An algorithm for security policy migration in multiple firewall networks

Manuel Cheminod, Luca Durante, Lucia Seno and Adriano Valenzano

National Research Council of Italy, CNR-IEIIT, C.so Duca degli Abruzzi 24, 10129 Torino, Italy

Abstract

Firewalls are effectively employed to protect network portions by blocking illegitimate traversing traffic. However, during traffic load peaks, possibly due to DoS-like attacks, they may become performance bottlenecks, introducing consistent delays/losses on legitimate packets. In multiple firewall networks, a cooperative approach to mitigate performance degradation caused by firewall overloads consists in suitably distributing responsibility for security policy implementation among available devices to balance workload. We present a technique for migrating security policies among firewalls in a sequence, formally verified to preserve the overall security policy implemented by the sequence itself. The technique can be used as building block in the development of cooperative solutions allowing to balance workload in networks where firewalls are arbitrarily placed to guard specific domains.

Keywords

Firewalls, Network security, Policy migration, Formal methods

1. Introduction

Nowadays firewalls are pervasively deployed over the Internet to protect network portions (or single devices) against undesired/malicious traffic possibly threatening confidentiality, integrity and availability of the provided services. Firewalls implement security policies, which, operatively, translate in lists of rules, each consisting of a *condition* defined over some packet header fields (*filtering fields*), and an *action*. Firewalls operating according to the *first matching strategy* check incoming packets against rule conditions sequentially, until the *matching rule* (i.e., the first rule in the list whose condition is satisfied by the packet) is found, and then apply the corresponding action (i.e., typically $allow$ or $deny$).

Firewall operation also impacts the legitimate traffic and, especially during traffic load peaks (when the average packet arrival rate exceeds the firewall average service rate), legitimate packets may suffer high delays and even be lost before being processed because of overloads and overflows [1, 2]. The time necessary to a firewall for processing a packet is mainly determined by the packet *matching time*, which is proportional to the number of rules against which the packet is checked before finding the matching one. In general, the higher the *firewall cardinality* (i.e., the number of rules in the list), the higher the firewall average service time. This criticality is exploited, for example, by DoF (Denial of Firewalling) attacks which generate traffic targeting the last rules in firewall lists [3].

ITASEC 2021, Italian Conference on CyberSecurity, April 07-09, 2021

manuel.cheminod@ieiit.cnr.it (M. Cheminod); luca.durante@ieiit.cnr.it (L. Durante); lucia.seno@ieiit.cnr.it (L. Seno); adriano.valenzano@ieiit.cnr.it (A. Valenzano)

This paper presents a technique which may be used as a building block towards the development of a cooperative solution, mitigating network performance degradation caused by firewalls during traffic load peaks (either naturally arising or caused by cyber-attacks). The idea is, when a firewall is overloaded, to split the security policy it implements so that part of it is cooperatively implemented by other, less loaded, firewalls within the protected network. Reducing the cardinality of the overloaded firewall, at the expense of that of the cooperating ones, allows to balance the workload among the available devices. The presented technique allows to migrate security policies along firewalls in a sequence, without altering the overall security policy implemented by the sequence itself, by translating them in lists of only $deny$ rules. The technique can be extended to more general configurations, where firewalls are not connected in sequences but arbitrarily placed to guard specific network domains. Note that, security administrators are only supposed to act (maintain/modify) on firewall configurations in their original form, while the proposed technique operates independently, ensuring that the overall network security policy is not changed in any way.

The paper is organized as follows: Sect. 2 provides an overview of state of the art techniques for firewall performance optimization, Sect. 3 formalizes notation and considered problem, while Sect. 4 describes the proposed technique for security policy migration. Sect. 5 concludes the paper and, finally, App. A collects proofs.

2. Related works

Several techniques have been proposed to mitigate firewall impact on legitimate traffic. Some of them aim at minimizing firewall cardinality without changing the implemented security policy (e.g., policy analysis [4, 5, 6, 7], firewall compression [8, 9]), and are typically run offline, before actual firewall configuration. Others, lean on the observation that traffic characteristics remain constant for long periods of time and aim at ordering firewall rules so that those with higher matching probability are at the top of the list, allowing the majority of packets to be checked against a small number of rules (rule ordering [10, 11, 12, 13]). The latter are run online, any time a change in the traffic profile is detected. Although effective, these techniques may not be sufficient to avoid firewall overloads and overflows, especially during traffic load peaks. Indeed, complex security policies may result in hundreds of rules even after analysis/compression. Moreover, compression often leads to lists of highly dependent rules which cannot easily be reordered, limiting the improvements achievable by compression and ordering combined. Finally, if the packet arrival rate increases but the distribution of matching probability among firewall rules does not change consistently, rule ordering is not helpful.

The technique we propose can be used in combination with all the previously mentioned ones but follows a different approach, i.e., like [14, 15, 16], exploits the presence of multiple firewalls within the protected network. In [14] a technique to balance workload by dynamically distributing rules among firewalls in a network is described. However, to increase the degrees of freedom in rule deployment, the technique relies on some unused IP header fields, written and checked by firewalls, allowing packets only to be filtered by a fraction of the firewalls they traverse. In [15], a solution for migrating rules from a central firewall to decentralized micro-firewalls in cloud/cloudlets architectures is presented. However, to preserve the overall security policy implemented in the network, the solution requires a rearrangement of traffic paths after rule migration. Differently from [14] and [15], the technique described in this paper does not imply any modification to firewall or network behavior. The proposed solution resembles the one in [16], in that it allows to migrate (part of) the security policy originally implemented by a firewall to downstream ones in the protected network, preserving the overall network security policy. Solution in [16], however, relies on the *go to* action, which allows packets to jump among firewall rules instead of being checked against their conditions sequentially. This paper extends [16], as the proposed technique does not rely on the *go to* construct, and can thus be used in combination with policy analysis, rule ordering and compression algorithms, which are traditionally developed for security policies not including jumps.

3. Notation and problem definition

In this section, we define the notation used through the paper, which partially relies on the ones in [8, 16], and formalize the tackled problem.

3.1. Firewalls and firewall sequences

We call *range* any non-empty, finite set of consecutive non-negative integers, $[a, b] \subseteq \mathbb{N}_0$ (with $\mathbb{N}_0$ set of natural numbers including zero). Note that $[a, a]$ is range only consisting of $a \in \mathbb{N}_0$. Firewalls operating a the IP layer filter packets based on the content of five header fields, i.e., source and destination IP address, source and destination port address and protocol number. Formally, a filtering field P_i is a range, and source and destination IP address, source and destination port address and protocol number are defined, respectively, as ranges $P_{1,2} = [0, 2^{32} - 1]$, $P_{3,4} = [0, 2^{16} - 1]$, and $P_5 = [0, 2^8 - 1]$. Even if we focus on IP-layer firewalls, in the following, we refer to a generic number n of filtering fields.

A packet p defined over fields P_i, $i = 1, \dots, n$, is a n-tuple

$$p = (p_1, p_2, \dots, p_n),\ p_i \in P_i,\ i = 1, \dots, n \tag{1}$$

A packet p is a point in $\mathbb{N}_0^n$, and the finite set of all possible packets defined over fields P_i, $i = 1, \dots, n$, $\mathcal{P} = P_1 \times P_2 \times \dots \times P_n \subseteq \mathbb{N}_0^n$, is a hyperrectangle (the generalization of a rectangle in a n-dimensional space) in $\mathbb{N}_0^n$. Note that, the Cartesian product of n ranges is a set of n-tuple of integers, i.e., of packets defined over n filtering fields.

A condition defined over fields P_i, $i = 1, \dots, n$, is a n-tuple

$$c = (C_1, C_2, \dots, C_n) \tag{2}$$

where for $i = 1, \dots, n$, C_i is a range such that $C_i \subseteq P_i$.

A rule r over fields P_i, $i = 1, \dots, n$, is defined as

$$r = \langle c, action \rangle \tag{3}$$

where c is a condition over fields P_i, $i = 1, \dots, n$ and $action$ varies in the set of all possible firewall actions, i.e., $action \in \{allow, deny\}$. From now on, we assume all packets, conditions and rules to be defined over the same filtering fields P_i, $i = 1, \ldots, n$. We use symbol $*$ as a possible value for condition components, i.e., $C_i = *$ means $C_i = P_i$, and the traditional dotted decimal notation for single or ranges of IP addresses. Note, however, that statements like $192.168.*.1$ do not define single ranges and are not admissible values for condition components.

A packet $p = (p_1, p_2, \ldots, p_n)$ is said to *match* a rule $r = \langle (C_1, C_2, \ldots, C_n), action \rangle$ if and only if $p_i \in C_i$, $i = 1, \ldots, n$, i.e., $p_i \in \mathcal{S}_r = C_1 \times C_2 \times \ldots \times C_n \subseteq \mathcal{P}$. Packet set $\mathcal{S}_r$, defined by rule r, is also a hyperrectangle in $\mathbb{N}_0^n$.

A firewall fw (see Fig. 1a, where, as in other examples in the paper, condition/action-rule correspondence has been made explicit through superscripts) is a tuple of rules

$$fw = (r_1, r_2, \ldots, r_{|fw|}) \tag{4}$$

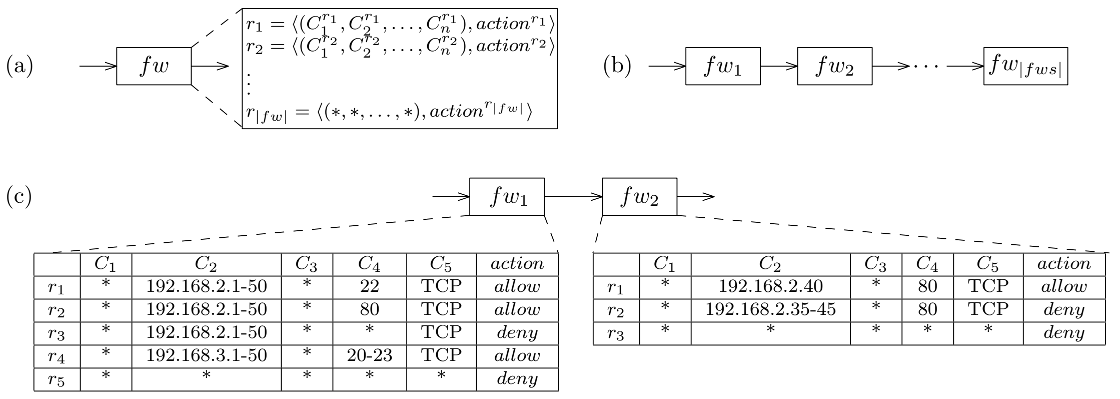
where $|fw|$ is the firewall cardinality. The operation performed by a firewall acting according to the first matching strategy, consists in checking any incoming packet against its rule conditions, sequentially, until the matching rule is found. Then, the matching rule action is applied to the packet, that can be either forwarded ($allow$), or discarded ($deny$). A packet may match multiple, possibly conflicting (i.e., characterized by different actions), rules within a firewall (*dependent rules*), in this case the packet fate is determined by the rule order. Note that, we currently focus only on filtering operation, while we do not model other possible firewall functions, e.g., NAT (Network Address Translation).

A firewall fw is said to be *complete* if any packet $p \in \mathcal{P}$ matches at least one rule in fw. We only consider complete firewalls and, to make sure of this, we assume a firewall last rule to be always $r_{|fw|} = \langle (*, *, \ldots, *), action \rangle$, matched by any packet $p \in \mathcal{P}$. Under the completeness hypothesis, any firewall fw defines a *filtering function* $f_{fw} : \mathcal{P} \cup \{-\} \to \mathcal{P} \cup \{-\}$, that maps any packet $p \in \mathcal{P}$ either in itself, if the packet is forwarded, or in $-$, the null packet, if it is discarded by fw. Note that, the domain of f_{fw} has been extended to the null packet, defining $f_{fw}(-) = -$ for any firewall. Firewalls characterized by different rule lists may define the same filtering function. In particular, two firewalls fw_1 and fw_2 are said to be equivalent over $\mathcal{P}$, or just *equivalent* ($fw_1 \equiv fw_2$), if they equally map all packets, i.e., if $\forall p \in \mathcal{P}$, $f_{fw_1}(p) = f_{fw_2}(p)$.

A firewall sequence (see Fig.1b) is a tuple of firewalls:

$$fws = (fw_1, fw_2, \ldots, fw_{|fws|}) \tag{5}$$

where the number of firewalls $|fws|$ is called *sequence cardinality*. The cumulative filtering operation performed by a firewall sequence (i.e., that defining whether a traversing packet is forwarded at the end of the sequence or discarded by one of the firewall within) can be equated to that of a single firewall. For this reason, analogously to single firewalls, any firewall sequence fws defines a filtering function $f_{fws} : \mathcal{P} \cup \{-\} \to \mathcal{P} \cup \{-\}$, which corresponds to the inverted order composition of the filtering functions defined by the firewalls of the sequence, i.e., which maps any packet $p \in \mathcal{P}$ in $f_{fw_{|fws|}} \circ \ldots \circ f_{fw_2} \circ f_{fw_1}(p)$. The definitions of equivalence provided for single firewalls can then be extended to the firewall sequence domain, i.e., two firewall sequences fws_1 and fws_2 are said to be equivalent ($fws_1 \equiv fws_2$) if $\forall p \in \mathcal{P}$, $f_{fws_1}(p) = f_{fws_2}(p)$. Note that a single firewall can be seen as a firewall sequence with only one component. Moreover, as described in detail later, in general, a firewall sequence is not equivalent to a single firewall orderly listing the rules of all firewalls in the sequence.


(a)
fw
$r_1 = \langle (C_1^{r_1}, C_2^{r_1}, \ldots, C_n^{r_1}), action^{r_1} \rangle$
$r_2 = \langle (C_1^{r_2}, C_2^{r_2}, \ldots, C_n^{r_2}), action^{r_2} \rangle$
$\vdots$
$r_{|fw|} = \langle (*, *, \ldots, *), action^{r_{|fw|}} \rangle$
(b)
fw_1
fw_2
$\cdots$
$fw_{|fws|}$
(c)
fw_1
fw_2

	C_1	C_2	C_3	C_4	C_5	*action*
r_1	*	192.168.2.1-50	*	22	TCP	*allow*
r_2	*	192.168.2.1-50	*	80	TCP	*allow*
r_3	*	192.168.2.1-50	*	*	TCP	*deny*
r_4	*	192.168.3.1-50	*	20-23	TCP	*allow*
r_5	*	*	*	*	*	*deny*

	C_1	C_2	C_3	C_4	C_5	*action*
r_1	*	192.168.2.40	*	80	TCP	*allow*
r_2	*	192.168.2.35-45	*	80	TCP	*deny*
r_3	*	*	*	*	*	*deny*

Figure 1: Examples of firewalls and firewall sequences.

3.2. Problem statement

We consider the problem of moving the security policy implemented by a firewall, so that it is implemented by a downstream firewall in the same sequence, without changing the overall security policy implemented by the sequence itself. Specifically, we focus on network configurations like the one in Fig. 1c, and consider the following problem:

Problem 1. *Given a firewall sequence $fws = (fw_1, fw_2)$, where $fw_i = (r_1^{fw_i}, r_2^{fw_i}, \ldots, r_{|fw_i|}^{fw_i})$, $i = 1, 2$, how to compute a firewall $\overline{fw}_2$ such that sequence $\overline{fws} = (\overline{fw}_1, \overline{fw}_2)$, where $\overline{fw}_1$ is the trivial firewall (i.e., the firewall that forwards every packet, $\overline{fw}_1 = (\langle (*, *, \ldots, *), allow \rangle)$), satisfies the following property:*

$$\overline{fws} \equiv fws \tag{6}$$

Prob. 1 is a simplified version of the problem we are actually interested in, and allows for an easier technique description. I.e., the ultimate goal is to make fw_2 (and/or other firewalls within the protected network) responsible for implementing only part of the security policy originally implemented by fw_1, to balance workload among available devices. Also, we are interested in realistic configurations, where firewalls are not connected in sequences but arbitrarily placed to guard specific network domains. Finally, Prob. 1 only considers *downstream* policy migration, where the overloaded firewall is the one firstly traversed by traffic. Although an upstream firewall is typically traversed by more packets and thus more likely to suffer from overloads, it is not always so. Situations described above are variations to Prob. 1, and can be solved by extending its solution, i.e., the significance of the proposed technique goes beyond Prob. 1.

Note that Prob. 1 is not trivial, e.g., defining $\overline{fw}_2$ by moving rules in fw_1 to the top of fw_2 is not an admissible solution. Indeed, migrating $allow$ rules, in general, makes (6) not satisfied. In fws, a packet matching an $allow$ rule in fw_1, is forwarded by the first firewall, but is still subjected to the filtering operation of (i.e., can still be dropped by) fw_2. In $\overline{fws}$, where $\overline{fw}_2$ is defined by listing rules in fw_1 before those in fw_2, the same packet is surely forwarded, as the trivial firewall $\overline{fw}_1$ forwards every packet and the matching rule of the packet in $\overline{fw}_2$ is the same migrated $allow$ rule matched by the packet in fw_1. Thus, the packet is not checked against rules originally in fw_2, possibly violating (6).

4. Security policy migration technique

In this section, we describe the proposed technique to solve Prob. 1, and the algorithms defined to implement it.

4.1. A solution to Problem 1

The proposed security policy migration technique is based on the observation that a packet whose matching rule in fw_1 is a *deny* rule, independently of its matching rule in fw_2, is dropped by fws and, for (6) to hold, should be dropped by $\overline{fws}$ as well. As a consequence, if we compute a set of *deny* rules, which discards all and only packets originally discarded by fw_1, we can move list these rules before those in fw_2 to obtain $\overline{fw}_2$ required by Prob. 1. Formally:

Theorem 1. *Given Prob. 1, if a set of deny rules $r_1, r_2, \ldots, r_k$ satisfies property*

$$\bigcup_{j=1}^{k} \mathcal{S}_{r_j} = \mathcal{D}_{fw_1} = \{p \in \mathcal{P} \mid f_{fw_1}(p) = -\}, \tag{7}$$

where $\mathcal{S}_{r_j}$ is packet set defined by rule r_j and $\mathcal{D}_{fw_1}$ is the set of packets discarded by fw_1, then

$$\overline{fw}_2 = (r_1, \ldots, r_k, r_1^{fw_2}, \ldots, r_{|fw_2|}^{fw_2}) \tag{8}$$

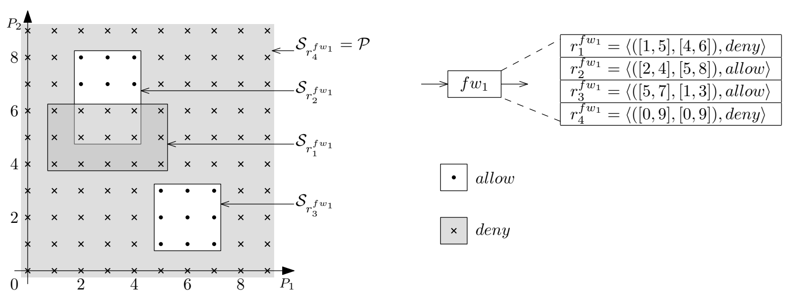
is a solution to Prob. 1. (Proof of Thm. 1 is provided in App. A.1.)

Note that, in Thm. 1, we refer to a set of rules, instead of a list, as rules $r_1, r_2, \ldots, r_k$ all have the same action (*deny*) and thus their relative order in (8) does not matter.

We now address how to compute a set of *deny* rules $r_1, r_2, \ldots, r_k$ satisfying (7). By definition of firewall operation, $\mathcal{D}_{fw_1}$ consists of packets having as matching rule in fw_1 a *deny* rule, i.e.,

$$\mathcal{D}_{fw_1} = \{p \in \mathcal{P} \mid \exists\, \ell \in L,\ \ p \in \mathcal{S}_{r_\ell^{fw_1}},\ \ \forall m \in I, m < \ell,\ \ p \notin \mathcal{S}_{r_m^{fw_1}}\} \tag{9}$$

where I and L are the sets of indexes, respectively, of all rules and *deny* rules in fw_1. From (9), by applying generic set properties, $\mathcal{D}_{fw_1}$ can be expressed in term of packet sets defined by



P_2
P_1
0
2
4
6
8
$\mathcal{S}_{r_4^{fw_1}} = \mathcal{P}$
$\mathcal{S}_{r_2^{fw_1}}$
$\mathcal{S}_{r_1^{fw_1}}$
$\mathcal{S}_{r_3^{fw_1}}$
fw_1
$r_1^{fw_1} = \langle([1,5],[4,6]), deny\rangle$
$r_2^{fw_1} = \langle([2,4],[5,8]), allow\rangle$
$r_3^{fw_1} = \langle([5,7],[1,3]), allow\rangle$
$r_4^{fw_1} = \langle([0,9],[0,9]), deny\rangle$
allow
deny

Figure 2: Hint on security policy migration for $n = 2$.

rules in fw_1 as (proof in App. A.2)

$$\mathcal{D}_{fw_1} = \bigcup_{\ell \in L} \left(\mathcal{S}_{r_\ell^{fw_1}} \setminus \left(\bigcup_{m \in M, m < \ell} \mathcal{S}_{r_m^{fw_1}} \right) \right) \tag{10}$$

$$= \bigcup_{\ell \in L} \left(\left(\dots \left(\left(\mathcal{S}_{r_\ell^{fw_1}} \setminus \mathcal{S}_{r_{m_1}^{fw_1}} \right) \setminus \mathcal{S}_{r_{m_2}^{fw_1}} \right) \dots \right) \setminus \mathcal{S}_{r_{m_{|M_\ell|}}^{fw_1}} \right) \tag{11}$$

where, $M = I \setminus L$ is the set of indexes of *allow* rules in fw_1 and, for each $\ell \in L$, $M_\ell = \{m_1, m_2, \dots, m_{|M_\ell|}\} = \{m \in M, m < \ell\}$ is the set of indexes of *allow* rules preceding *deny* rule $r_\ell^{fw_1}$ in $\overline{fw}_1$. Set $\mathcal{D}_{fw_1}$ is expressed in (11) as a union of sets, one for each *deny* rule $r_\ell^{fw_1}$, resulting from sequences of set subtractions, where the result of a subtraction is the minuend of the next one. We want to compute a set of *deny* rules $r_1, r_2, \dots, r_k$ such that (7) holds for $\mathcal{D}_{fw_1}$ described by (11). Note that, while a packet set defined by a rule (or a union of packet sets defined by rules), can be directly translated into the rule (or set of rules) defining it, the set resulting from the subtraction between two packet set defined by rules (although can be always expressed as a union of hyperrectangles) is not necessarily a single hyperrectangle and cannot directly be translated into a rule set defining it. We thus define a *rule subtraction operator*, `sub_rule(`r_d, r_a`)`, which allows to compute the set of rules defining a set expressed by a subtraction between two packet sets defined by rules. In general,

Definition 1. A *rule subtraction operator* is any operator that given rules r_d and r_a, returns a, possibly empty, set $\mathcal{R}$ of rules, with the same action as r_d, such that $\bigcup_{r \in \mathcal{R}} \mathcal{S}_r = \mathcal{S}_{r_d} \setminus \mathcal{S}_{r_a}$.

Once defined `sub_rule(`r_d, r_a`)`, for each $\ell \in L$, rule set $\mathcal{R}_\ell^{m_{|M_\ell|}}$ defining the packet set resulting from the sequence of set subtractions from $\mathcal{S}_{r_\ell^{fw_1}}$ in (11), can be iteratively computed as (proof in App. A.2):

$$\mathcal{R}_\ell^{m_0} = \{r_\ell^{fw_1}\}, \qquad \mathcal{R}_\ell^{m_t} = \bigcup_{r \in \mathcal{R}_\ell^{m_{t-1}}} \texttt{sub\_rule}(r, r_{m_t}^{fw_1}) \quad t = 1, \dots, m_{|M_\ell|} \tag{12}$$

Note that, in (11), for each $\ell \in L$, only the first set subtraction in the sequence is among packet sets defined by rules and (as described in (12)) the set of rules defining it can be computed by using `sub_rule(`r_d, r_a`)` once. The next set subtractions in the sequence require the execution of a rule subtraction for each rule resulting from the previous rule subtraction(s). Rules $r_1, r_2, \dots, r_k$ satisfying (7) are then those in $\bigcup_{\ell \in L} \mathcal{R}_\ell^{m_{|M_\ell|}}$. As required by Thm. 1, they are all *deny* rules, as they result from sequences of rule subtractions from *deny* rules $r_\ell^{fw_1}$.

Fig. 2 shows a simple clarifying example. Rectangles in the figure correspond to rules $r_j^{fw_1}$, $j = 1, \dots, 4$, defined over $n = 2$ filtering fields $P_i = [0, 9]$, $i = 1, 2$ and listed in firewall fw_1. The filtering function defined by fw_1, f_{fw_1}, is represented in Fig. 2 through symbols and colors: dotted white points of $\mathbb{N}_0^2$ correspond to packets forwarded by fw_1, while crossed grey ones to packets dropped by the firewall. Computing a set of *deny* rules $r_1, r_2, \dots, r_k$ satisfying (7) means finding a set of rectangles covering all and only crossed grey points in Fig. 2, which, by (11), are those in set $\mathcal{D}_{fw_1} = \mathcal{S}_{r_1^{fw_1}} \cup \left(\left(\mathcal{S}_{r_4^{fw_1}} \setminus \mathcal{S}_{r_2^{fw_1}} \right) \setminus \mathcal{S}_{r_3^{fw_1}} \right)$. The required set of *deny* rules can be computed as $\mathcal{R}_1^0 \cup \mathcal{R}_4^3$, where, based on (12), $\mathcal{R}_1^0 = \{r_1^{fw_1}\}$ and $\mathcal{R}_4^3 = \bigcup_{r\in\mathcal{R}_4^2}$ `sub_rule(`$r, r_3^{fw_1}$`)`, with $\mathcal{R}_4^2 =$ `sub_rule(`$r_4^{fw_1}, r_2^{fw_1}$`)`.

Alg. 1, implements the described procedure. The algorithm takes as inputs firewalls $fw_i = (r_1^{fw_i}, r_2^{fw_i}, \dots, r_{|fw_i|}^{fw_i})$, $i = 1, 2$, and returns $\overline{fw}_2$, solution to Prob. 1, by computing *deny* rules $r_1, r_2, \dots, r_k$ satisfying (7) and listing them, before rules in fw_2, in $\overline{fw}_2$, as in (8). Alg. 1 sequentially checks rules in fw_1 (line 2), and any time it finds a *deny* rule $r_\ell^{fw_1}$ (line 3), computes rule set $R_\ell^{m_{|M_\ell|}}$ as described in (12). In detail, Alg. 1 initializes a rule set $\mathcal{R}_{min}$ to $\{r_\ell^{fw_1}\}$ (line 4), checks rules preceding $r_\ell^{fw_1}$ (line 5), and any time finds an *allow* rule $r_m^{fw_1}$ (line 6), executes rule subtractions between each rule in $\mathcal{R}_{min}$ and $r_m^{fw_1}$ (lines 8-10). Routine `sub_rule(`r_d, r_a`)` (line 9) implements the defined rule subtraction operator, and is described in detail in Subsect. 4.2. Rules resulting from rule subtractions having $r_m^{fw_1}$ as subtrahend are stored in $\mathcal{R}_{tmp}$ and, subsequently, copied back in $\mathcal{R}_{min}$ to be the new minuend. Once the process is repeated for each *allow* rule $r_m^{fw_1}$ preceding $r_\ell^{fw_1}$, $\mathcal{R}_{min}$ is equal to $\mathcal{R}_\ell^{m_{|M_\ell|}}$ and Alg. 1 lists the rules it contains in $\overline{fw}_2$ (line 14).

4.2. A rule subtraction operator

In this subsection, we describe the defined rule subtraction operator, `sub_rule(`r_d, r_a`)`, implemented by Alg. 3, employed by Alg. 1. Alg. 3 takes as inputs rules $r_d = \langle (C_1^{r_d}, C_2^{r_d}, \dots, C_n^{r_d}), \mathit{action}^{r_d} \rangle$ and $r_a = \langle (C_1^{r_a}, C_2^{r_a}, \dots, C_n^{r_a}), \mathit{action}^{r_a} \rangle$, and returns a set $\mathcal{R}$, $|\mathcal{R}| \in [0, 2n]$, of rules with the same action as r_d, such that $\bigcup_{r\in\mathcal{R}} \mathcal{S}_r = \mathcal{S}_{r_d} \setminus \mathcal{S}_{r_a}$ (proof in App. A.3). Alg. 3 relies on the *range subtraction operator*, `sub_range(`C_d, C_a`)`, which, given two ranges $C_d = [a_d, b_d]$ and $C_a = [a_a, b_a]$, returns the set of ranges $\mathcal{C}$, of minimal cardinality $|\mathcal{C}| \in [0, 2]$, satisfying

Algorithm 1: Security policy migration algorithm

```
Data: $fw_1 = (r_1^{fw_1}, r_2^{fw_1}, \dots, r_{|fw_1|}^{fw_1})$ $fw_2 = (r_1^{fw_2}, r_2^{fw_2}, \dots, r_{|fw_2|}^{fw_2})$
Result: $\overline{fw}_2 = (r_1^{\overline{fw}_2}, r_2^{\overline{fw}_2}, \dots, r_{|\overline{fw}_2|}^{\overline{fw}_2})$
1  $\overline{fw}_2 \leftarrow -$ ;                    /* initialization, firewall with no rules */
2  for ($\ell = 1; \ell \le |fw_1|; \ell++$) do
3      if ($r_\ell^{fw_1}$ is a deny rule) then
4          $\mathcal{R}_{min} \leftarrow \{r_\ell^{fw_1}\}$;
5          for ($m = 1; m < \ell; m++$) do
6              if ($r_m^{fw_1}$ is an allow rule) then
7                  $\mathcal{R}_{tmp} \leftarrow \varnothing$;
8                  forall $r \in \mathcal{R}_{min}$ do
9                      $\mathcal{R}_{tmp} \leftarrow \mathcal{R}_{tmp} \cup$ sub_rule($r, r_m^{fw_1}$);
10                 end
11                 $\mathcal{R}_{min} \leftarrow \mathcal{R}_{tmp}$;
12             end
13         end
14         append all $r \in \mathcal{R}_{min}$ to $\overline{fw}_2$ in any order;
15     end
16 end
17 append rules in $fw_2$ to $\overline{fw}_2$;
```

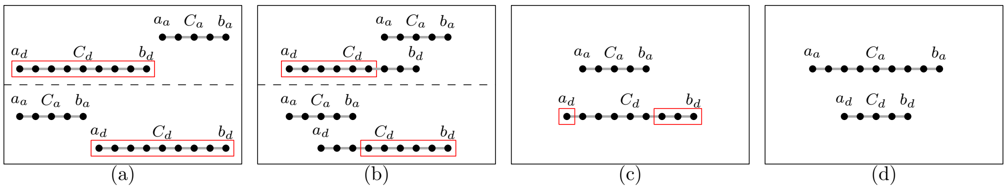
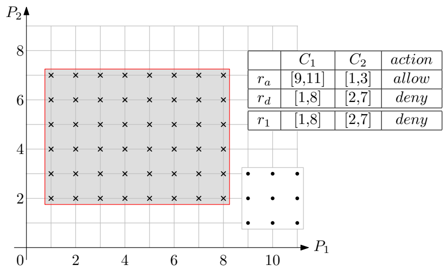
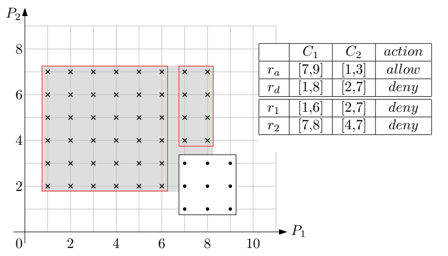


Figure 3: Range subtraction.



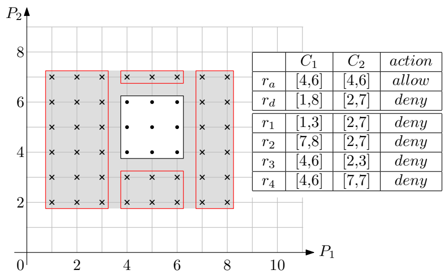
	C_1	C_2	*action*
r_a	[9,11]	[1,3]	*allow*
r_d	[1,8]	[2,7]	*deny*
r_1	[1,8]	[2,7]	*deny*

(a) Resulting rule is $r_1 = r_d$.



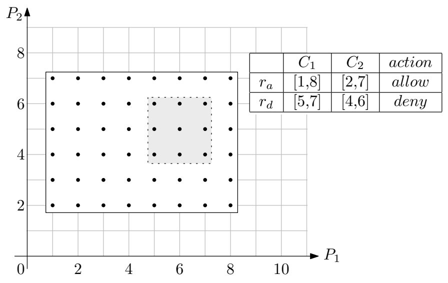
	C_1	C_2	*action*
r_a	[7,9]	[1,3]	*allow*
r_d	[1,8]	[2,7]	*deny*
r_1	[1,6]	[2,7]	*deny*
r_2	[7,8]	[4,7]	*deny*

(b) Two resulting rules.



	C_1	C_2	*action*
r_a	[4,6]	[4,6]	*allow*
r_d	[1,8]	[2,7]	*deny*
r_1	[1,3]	[2,7]	*deny*
r_2	[7,8]	[2,7]	*deny*
r_3	[4,6]	[2,3]	*deny*
r_4	[4,6]	[7,7]	*deny*

(c) Four resulting rules.



	C_1	C_2	*action*
r_a	[1,8]	[2,7]	*allow*
r_d	[5,7]	[4,6]	*deny*

(d) No resulting rules.

Figure 4: Examples of rule subtraction for $n = 2$.

property $\bigcup_{C \in \mathcal{C}} C = C_d \setminus C_a$. The range subtraction operator is implemented by Alg. 2, which distinguishes between the four cases in Fig. 3: if ranges C_d and C_a are non overlapping (cases captured by line 5 or 9, shown in Fig. 3a), $\mathcal{C} = \{C_d\}$, if C_d and C_a partially overlap (cases captured by line 4 or 8, shown in Fig. 3b), $\mathcal{C}$ contains a single range, portion of C_d, if $C_a \subset C_d$ (cases captured by both lines 4 and 8, shown in Fig. 3c), $\mathcal{C}$ contains two ranges, portions of C_d, and, finally, if $C_d \subset C_a$ (none of the above, Fig. 3d), $\mathcal{C} = \varnothing$.

Alg. 3 iteratively computes a set $\mathcal{C}_1$ of conditions defined over n filtering fields (lines 3-8), whose elements are used to compute rules of set $\mathcal{R}$ (line 9). Set $\mathcal{C}_1$ is initialized to sub_range$(C_1^{r_d}, C_1^{r_a})$ and is then updated in $n-1$ steps (lines 5-8). At each step $k \in [2, n]$, $\mathcal{C}_1$ is obtained as the union of two sets of conditions (line 6). The first set is recursively computed as the Cartesian product of current $\mathcal{C}_1$ and set containing single range $C_k^{r_d}$ (the result of the Cartesian product between sets of tuple of ranges is a set of tuple of ranges, i.e., of conditions). The second set is computed as the Cartesian product of set containing single condition $(C_1^{r_d} \cap C_1^{r_a}, \ldots, C_{k-1}^{r_d} \cap C_{k-1}^{r_a})$, with set of ranges sub_range$(C_n^{r_d}, C_n^{r_a})$. Fig. 4

shows examples of pairs of rules defined over $n = 2$ filtering fields. Considering, e.g., case in Fig. 4c, $C_1^{r_d} = [1, 8]$, $C_2^{r_d} = [2, 7]$, $C_1^{r_a} = [4, 6]$, and $C_2^{r_a} = [4, 6]$. Initially, Alg. 3 sets $\mathcal{C}_1 = \texttt{sub\_range}(C_1^{r_d}, C_1^{r_a}) = \{[1, 3], [7, 8]\}$ and $\mathcal{C}_2 = \{C_1^{r_d} \cap C_1^{r_a}\} = \{[4, 6]\}$, at step $k = 2$ (the only step since $n = 2$), $\mathcal{C}_1$ is computed as the union of two sets of conditions. The first is $\mathcal{C}_1 \times C_2^{r_d} = \{[1, 3], [7, 8]\} \times \{[2, 7]\} = \{([1, 3], [2, 7]), ([7, 8], [2, 7])\}$, the second is $\mathcal{C}_2 \times \texttt{sub\_range}(C_2^{r_d}, C_2^{r_a}) = \{[4, 6]\} \times \texttt{sub\_range}([2, 7], [4, 6]) = \{[4, 6]\} \times \{[2, 3], [7, 7])\} = \{([4, 6], [2, 3]), ([4, 6], [7, 7])\}$. Then $\mathcal{C}_1 := \{([1, 3], [2, 7]), ([7, 8], [2, 7]), ([4, 6], [2, 3]), ([4, 6], [7, 7])\}$, from which, since r_d is a *deny* rule, $\mathcal{R} = \{r_1 = \langle([1, 3], [2, 7]), \mathit{deny}\rangle, r_2 = \langle([7, 8], [2, 7]), \mathit{deny}\rangle, r_3 = \langle([4, 6], [2, 3]), \mathit{deny}\rangle, r_4 = \langle([4, 6], [7, 7]), \mathit{deny}\rangle\}$.

In Fig. 4, rules returned by $\texttt{sub\_rule}(r_d, r_a)$ are listed in the bottom part of tables and shown in red in the plots. The number $|\mathcal{R}|$ of resulting rules depends on r_d and r_a. In particular, when r_d and r_a are independent, i.e., $\mathcal{S}_{r_d}$ and $\mathcal{S}_{r_a}$ are disjoint (e.g., Fig. 4a), the result is a single rule equal to r_d, if r_d and r_a are dependent and the corresponding rectangles partially overlap (e.g., Fig. 4b), $|\mathcal{R}| \in [1, 3]$, if $S_{r_a} \subset S_{r_d}$ (e.g., Fig. 4c), $|\mathcal{R}| = 4$, and, finally if r_d is shadowed by r_a, i.e., $S_{r_d} \subset S_{r_a}$ (e.g., Fig. 4d), there are no resulting rules, i.e., $|\mathcal{R}| = 0$. In Figs. 4b and 4c, rules returned by $\texttt{sub\_rule}(r_d, r_a)$ are not contiguous as conditions are defined as tuple of ranges including their extremes and $\texttt{sub\_rule}(r_d, r_a)$ returns independent rules. Other rule subtraction operators can be defined, however $\texttt{sub\_rule}(r_d, r_a)$ guarantees that the number of resulting rules, $|\mathcal{R}|$, is kept to the minimum.

Fig 5 shows result obtained when security policy migration, implemented by Alg. 1, is applied to example in Fig. 1c. As can be seen, $\overline{fw}_1$ is the trivial firewall, while $\overline{fw}_2$ has been obtained by placing 12 *deny* rules before those originally in fw_2. Actually, *deny* rules at the top of $\overline{fw}_2$ are the result of a further optimization, as often subsets of rules returned by Alg. 3 can be unified in single rules. As a first observation, expressing the security policy implemented by fw_1 with only *deny* rules, requires a higher number of rules, which penalizes the cooperating firewall. One way to mitigate this effect is to use compression algorithms on $\overline{fw}_2$. As a second observation, the proposed technique can be, in principle, used to move only part of the security policy implemented by fw_1, as by migrating any subset of *deny* rules $r_1, r_2, \ldots, r_k$ to fw_2, (7) is still satisfied. However, from the performance point of view, a better solution would be to split

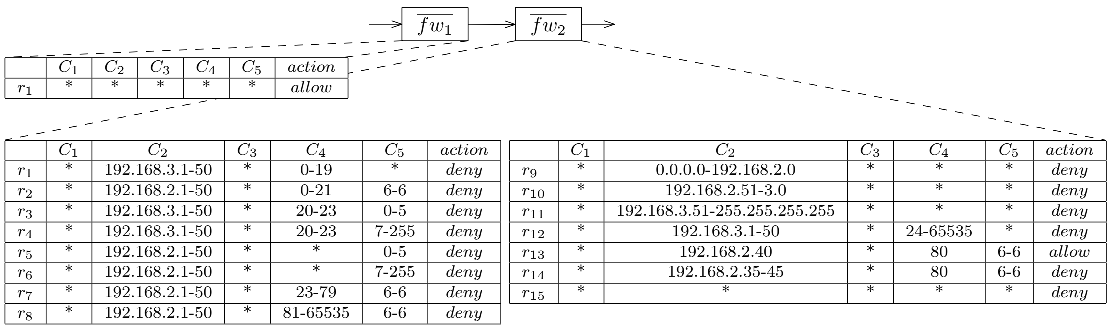
Algorithm 2: Algorithm for range subtraction.

```
Data: $C_d = [a_d, b_d], C_a = [a_a, b_a], C_d, C_a \subseteq \mathbb{N}_0$.
Result: A set of ranges $\mathcal{C}$, $|\mathcal{C}| \in [0, 2]$.
Function sub_range($C_d$,$C_a$):
    $\mathcal{C} \leftarrow \varnothing$;                                   /* initialization */
    if ($a_a > a_d$) then
        if ($a_a \leq b_d$) then $\mathcal{C} \leftarrow \mathcal{C} \cup \{[a_d, a_a - 1]\}$ ;
        else $\mathcal{C} \leftarrow \mathcal{C} \cup \{[a_d, b_d]\}$ ;
    end
    if ($b_a < b_d$) then
        if ($b_a \geq a_d$) then $\mathcal{C} \leftarrow \mathcal{C} \cup \{[b_a + 1, b_d]\}$ ;
        else $\mathcal{C} \leftarrow \mathcal{C} \cup \{[a_d, b_d]\}$ ;
    end
    return $\mathcal{C}$
```

Algorithm 3: Algorithm for rule subtraction in $\mathbb{N}_0^n$.

```
Data: $r_d = \langle (C_1^{r_d}, C_2^{r_d}, \ldots, C_n^{r_d}), action^{r_d} \rangle$, $r_a = \langle (C_1^{r_a}, C_2^{r_a}, \ldots, C_n^{r_a}), action^{r_a} \rangle$
Result: A set $\mathcal{R}$ of rules with the same action as $r_d$, $|\mathcal{R}| \in [0, 2n]$.
1  Function sub_rule($r_d$,$r_a$):
2      $\mathcal{R} \leftarrow \varnothing$;                                  /* initialization */
3      $\mathcal{C}_1 \leftarrow$ sub_range($C_1^{r_d}, C_1^{r_a}$);
4      $\mathcal{C}_2 \leftarrow \{C_1^{r_d} \cap C_1^{r_a}\}$; /* the intersection of two ranges is always a single range */
5      for ($k = 2$; $k \leq n$; $k++$) do
6          $\mathcal{C}_1 \leftarrow (\mathcal{C}_1 \times \{C_k^{r_d}\}) \cup (\mathcal{C}_2 \times$ sub_range($C_k^{r_d}, C_k^{r_a}$));
7          $\mathcal{C}_2 \leftarrow \mathcal{C}_2 \times \{C_k^{r_d} \cap C_k^{r_a}\}$;
8      end
9      forall $c \in \mathcal{C}_1$ do $\mathcal{R} \leftarrow \mathcal{R} \cup \{r = \langle c, action^{r_d} \rangle\}$ ;
10     return $\mathcal{R}$
```

the security policy of fw_1 before translating it into *deny* rules, so as to minimize the number of rules added to fw_2. This can be done by suitably partitioning $\mathcal{P}$ and, accordingly, rules in fw_1 [16], so that a new firewall $\widehat{fw_1}$ is given as a input to Alg. 3. The same approach, as detailed in [16], allows to extend the technique to the case of more complex topologies, where firewalls are arbitrarily placed to guard specific domains. Note that, in this case, additional constraints deriving from network topology should be considered in security policy migration.



	C_1	C_2	C_3	C_4	C_5	*action*
r_1	*	*	*	*	*	*allow*

	C_1	C_2	C_3	C_4	C_5	*action*
r_1	*	192.168.3.1-50	*	0-19	*	*deny*
r_2	*	192.168.2.1-50	*	0-21	6-6	*deny*
r_3	*	192.168.3.1-50	*	20-23	0-5	*deny*
r_4	*	192.168.3.1-50	*	20-23	7-255	*deny*
r_5	*	192.168.2.1-50	*	*	0-5	*deny*
r_6	*	192.168.2.1-50	*	*	7-255	*deny*
r_7	*	192.168.2.1-50	*	23-79	6-6	*deny*
r_8	*	192.168.2.1-50	*	81-65535	6-6	*deny*

	C_1	C_2	C_3	C_4	C_5	*action*
r_9	*	0.0.0.0-192.168.2.0	*	*	*	*deny*
r_{10}	*	192.168.2.51-3.0	*	*	*	*deny*
r_{11}	*	192.168.3.51-255.255.255.255	*	*	*	*deny*
r_{12}	*	192.168.3.1-50	*	24-65535	*	*deny*
r_{13}	*	192.168.2.40	*	80	6-6	*allow*
r_{14}	*	192.168.2.35-45	*	80	6-6	*deny*
r_{15}	*	*	*	*	*	*deny*

Figure 5: Sequence $\overline{fws}$ obtained from example in Fig. 1c after security policy migration.

5. Conclusions

We presented a technique for migrating security policies along firewalls in a sequence, which is formally verified to preserve the overall security policy implemented by the sequence itself. The proposed technique can be extended to the case of more general topologies comprising firewall-protected domains and is the building block for the development of cooperative solutions balancing workload by distributing filtering responsibility among firewalls available within a protected network. Future work will analyze solutions to compress rule lists obtained after migration in the cooperating firewall, as the proposed technique may lead to rule proliferation. Techniques for suitably partitioning the packet space will also be investigated to split policies based on network/workload characteristics and further optimize network performance.

6. Acknowledgments

This work was supported in part by the EU H2020-SU-ICT-03-2018 Project No. 830929 CyberSec4Europe (cybersec4europe.eu).

References

[1] M. Cheminod, L. Durante, L. Seno, A. Valenzano, Performance evaluation and modeling of an industrial application-layer firewall, IEEE Trans. Ind. Informat. 14 (2018) 2159–2170.

[2] K. Salah, K. Elbadawi, R. Boutaba, Performance modeling and analysis of network firewalls, IEEE Trans. Netw. Service Manag. 9 (2012) 12–21.

[3] K. Salah, K. Sattar, Z. Baig, M. Sqalli, P. Calyam, Resiliency of open-source firewalls against remote discovery of last-matching rules, in: Proc. Int. Conf. on Security of Information and Networks (SIN), 2009, p. 186–192.

[4] C. Bodei, L. Ceragioli, P. Degano, R. Focardi, L. Galletta, F. Luccio, M. Tempesta, L. Veronese, FWS: Analyzing, maintaining and transcompiling firewalls, J. Comp. Sec. 29 (2021) 77–134.

[5] A. A. Jabal, M. Davari, E. Bertino, C. Makaya, S. Calo, D. Verma, A. Russo, C. Williams, Methods and tools for policy analysis, ACM Comput. Surv. 51 (2019).

[6] F. Valenza, S. Spinoso, C. Basile, R. Sisto, A. Lioy, A formal model of network policy analysis, in: Proc. IEEE Int. Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), 2015, pp. 516–522.

[7] E. Al-Shaer, H. Hamed, R. Boutaba, M. Z. Hasan, Conflict Classification and Analysis of Distributed Firewall Policies, IEEE J. Sel. Areas Commun. 23 (2005) 2069–2084.

[8] A. X. Liu, E. Torng, C. R. Meiners, Firewall Compressor: An Algorithm for Minimizing Firewall Policies, in: Proc. IEEE Conf. on Comp. Comm. (INFOCOM), 2008, pp. 176–180.

[9] Y. Cheng, W. Wang, J. Wang, H. Wang, FPC: A new approach to firewall policies compression, Tsinghua Science and Technology 24 (2019) 65–76.

[10] E. W. Fulp, Optimization of network firewall policies using ordered sets and directed acyclical graphs, in: Proc. of IEEE Internet Management Conference, 2005, pp. 1–4.

[11] H. Hamed, E. Al-Shaer, Dynamic rule-ordering optimization for high-speed firewall filtering, in: Proc. ACM Symp. on Inf. Comp. and Comm. Sec. (ASIACCS), 2006, p. 332–342.

[12] R. Mohan, A. Yazidi, B. Feng, J. Oommen, On optimizing firewall performance in dynamic networks by invoking a novel swapping window–based paradigm, Int. J. Commun. Syst. 31 (2018) e3773.

[13] T. Harada, K. Tanaka, K. Mikawa, A heuristic algorithm for relaxed optimal rule ordering problem, in: Proc. of Cyber Security in Networking Conference (CSNet), 2018, pp. 1–8.

[14] G. Yan, S. Chen, S. Eidenbenz, Dynamic balancing of packet filtering workloads on distributed firewalls, in: Proc. of IEEE Int. Workshop on QoS (IWQoS), 2008, pp. 209–218.

[15] S. Bagheri, A. Shameli-Sendi, Dynamic firewall decomposition and composition in the cloud, IEEE Trans. Inf. Forensics Security 15 (2020) 3526–3539.

[16] L. Durante, L. Seno, A. Valenzano, A formal model and technique to redistribute the packet filtering load in multiple firewall networks, IEEE Trans. Inf. Forensics Security 16 (2021) 2637–2651.

A. Appendix

A.1. Proof of Thm. 1

Given $fws = (fw_1, fw_2)$, $fw_i = (r_1^{fw_i}, r_2^{fw_i}, \ldots, r_{|fw_i|}^{fw_i})$, $i = 1, 2$, and $\overline{fws} = (\overline{fw}_1, \overline{fw}_2)$, where $\overline{fw}_1 = (\langle(*, *, \ldots, *), allow\rangle)$ and $\overline{fw}_2 = (r_1, \ldots, r_k, r_1^{fw_2}, \ldots, r_{|fw_2|}^{fw_2})$, if $r_1, \ldots, r_k$ are $deny$ rules such that $\bigcup_{j=1}^{k} \mathcal{S}_{r_j} = \mathcal{D}_{fw_1} = \{p \in \mathcal{P} \mid f_{fw_1}(p) = -\}$, then $fws \equiv \overline{fws}$.

Proof. By firewall completeness, a partition of $\mathcal{P}$ is $\{\mathcal{D}_{fw_1}, \mathcal{A}_{fw_1}\}$, where $\mathcal{D}_{fw_1} = \{p \in \mathcal{P} \mid f_{fw_1}(p) = -\}$ and $\mathcal{A}_{fw_1} = \{p \in \mathcal{P} \mid f_{fw_1}(p) = p\}$ are the sets of packets, respectively, discarded and forwarded by fw_1. By hypothesis, packets in $\mathcal{D}_{fw_1}$ match at least one rule in $r_1, \ldots, r_k$. Conversely, since $\mathcal{A}_{fw_1}$ and $\mathcal{D}_{fw_1}$ are disjoint sets, packets in $\mathcal{A}_{fw_1}$ do not match any of rules $r_1, \ldots, r_k$. Packets in $\mathcal{D}_{fw_1}$ are discarded by fws as, by definition of $\mathcal{D}_{fw_1}$, they are discarded by fw_1. Equivalently, packets in $\mathcal{D}_{fw_1}$ are discarded by $\overline{fws}$, as they are forwarded by the trivial firewall $\overline{fw}_1$, but discarded by fw_2, as they match at least one of the $deny$ rules $r_1, \ldots, r_k$ placed at the top of fw_2.

Packets in $\mathcal{A}_{fw_1}$ are forwarded by fw_1 by definition of $\mathcal{A}_{fw_1}$, thus their fate in fws is determined by fw_2. In $\overline{fws}$, packets in $\mathcal{A}_{fw_1}$ are forwarded by the trivial firewall $\overline{fw}_1$ and their fate is determined by $\overline{fw}_2$. However, since packets in $\mathcal{A}_{fw_1}$ do not match any of rules $r_1, \ldots, r_k$, they have as a matching rule in $\overline{fw}_2$ the same ($allow$ or $deny$) matching rule they have in fw_2, i.e., $f_{fw_2}(p) = f_{\overline{fw}_2}(p)$. Summarizing, $\forall p \in \mathcal{D}_{fw_1}, f_{fws}(p) = f_{\overline{fws}}(p) = -$ and $\forall p \in \mathcal{A}_{fw_1}, f_{fws}(p) = f_{\overline{fws}}(p) = f_{fw_2}(p)$. Since $\{\mathcal{D}_{fw_1}, \mathcal{A}_{fw_1}\}$ is a partition of $\mathcal{P}$, it follows that $\forall p \in \mathcal{P}$, $f_{\overline{fws}}(p) = f_{fw}(p)$, i.e., $\overline{fws} \equiv fws$, which proves the thesis. □

A.2. Proofs referenced in Subsect. 4.1

Given a firewall $fw_1 = (r_1^{fw_1}, r_2^{fw_1}, \ldots, r_{|fw_1|}^{fw_1})$, the set of packets discarded by the firewall, $\mathcal{D}_{fw_1} = \{p \in \mathcal{P} \mid f_{fw_1}(p) = -\}$, can be expressed as:

$$\mathcal{D}_{fw_1} = \bigcup_{\ell \in L} \left(\mathcal{S}_{r_\ell^{fw_1}} \setminus \left(\bigcup_{m \in M, m < \ell} \mathcal{S}_{r_m^{fw_1}} \right) \right) \tag{13}$$

where L, and M are, respectively, is the set of indexes of $deny$ and $allow$ rules in fw_1.

Proof. By definition of firewall operation, $\mathcal{D}_{fw_1}$ is the set of packets having as a matching rule in $\overline{fw}_1$ a $deny$ rule, i.e.,

$$\begin{aligned}\mathcal{D}_1 &= \{p \in \mathcal{P} \mid \exists\, \ell \in L,\ p \in \mathcal{S}_{r_\ell^{fw_1}},\ \forall m \in I, m < \ell,\ p \notin \mathcal{S}_{r_m^{fw_1}}\} \\ &= \bigcup_{\ell \in L} \left(\mathcal{S}_{r_\ell^{fw_1}} \setminus \left(\bigcup_{m \in I, m < \ell} \mathcal{S}_{r_m^{fw_1}} \right) \right) \end{aligned} \tag{14}$$

where $I = L \cup M$ is the set of indexes of all rules in fw_1. In (14), each set $\mathcal{S}_{r_{\overline{m}}^{fw_1}}$, defined by a $deny$ rule in fw_1 (i.e., $\overline{m} \in L$), appearing at least once in the second (internal) set union, also appears once in the first (external) set union. That is, for any term $\mathcal{C} \setminus \left(\bigcup_{m<\overline{m}} \left(\mathcal{S}_{r_m^{fw_1}}\right) \cup \mathcal{S}_{r_{\overline{m}}^{fw_1}} \cup \mathcal{D}\right)$, where $\overline{m} \in L$, and $\mathcal{C}$ and $\mathcal{D}$ have been used to indicate unions of sets defined by rules in fw_1, there is also a term $\mathcal{S}_{r_{\overline{m}}^{fw_1}} \setminus \left(\bigcup_{m<\overline{m}} \mathcal{S}_{r_m^{fw_1}}\right)$. By set property $(\mathcal{A} \setminus \mathcal{B}) \cup (\mathcal{C} \setminus (\mathcal{A} \cup \mathcal{B} \cup \mathcal{D})) = (\mathcal{A} \setminus \mathcal{B}) \cup (C \setminus (\mathcal{B} \cup \mathcal{D}))$, which holds for generic sets $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}$, considering $\mathcal{S}_{r_{\overline{m}}^{fw_1}} = \mathcal{A}$ and $\bigcup_{m<\overline{m}} \left(\mathcal{S}_{r_m^{fw_1}}\right) = \mathcal{B}$, it follows that term $\mathcal{S}_{r_{\overline{m}}^{fw_1}}$ can be neglected any time it appears in the second (internal) set union in (14), i.e.,

$$\mathcal{D}_{fw_1} = \bigcup_{\ell \in L} \left(\mathcal{S}_{r_\ell^{fw_1}} \setminus \left(\bigcup_{m \in I \setminus L, m<\ell} \mathcal{S}_{r_m^{fw_1}} \right) \right) \tag{15}$$

since $I \setminus L = M$, equality (15) proves the thesis. □

We consider a packet set $\mathcal{D}_{fw_1}$ defined by

$$\mathcal{D}_{fw_1} = \bigcup_{\ell \in L} \left(\left(\dots \left(\left(\mathcal{S}_{r_\ell^{fw_1}} \setminus \mathcal{S}_{r_{m_1}^{fw_1}} \right) \setminus \mathcal{S}_{r_{m_2}^{fw_1}} \right) \dots \right) \setminus \mathcal{S}_{r_{m_{|M_\ell|}}^{fw_1}} \right) \tag{16}$$

where, in general $\mathcal{S}_r$ is the packet set defined by a rule r and where L and M are sets of indexes and, for each $\ell \in L$, $M_\ell = \{m_1, m_2, \dots, m_{|M_\ell|}\} = \{m \in M, m < \ell\}$ is the set of indexes in M preceding ℓ. We then consider, rule set $\mathcal{R}^* = \bigcup_{\ell \in L} \mathcal{R}_\ell^{m_{|M_\ell|}}$, where for each $\ell \in L$ rule set $\mathcal{R}_\ell^{m_{|M_\ell|}}$ is iteratively computed as:

$$\mathcal{R}_\ell^{m_0} = \{r_\ell^{fw_1}\}, \qquad \mathcal{R}_\ell^{m_t} = \bigcup_{r \in \mathcal{R}_\ell^{m_{t-1}}} \texttt{sub\_rule}(r, r_{m_t}^{fw_1}) \quad t = 1, \dots, m_{|M_\ell|} \tag{17}$$

where $\texttt{sub\_rule}(r_d, r_a)$ is a rule subtraction operator, want to prove that $\bigcup_{r \in \mathcal{R}^*} \mathcal{S}_r = \mathcal{D}_{fw_1}$

Proof. We prove that for each $\ell \in L$, rule set $\mathcal{R}_\ell^{m_{|M_\ell|}}$, computed as defined in (17), is such that:

$$\bigcup_{r \in \mathcal{R}_\ell^{m_{|M_\ell|}}} \mathcal{S}_r = \left(\dots \left(\left(\mathcal{S}_{r_\ell^{fw_1}} \setminus \mathcal{S}_{r_{m_1}^{fw_1}} \right) \setminus \mathcal{S}_{r_{m_2}^{fw_1}} \right) \dots \right) \setminus \mathcal{S}_{r_{m_{|M_\ell|}}^{fw_1}} \tag{18}$$

We prove (18), by proving by induction over t, that for each $t = 1, \dots, |M_\ell|$, rule set $\mathcal{R}_\ell^{m_t}$, computed as defined in (17), is such that:

$$\bigcup_{r \in \mathcal{R}_\ell^{m_t}} \mathcal{S}_r = \left(\dots \left(\left(\mathcal{S}_{r_\ell^{fw_1}} \setminus \mathcal{S}_{r_{m_1}^{fw_1}} \right) \setminus \mathcal{S}_{r_{m_2}^{fw_1}} \right) \dots \right) \setminus \mathcal{S}_{r_{m_t}^{fw_1}} \tag{19}$$

Base case ($t = 1$): according to (17), $\mathcal{R}_\ell^{m_1} = \texttt{sub\_rule}(r_\ell^{fw_1}, r_{m_1}^{fw_1})$. By definition of rule subtraction operator, it holds that $\bigcup_{r \in \mathcal{R}_\ell^{m_1}} \mathcal{S}_r = \mathcal{S}_{r_\ell^{fw_1}} \setminus \mathcal{S}_{r_{m_1}^{fw_1}}$, which proves the thesis.

Induction ($t - 1 \Rightarrow t$): $\mathcal{R}_\ell^{m_{t-1}}$ is such that

$$\bigcup_{r \in \mathcal{R}_\ell^{m_{t-1}}} \mathcal{S}_r = \left(\dots \left(\left(\mathcal{S}_{r_\ell^{fw_1}} \setminus \mathcal{S}_{r_{m_1}^{fw_1}} \right) \setminus \mathcal{S}_{r_{m_2}^{fw_1}} \right) \dots \right) \setminus \mathcal{S}_{r_{m_{t-1}}^{fw_1}} \tag{20}$$

From (20), set $\left(\left(\dots\left(\left(\mathcal{S}_{r_\ell^{fw_1}} \setminus \mathcal{S}_{r_{m_1}^{fw_1}}\right) \setminus \mathcal{S}_{r_{m_2}^{fw_1}}\right) \dots\right) \setminus \mathcal{S}_{r_{m_{t-1}}^{fw_1}}\right) \setminus \mathcal{S}_{r_{m_t}^{fw_1}}$ can be expressed as:

$$\left(\bigcup_{r\in\mathcal{R}_\ell^{m_{t-1}}} \mathcal{S}_r\right) \setminus \mathcal{S}_{r_{m_t}^{fw_1}} = \bigcup_{r\in\mathcal{R}_\ell^{m_{t-1}}} \left(\mathcal{S}_r \setminus \mathcal{S}_{r_{m_t}^{fw_1}}\right) \tag{21}$$

where the second equality in (21) derives from set property, valid for generic sets $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{A}_j$, $\mathcal{A} = \bigcup_{j=1}^k \mathcal{A}_j \quad\Rightarrow\quad \mathcal{A} \setminus B = \bigcup_{j=1}^k (\mathcal{A}_j \setminus \mathcal{B})$. Since, according to (17), $\mathcal{R}_\ell^{m_t} = \bigcup_{r\in\mathcal{R}_\ell^{m_{t-1}}} \texttt{sub\_rule}(r, r_{m_t}^{fw_1})$, by definition of rule subtraction operator, thesis holds true. $\square$

A.3. Proof referenced in Subsect. 4.2

Alg. 3 implements a rule subtraction operator, i.e., given two rules $r_d = \langle(C_1^{r_d}, C_2^{r_d}, \dots, , C_n^{r_d}), action^{r_d}\rangle$ and $r_a = \langle(C_1^{r_a}, C_2^{r_a}, \dots, , C_n^{r_a}), action^{r_a}\rangle$, it returns a set of rules $\mathcal{R}$, with the same action as r_d, satisfying property: $\bigcup_{r\in\mathcal{R}} \mathcal{S}_r = \mathcal{S}_{r_d} \setminus \mathcal{S}_{r_a}$.

Proof. Firstly, we prove that $\mathcal{S}_{r_d} \setminus \mathcal{S}_{r_a}$ can be expressed in terms of a union of hyperrectangles, that can be translated in a set $\mathcal{R}$ of rules. Then we show that Alg. 3 computes $\mathcal{R}$ this way.

Given a rule $r = \langle(C_1, C_2, \dots, C_n), action\rangle$, the set of packets matching the rule is $\mathcal{S}_r = C_1 \times C_2 \times \dots \times C_n \subseteq \mathcal{P}$. Note that, any packet set that can be expressed as the Cartesian product of n ranges can be defined by a rule having as a condition the n-tuple of ranges. We call $\overset{k}{\mathcal{S}}_r$, $k \le n$, the projection of $\mathcal{S}_r$ over set $\overset{k}{\mathcal{P}} \subseteq \mathbb{N}_0^k$, where $\overset{k}{\mathcal{P}}$ is the projection of $\mathcal{P}$ over $\mathbb{N}_0^k$:

$$\overset{k}{\mathcal{S}}_r = \{p \in \overset{k}{\mathcal{P}} \mid p_i \in C_i, i = 1, \dots, k\} = C_1 \times C_2 \times \dots \times C_k \tag{22}$$

Set $\overset{k}{\mathcal{S}}_r$, $k \le n$, is a hyperrectangle in $\mathbb{N}_0^k$. Given a rule r_d, from (22) it holds that $\overset{k}{\mathcal{S}}_{r_d} = \overset{k-1}{\mathcal{S}}_{r_d} \times C_k^{r_d}$. Since by set property $\mathcal{A} = (\mathcal{A} \setminus \mathcal{B}) \cup (\mathcal{A} \cap \mathcal{B})$, which holds for any pair of sets $\mathcal{A}$ and $\mathcal{B}$, we have that $\overset{k-1}{\mathcal{S}}_{r_d} = (\overset{k-1}{\mathcal{S}}_{r_d} \setminus \overset{k-1}{\mathcal{S}}_{r_a}) \cup (\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a})$, we obtain:

$$\overset{k}{\mathcal{S}}_{r_d} = ((\overset{k-1}{\mathcal{S}}_{r_d} \setminus \overset{k-1}{\mathcal{S}}_{r_a}) \cup (\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a})) \times C_k^{r_d} = ((\overset{k-1}{\mathcal{S}}_{r_d} \setminus \overset{k-1}{\mathcal{S}}_{r_a}) \times C_k^{r_d}) \cup ((\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a}) \times C_k^{r_d}) \tag{23}$$

By set property $\mathcal{A} \setminus \mathcal{B} = \mathcal{A} \setminus (\mathcal{A} \cap \mathcal{B})$, we have $\overset{k}{\mathcal{S}}_{r_d} \setminus \overset{k}{\mathcal{S}}_{r_a} = \overset{k}{\mathcal{S}}_{r_d} \setminus (\overset{k}{\mathcal{S}}_{r_d} \cap \overset{k}{\mathcal{S}}_{r_a})$ and by using (23) we obtain

$$\begin{aligned}\overset{k}{\mathcal{S}}_{r_d} \setminus \overset{k}{\mathcal{S}}_{r_a} &= (((\overset{k-1}{\mathcal{S}}_{r_d} \setminus \overset{k-1}{\mathcal{S}}_{r_a}) \times C_k^{r_d}) \cup ((\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a}) \times C_k^{r_d})) \setminus (\overset{k}{\mathcal{S}}_{r_d} \cap \overset{k}{\mathcal{S}}_{r_a}) \\ &= (((\overset{k-1}{\mathcal{S}}_{r_d} \setminus \overset{k-1}{\mathcal{S}}_{r_a}) \times C_k^{r_d}) \setminus (\overset{k}{\mathcal{S}}_{r_d} \cap \overset{k}{\mathcal{S}}_{r_a})) \cup (((\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a}) \times C_k^{r_d}) \setminus (\overset{k}{\mathcal{S}}_{r_d} \cap \overset{k}{\mathcal{S}}_{r_a}))\end{aligned} \tag{24}$$

We can rewrite $((\overset{k-1}{\mathcal{S}}_{r_d} \setminus \overset{k-1}{\mathcal{S}}_{r_a}) \times C_k^{r_d})$ and $(\overset{k}{\mathcal{S}}_{r_d} \cap \overset{k}{\mathcal{S}}_{r_a})$ as $((\overset{k-1}{\mathcal{S}}_{r_d} \setminus (\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a})) \times C_k^{r_d})$ and $(\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a}) \times (C_k^{r_d} \cap C_k^{r_a})$, respectively. Since $(\overset{k-1}{\mathcal{S}}_{r_d} \setminus (\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a}))$ and $(\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a})$ are disjoint,

$((\overset{k-1}{\mathcal{S}}_{r_d} \setminus \overset{k-1}{\mathcal{S}}_{r_a}) \times C_k^{r_d})$ and $(\overset{k}{\mathcal{S}}_{r_d} \cap \overset{k}{\mathcal{S}}_{r_a})$ are disjoint and (24) becomes:

$$\overset{k}{\mathcal{S}}_{r_d} \setminus \overset{k}{\mathcal{S}}_{r_a} = ((\overset{k-1}{\mathcal{S}}_{r_d} \setminus \overset{k-1}{\mathcal{S}}_{r_a}) \times C_k^{r_d}) \cup (((\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a}) \times C_k^{r_d}) \setminus (\overset{k}{\mathcal{S}}_{r_d} \cap \overset{k}{\mathcal{S}}_{r_a})) \tag{25}$$

Since sets $\overset{k}{\mathcal{S}}_r$, $k \leq n$ are hyperrectangles, it holds that

$$\overset{k}{\mathcal{S}}_{r_d} \cap \overset{k}{\mathcal{S}}_{r_a} = (C_1^{r_d} \cap C_1^{r_a}) \times (C_2^{r_d} \cap C_2^{r_a}) \times \ldots \times (C_k^{r_d} \cap C_k^{r_a}) = (\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a}) \times (C_k^{r_d} \cap C_k^{r_a}) \tag{26}$$

From (26), we can rewrite the second term in union (25) as:

$$\begin{aligned}
((\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a}) \times C_k^{r_d}) \setminus (\overset{k}{\mathcal{S}}_{r_d} \cap \overset{k}{\mathcal{S}}_{r_a}) &= ((\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a}) \times C_k^{r_d}) \setminus ((\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a}) \times (C_k^{r_d} \cap C_k^{r_a})) \\
&= (\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a}) \times (C_k^{r_d} \setminus (C_k^{r_d} \cap C_k^{r_a})) \\
&= (\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a}) \times (C_k^{r_d} \setminus C_k^{r_a}) \qquad (27)
\end{aligned}$$

where the last equality is obtained by using again set property $\mathcal{A} \setminus \mathcal{B} = \mathcal{A} \setminus (\mathcal{A} \cap \mathcal{B})$.

Summarizing, from (25) and (27) we have that:

$$\overset{k}{\mathcal{S}}_{r_d} \setminus \overset{k}{\mathcal{S}}_{r_a} = ((\overset{k-1}{\mathcal{S}}_{r_d} \setminus \overset{k-1}{\mathcal{S}}_{r_a}) \times C_k^{r_d}) \cup ((\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a}) \times (C_k^{r_d} \setminus C_k^{r_a})) \tag{28}$$

To translate set (28) into a set of rules defining it, we prove, by induction over k, that (28) can be expressed as a set of sets of the kind $C_1 \times C_2 \times \ldots\ C_n$ (i.e., of hyperrectangles).

Base case ($k = 1$): $\overset{1}{\mathcal{S}}_{r_d} \setminus \overset{1}{\mathcal{S}}_{r_a} = (C_1^{r_d} \setminus C_1^{r_a})$. $(C_1^{r_d} \setminus C_1^{r_a})$ can be computed by $\texttt{sub\_range}(C_1^{r_d}, C_1^{r_a})$, which returns a set of ranges (i.e., of hyperrectangles in $\mathbb{N}_0^1$).

Induction step ($k - 1 \Rightarrow k$): We assume $\overset{k-1}{\mathcal{S}}_{r_d} \setminus \overset{k-1}{\mathcal{S}}_{r_a}$ to be expressed as a set of hyperrectangles. The first term in union (28), is then a hyperrectangle. The second term in union in union (28) is $((\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a}) \times (C_k^{r_d} \setminus C_k^{r_a}))$. By (26), $(\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a})$ is a hyperrectangle, and since $(C_k^{r_d} \setminus C_k^{r_a})$ can be computed by $\texttt{sub\_range}(C_k^{r_d}, C_k^{r_a})$ which returns a set of ranges, $((\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a}) \times (C_k^{r_d} \setminus C_k^{r_a}))$ returns a set of hyperrectangles, which proves the thesis.

Now we prove that Alg. 3 computes conditions $(C_1, C_2, \ldots, C_n)$ defining the just described hyperrectangles (28), from which it generates rules of $\mathcal{R}$. In (28) set $\overset{k}{\mathcal{S}}_{r_d} \setminus \overset{k}{\mathcal{S}}_{r_a}$ is incrementally built by set $\overset{k-1}{\mathcal{S}}_{r_d} \setminus \overset{k-1}{\mathcal{S}}_{r_a}$, $k \leq n$, we call the set of conditions defining the first set $\overset{k}{\mathcal{C}}_1$. We compute $C_k^{r_d} \setminus C_k^{r_a}$ as $\texttt{sub\_range}(C_k^{r_d}, C_k^{r_a})$. Moreover, we call the set of conditions defining $\overset{k-1}{\mathcal{S}}_{r_d} \cap \overset{k-1}{\mathcal{S}}_{r_a}$ as $\overset{k-1}{\mathcal{C}_2}$. We can now rewrite (28) in terms of set of conditions as:

$$\overset{k}{\mathcal{C}}_1 = (\overset{k-1}{\mathcal{C}_1} \times C_k^{r_d}) \cup (\overset{k-1}{\mathcal{C}_2} \times \texttt{sub\_range}(C_k^{r_d}, C_k^{r_a})) \tag{29}$$

In Alg. 3, two set $\mathcal{C}_1$ and $\mathcal{C}_2$ are initialized (lines 3-4) respectively to $\texttt{sub\_range}(C_1^{r_d}, C_1^{r_a})$ and $\overset{1}{\mathcal{S}}_{r_d} \cap \overset{1}{\mathcal{S}}_{r_a}$. Both are then updated in n step (loop in lines 5-8), in particular line 6 implements (29). At the end, $\mathcal{C}_1 = \overset{n}{\mathcal{C}}_1$, and the set of conditions is used to build rule set $\mathcal{R}$, which proves the thesis. $\square$