

I2NSF
Internet-Draft
Intended status: Experimental
Expires: January 9, 2017

C. Basile
PoliTo
D. Lopez
TID
July 8, 2016

A Model of Security Capabilities for Network Security Functions
draft-baspez-i2nsf-capabilities-00

Abstract

This document presents a model of Security Capabilities. Security Capabilities are intended to describe the potentiality that Network Security Functions (NSFs) have for security policy enforcement purposes. Therefore, Security Capabilities are represented as abstract functionalities that a NSF owns in terms of enforcement actions, conditions that can apply in order to determine to which packet or traffic enforce the actions, and other mechanisms that NSF use to determine the actions to enforce. The proposed capability model defines without ambiguities the operations a function can do in term of security policy enforcement.

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 9, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|--|----|
| 1. Introduction | 3 |
| 2. Requirements Language | 3 |
| 3. Policy model | 4 |
| 3.1. Geometric model of policies | 4 |
| 3.2. Condition types | 7 |
| 4. Capability Model | 8 |
| 4.1. Algebra of capabilities | 9 |
| 5. References | 10 |
| 5.1. Normative References | 10 |
| 5.2. Informative References | 11 |
| Authors' Addresses | 11 |

1. Introduction

Security Capabilities are intended to describe the potentiality that Network Security Functions (NSFs) have for security policy enforcement purposes. Security Capabilities are abstract concepts that are independent of the actual security control that will implement them. However, every NSF will be associated to the capabilities it owns. Security Capabilities are required to allow interoperability among network functions. It would be a market enabler having a way to substitute a NSF with an equivalent one (i.e., having the same functionality). Moreover, Security Capabilities are very useful to reason about generic functions, which may be needed at design time. That is, it is not needed to refer to a specific product when designing the network, rather the functions characterized by their capabilities are considered.

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

The Capability Model has been designed to support at least capability matching, i.e., to identify the NSFs in a catalog that can perform the operations required to enforce a high-level policy.

Moreover, the Capability Model has been preliminarily validated by verifying that it allows the correct description of several existing security controls.

2. Requirements Language

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 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

3. Policy model

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

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

3.1. Geometric model of policies

We refer in this paper to the policy model defined in [Bas12] as geometric model, which is summarized here. Policies are specified by means of a set of rules in the "if condition then action" format [RFC3198]. Rules are formed by a condition clause and an action clause. 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 Allow and Deny, thus $A=\{\text{Allow},\text{Deny}\}$. For channel protection controls, they may be informally written as "enforce confidentiality", "enforce data authentication and integrity", and "enforce confidentiality and data authentication and integrity". However, these actions need to be instantiated to the technology used, for instance AH-transport mode and ESP-transport mode (and combinations thereof) are a more precise and univocal definition of channel protection actions.

Conditions are typed predicates concerning a given selector. A selector describes the values that a protocol field may take, e.g.,

the IP source selector is the set of all possible IP addresses, and it may also refer to the part of the packet where the values come from, e.g., the IP source selector refers to the IP source field in the IP header. Geometrically, a condition is the subset of its selector for which it evaluates to true. A condition on a given selector matches a packet if the value of the field referred to by the selector belongs to the condition. For instance, in Figure 1 the conditions are $s1 \subseteq S1$ (read as $s1$ subset of $S1$) and $s2 \subseteq S2$ ($s1$ subset of $S2$), both $s1$ and $s2$ match the packet $x1$, while only $s2$ matches $x2$.

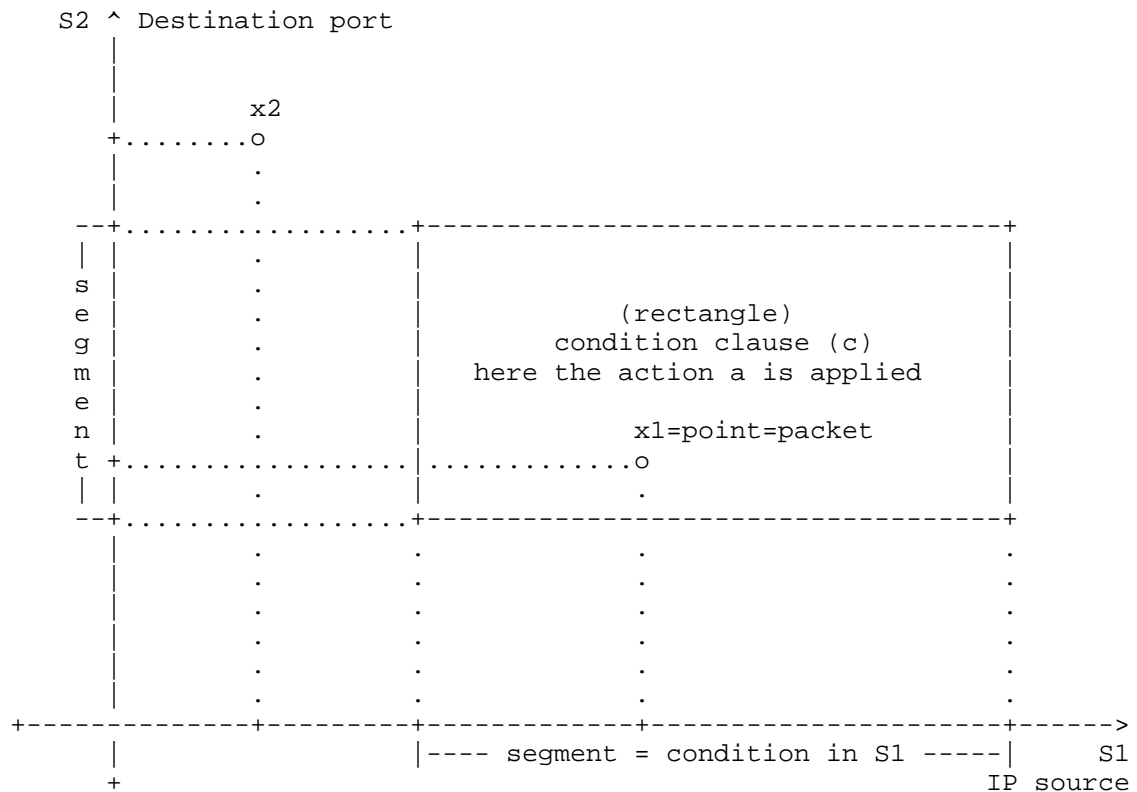


Figure 1: Geometric representation of a rule $r=(c,a)$ that matches $x1$ but does not match $x2$.

To consider conditions in different selectors, the decision space is extended using the Cartesian product because distinct selectors refer to different fields, possibly from different protocol headers. Hence, given a policy-enabled element that allows the definition of conditions on the selectors $S1, S2, \dots, S_m$ (where m is the number of

selectors available at the security control we want to model), its selection space is:

$$S = S_1 \times S_2 \times \dots \times S_m$$

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

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

$$c = s_1 \times s_2 \times \dots \times s_m \quad [\quad S_1 \times S_1 \times S_2 \times \dots \times S_m = S$$

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

In the geometric model, a rule is expressed as $r=(c,a)$, where $c \in S$ (the condition clause is a subset of the selection space), and the action a belongs to A . A condition clause of a rule matches a packet, or briefly a rule matches a packet, if all the conditions forming the clause match the packet: in Figure 1, the rule with condition clause c matches the packet x_1 but not x_2 .

The rule set R is composed of n rules $r_i=(c_i,a_i)$.

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

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

Resolution strategies may use, besides intrinsic rule data (i.e., condition clause and action clause), also ``external data`` associated to each rule, such as priority, identity of the creator,

and creation time. Formally, every rule ri is associated by means of the function $e(.)$ to:

$$e(ri) = (ri, f1(ri), f2(ri), \dots)$$

where $E = \{fj: R \rightarrow Xj\}_{j \in J}$ is the set that includes all the functions that map rules to external attributes in Xj . However, E , e , and all the Xj are determined by the resolution strategy used.

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

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 modelling events like an additional selector.

3.2. Condition types

After having analysed the literature and the existing security controls, we have categorized the types of selectors in exact match, range-based, regex-based, and custom-match [Bas15][Lunt].

Exact match selectors are (unstructured) sets: elements can only be checked for equality, as no order is defined on them. As an example, the protocol type field of the IP header is a unordered set of integer values associated to protocols.

Range-based selectors are ordered sets where it is possible to naturally specify ranges as they can be easily mapped to integers. As an example, the ports in the TCP protocol are well represented using a range-based selector (e.g., 1024-65535). We include in the range-based selectors all the category of selectors that have been defined by Al-Shaer et al. as prefix match [Alshaer]. These selectors allow the specification of ranges of values by means of simple regular expressions. The typical case is the IP address selector (e.g., 10.10.1.*). There is no need to distinguish between prefix match and range-based selectors as 10.10.1.* easily maps to [10.10.1.0, 10.10.1.255].

Another category of selector types includes the regex-based selectors, where the matching is performed by using regular

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

Finally, we introduce the idea of custom check selectors. For instance the malware analysis looks for specific patterns and returns a Boolean value is an example of custom check selector, if the logic of checking is not seen (nor really interesting) from the outside. In order to be properly used by high-level policy based processed (like reasoning systems, refinement systems) these custom check selector need at least to be described as black-boxes, that is, the list of fields that they process (inputs) in order to return the Boolean verdict (output).

4. Capability Model

Our model of capabilities is based on actions and traffic classification features. Indeed, the need for enforcing one of the actions that a security control can apply to packets/flows is the main reason to use a security control. Moreover, security controls have classification features that permit the identification of the target packets/flows of the actions enforced, i.e., the selectors presented in Section 3.1. A security manager decides for a specific security control depending on the actions and classification features. If the security control can enforce the needed actions and has the classification features needed to identify the packets flows required by a policy, then the security control is capable of enforcing the policy. Our refinement model needs to know NSF's capabilities to perform its operations.

However, security controls may have specific characteristics that automatic processes or administrators need to know when they have to generate configurations, like the available resolution strategies and the possibility to set default actions. We have ignored, to simplify this presentation, options to generate configurations that may have better performance, like the use of chains or ad hoc structures [Taylor]. Adding support to these forms of optimization is certainly feasible with a limited effort but it was outside the scope of this paper, that is, to show that adding security awareness to NFV

management and orchestration features is possible. It is one of the task for future work.

Capabilities can be used for two purposes: describing generic security functions, and describing specific products. With the term generic security function (GNSF) we denote known classes of security functions. The idea is to have generic components whose behaviour is as well understood as for the network components (i.e., a switch is a switch and we know to use it even if it may have some vendor-specific functions). These generic functions can be substituted by any product that owns the required capability at instantiation time.

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

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

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

4.1. Algebra of capabilities

Our capabilities are defined by a 4-tuple:

$$(Ac; Cc; RSc; Dc) \mid (AC; CC; RSC; DC) = K$$

where AC is the set of all the supported actions, CC is set of all the supported conditions types, RSC is the set of all the supported resolutions strategies, and Dc [DC={F} U A, where F indicates that the default action is supported and can be freely selected by the policy editor, and explicitly indicates the default action if this cannot be explicitly configured.

We defined a syntax to specify:

- o subsets of actions: $Ac=\{a:action1, a:action2, \dots\} \mid AC$
- o subset of conditions: $Cc=\{c:cond1, c:cond2, \dots\} \mid CC$
- o subset of resolution strategies $RSc=\{rs:res1, rs:res2, \dots\} \mid RSC$

Given $cap1=(Ac1,Cc1,RSc1,def1)$ and $cap2=(Ac2,Cc2,RSc2,def2)$, we define

- o capability addition: $cap1+cap2 = (Ac1 \cup Ac2, Cc1 \cup Cc2, RSc1, def1)$
- o capability subtraction: $cap_1-cap_2 = ({Ac1 \setminus Ac2, Cc1 \setminus Cc2, RSc1, def1})$

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

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

- o $Apf = \{a:Allow, a:Deny\}$
- o $Cpf = \{c:IPsrc, c:IPdst, c:Psrc, c:Pdst, c:protType\}$
- o $Ctime = \{c:timestart, c:days, c:datestart, c:datestop\}$
- o $cap_pf=(Apf; Cpf; \{FMR\}; F)$
- o $cap_pf+time=cap_pf + Ctime$

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

5. References

5.1. Normative References

- [I-D.ietf-i2nsf-framework] elopez@fortinet.com, e., Lopez, D., Dunbar, L., Strassner, J., Zhuang, X., Parrott, J., Krishnan, R., and S. Durbha, "Framework for Interface to Network Security Functions", draft-ietf-i2nsf-framework-02 (work in progress), July 2016.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

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

5.2. Informative References

- [Alshaer] Al Shaer, E. and H. Hamed, "Modeling and management of firewall policies", 2004.
- [Bas12] Basile, C., Cappadonia, A., and A. Liroy, "Network-Level Access Control Policy Analysis and Transformation", 2012.
- [Bas15] Basile, C. and A. Liroy, "Analysis of application-layer filtering policies with application to HTTP", 2015.
- [Cormen] Cormen, T., "Introduction to Algorithms", 2009.
- [Lunt] van Lunteren, J. and T. Engbersen, "Fast and scalable packet classification", 2003.
- [Taylor] Taylor, D. and J. Turner, "Scalable packet classification using distributed crossproducting of field labels", 2004.

Authors' Addresses

Cataldo Basile
Politecnico di Torino
Corso Duca degli Abruzzi, 34
Torino, 10129
Italy

Phone: +39 011 090 7173
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

