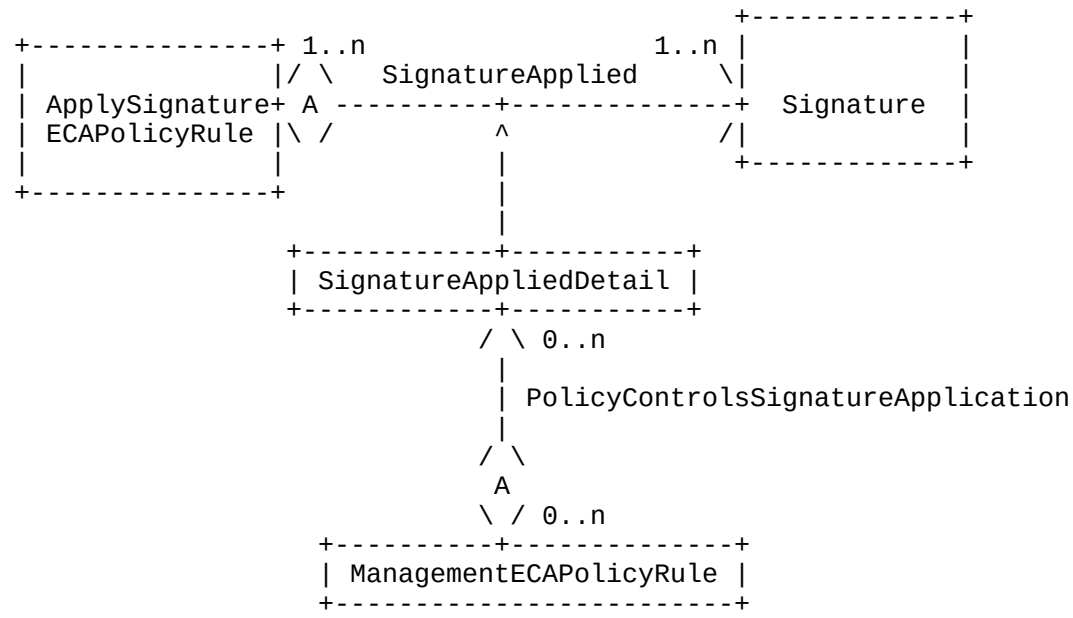


Figure 15.  Modeling Sginature Application Mechanisms

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.

# Appendix B. Network Security Event Class Definitions

This Appendix defines a preliminary set of Network Security Event classes, along with their attributes.

## B.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.

### B.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).

### B.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)

### B.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

### B.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.

#### B.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).

#### B.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)

### B.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.

### B.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.

### B.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 (which are defined in X.733) include:

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

## B.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.

### B.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).

### B.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)

### B.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

## B.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.

### B.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).

### B.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.

### B.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.

## Appendix C. Network Security Condition Class Definitions

This Appendix defines a preliminary set of Network Security Condition classes, along with their attributes.

### C.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 act on different types of packet formats. Its subclasses are shown in Figure 16, and are defined in the following sections.

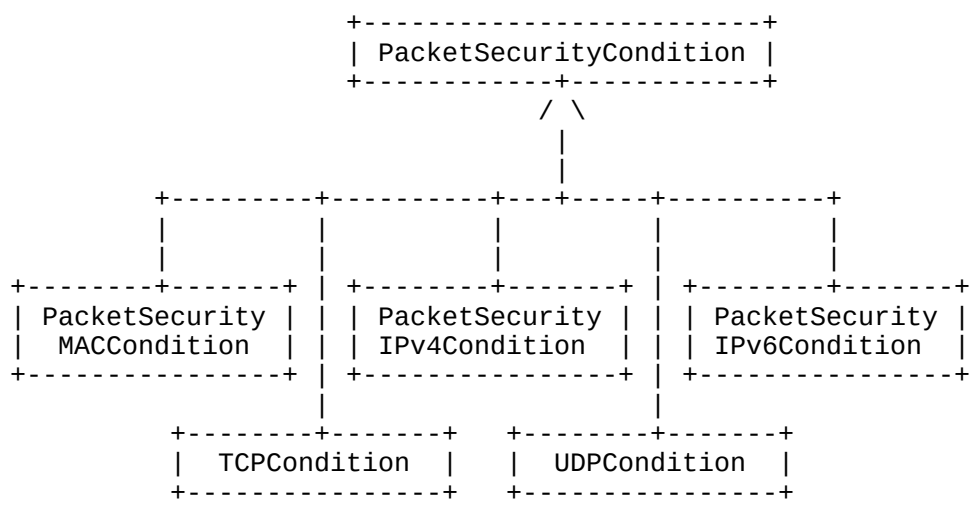


Figure 16. Network Security Info Sub-Model PacketSecurityCondition Class Extensions

#### C.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.

##### C.1.1.1. The pktSecCondMACDest Attribute

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

##### C.1.1.2. The pktSecCondMACSrc Attribute

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

#### C.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.

#### C.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.

#### C.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).

### C.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.

#### C.1.2.1. The pktSecCondIPv4SrcAddr Attribute

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

#### C.1.2.2. The pktSecCondIPv4DestAddr Attribute

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

#### C.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).

#### C.1.2.4. The pktSecCondIPv4DSCP Attribute

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

#### C.1.2.5. The pktSecCondIPv4ECN Attribute

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

#### C.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).

#### C.1.2.7. The pktSecCondIPv4TTL Attribute

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

### C.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.

#### C.1.3.1. The pktSecCondIPv6SrcAddr Attribute

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

#### C.1.3.2. The pktSecCondIPv6DestAddr Attribute

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

#### C.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.

#### C.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.

#### C.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).

#### C.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).

#### C.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).

#### C.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).

### C.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.

#### C.1.4.1. The pktSecCondTPCSrcPort Attribute

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

#### C.1.4.2. The pktSecCondTPCDestPort Attribute

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

#### C.1.4.3. The pktSecCondTCPSeqNum Attribute

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

#### C.1.4.4. The pktSecCondTCPFlags Attribute

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

### C.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.

#### C.1.5.1.1. The pktSecCondUDPSrcPort Attribute

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

#### C.1.5.1.2. The pktSecCondUDPDestPort Attribute

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

#### C.1.5.1.3. The pktSecCondUDPLength Attribute

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

### C.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.

### C.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

### C.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.

### C.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.

### C.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.

# Appendix D. Network Security Action Class Definitions

This Appendix defines a preliminary set of Network Security Action classes, along with their attributes.

## D.1. IngressAction

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

## D.2. EgressAction

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

## D.3. ApplyProfileAction

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

## Appendix E. Geometric Model

The geometric model defined in [Bas12] is summarized here. Note that our work has extended the work of [Bas12] to model ECA Policy Rules, instead of just condition-action Policy Rules. However, the geometric model in this Appendix is simplified in this version of this I-D, and is used to define just the CA part of the ECA model.

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 or 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 example, AH-transport mode and ESP-transport mode (and combinations thereof) are a more precise definition of channel protection actions.

Conditions are typed predicates concerning a given selector. A selector describes the values that a protocol field may take. For example, 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 17 the conditions are s1 <= S1 (read as s1 subset of or equal to S1) and s2 <= S2 (s2 subset of or equal to S2), both s1 and s2 match the packet x1, while only s2 matches 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.

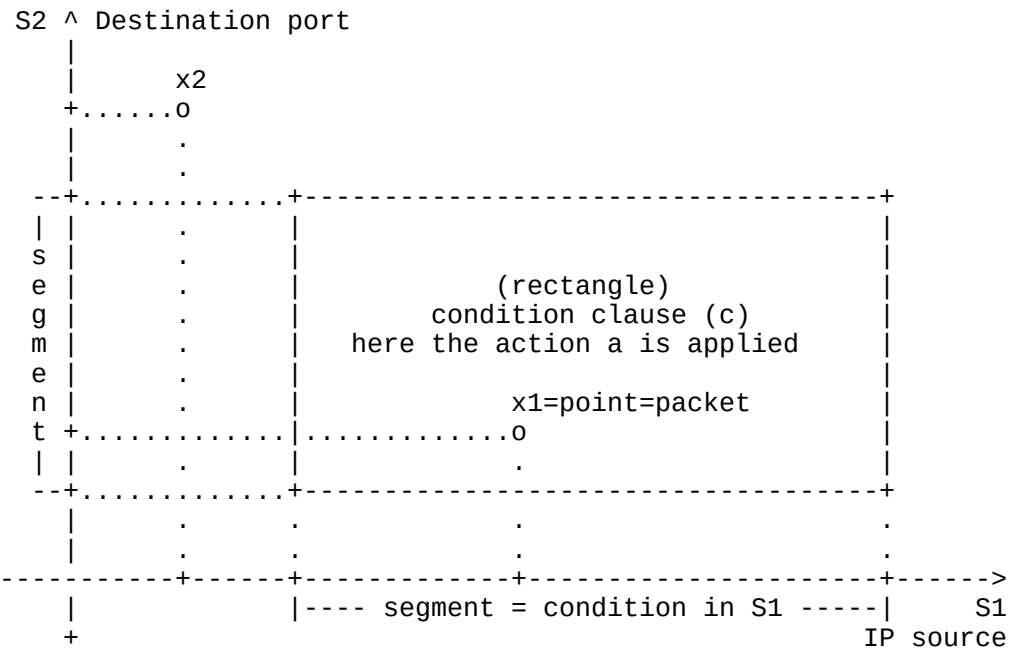


Figure 17: Geometric representation of a rule r=(c,a) that matches x1, but does not match x2.

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 the union of hyper-rectangles (as they are Cartesian product of conditions), are valid. This is an intrinsic constraint of the policy language, as it specifies 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 17 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 match a packet (rules match a packet), if all the conditions forming the clauses match the packet. In Figure 17, 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(.):

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

where E={fj:R -> Xj} (j=1,2,...) is the set that includes all 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 as:

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.

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

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

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