Atomizing Firewall Policies for Anomaly Analysis and Resolution

Atomizing firewall policies for anomaly analysis and resolution

Daniele Bringhenti, Simone Bussa, Riccardo Sisto, Fulvio Valenza

***Abstract*—Nowadays, the security management of packet filtering firewall policies got complicated due to the evolution of modern computer networks, characterized by growing size and heterogeneity of communications. The traditional manual approaches for configuring firewalls have become error-prone, unoptimized and time-consuming, leading to an increasing number of policy anomalies, including both sub-optimizations and conflicts. In literature, the techniques proposed for anomaly management have several shortcomings, as their anomaly analysis is usually excessively complex, while their anomaly resolution cannot solve all anomalies. In order to overcome these shortcomings, this paper proposes a comprehensive approach for firewall policy anomaly analysis and resolution, based on the formal concept of atomic predicates. This approach has the aim to simplify the anomaly management operations, make them efficient and solve all configuration anomalies. The achievement of these objectives has been experimentally proved through the validation of a framework which implements the proposed approach, and whose time performance and anomaly management efficiency have been compared with the relevant alternative approaches.**

***Index Terms*—firewall, policy anomaly management, policy-based systems**

I. Introduction

Packet filtering firewalls represent the most commonly used security function type to protect computer networks from undesired or malicious traffic. They may be deployed at the network edge as a first defense line to stop external cyber attacks, but also at the network core so as to block unauthorized traffic [1]. The behavior of a firewall is determined by its policy, characterized by a rule list establishing which packet classes must be discarded and which ones can be forwarded toward their destination.

The recent evolution of next-generation computer networks poses new challenges for the management of firewall policies. Modern networks, such as the ones based on virtualization paradigms, have introduced higher dynamism and agility, with which all network management operations must comply. Besides, they are characterized by increasing size and heterogeneity of communications. Networks based on the Internet-of-Things technology exemplify how everyday unsafe devices can connect to each other, opening a multitude of communications that administrators should take care of. Due to this higher complexity, the traditional manual approaches used by network administrators for configuring firewall policies have become even more error-prone, unoptimized, and time-consuming [2]. Those approaches could work with small-sized networks, where everything was almost static and under the direct control of a human user, while they now lead to an increasing number of misconfigurations. Moreover, after a cyber attack is detected, an administrator introduces new rules or removes existing ones from a firewall policy as fast as possible, and the urgency of this operation may introduce further problems in the configuration [3].

All these misconfigurations due to manual approaches can cause anomalies among firewall policy rules, which can be categorized as sub-optimizations and conflicts. Sub-optimizations occur when different policy rules enforce the same filtering action on the same packet class, and when removing a sub-optimal rule does not change the firewall behavior. Their presence may decrease the filtering performance and increase the complexity of firewall management. Instead, conflicts are characterized by the application of opposite actions of different rules on the same packets, thus causing an undesired firewall behavior. They may lead to discarding benign traffic, or creating new vulnerabilities to be exploited by attackers.

Identifying and removing all the anomalies appearing in a firewall policy is essential. The two policy management operations that address these two problems are, respectively, anomaly analysis and resolution, which share a strong connection with each other. First, as described in the survey by Jabal et al. [4], anomaly analysis represents a preliminary step of the anomaly management process, as it consists in assessing the correctness, consistency, and minimality of security policies, with the objective of identifying all their anomalies (e.g., errors and sub-optimizations). In the literature about anomaly analysis, several anomaly classification schemes have been proposed, even though all proposed anomaly classes are still sub-classes of sub-optimizations and conflicts. Then, anomaly resolution works on the anomalies identified by anomaly analysis. Its goal is to remove the conflicts to guarantee a correct firewall behavior, and the sub-optimizations to improve the efficiency of the filtering operations. Several studies have been proposed in literature about these two operations, but they still have limitations. On the one hand, even if several anomaly analysis approaches proposed in literature are accurate and detailed, e.g., the ones described in [5]–[8], they are excessively complex. Besides, most of the existing anomaly analysis approaches do not provide sufficient guidelines to help the administrator resolve the identified anomalies, nor do they provide (semi-)automatic resolution methods coupled with them. As a result, the human administrator is left with having to read the reports produced by those anomaly analysis tools and risks introducing more anomalies because he must apply manual changes that may introduce new sub-optimizations and conflicts, which were not present when the

D. Bringhenti, S. Bussa, R. Sisto, and F. Valenza are with the Politecnico di Torino, Dip. Automatica e Informatica; e-mail: {first.last}@polito.it.

anomaly analysis algorithms were run. On the other hand, approaches that perform anomaly resolution (e.g., [9]–[12]) are not able to solve all anomalies, as it is difficult to identify all the packet classes for which an anomaly type exists in a firewall configuration. Moreover, many of them do not have optimal performance, and do not leave the possibility for the administrator to guide the anomaly resolution.

This paper aims to fill these existing literature gaps by proposing an alternative method for firewall policy anomaly analysis and resolution that can simplify those operations, make them efficient and solve all configuration anomalies at the same time. To this purpose, the main contributions of this paper are the following.

First, we propose a simplified anomaly analysis algorithm. The proposed method uses the concept of atomic predicate, derived from the studies by Yang and Lam [13], [14], for the representation of the packet classes identified by firewall rule conditions. The application of this idea leads to divide the packet space into disjoint packet classes, and to create an alternative representation of the firewall policy, named atomized policy. It is thus possible to identify an anomaly between rules of this atomized policy if their conditions are represented by the same atomic predicate, i.e., if they identify the same packet class. The anomalies thus identified can be duplications or contradictions, depending on whether the actions involved in those anomalies are the same or different. In this way, we achieve a simple and intuitive correspondence to the previously mentioned two classes into which all misconfigurations can be categorized (i.e., sub-optimizations and conflicts). In fact, duplications that may appear in atomized policies are sub-optimizations, while contradictions are conflicts.

Second, we propose an efficient anomaly resolution algorithm that works on the anomalies identified with the approach based on atomic predicates. This algorithm allows the creation of a completely anomaly-free firewall policy, where each rule is disjoint from the others, and it allows to solve the full amount of anomalies that may afflict a firewall policy, independently of they type (i.e., contradiction or duplication). The capability of our algorithm to solve all anomalies in any firewall policy represents a main novelty of our proposal, enabled by the atomic predicate usage, whereas several other state-of-the-art approaches can only solve a limited number of anomalies, after having identified all of them. The proposed resolution strategy also provides high flexibility and user-friendliness, because it supports both a human-assisted anomaly resolution mechanism, and automatic ones based on general requirements requested by the user (e.g., a user may specify that, in case of conflict for the action to be applied on a packet class, "deny" is enforced on it). Besides, as these automatic and semi-automatic mechanisms for anomaly resolution are strictly coupled with the proposed anomaly analysis algorithm, our approach does not expose the administrator to introduce new anomalies, because the human user is not left alone in solving the identified ones.

The framework that we developed for implementing this approach has been validated to assess its efficiency and efficacy. In particular, execution time performance and anomaly management efficiency (i.e., percentage of solved anomalies) are the metrics used in a comparison validation with relevant related work. To provide insight regarding this comparison, our algorithm takes 27s to identify the anomalies in a policy of 90 rules, compared to the 245s taken by the algorithm of Al-Shaer et al. [6]). Besides, it can solve the anomalies in a policy of 132 rules in 0.57s, while the combination and greedy algorithms proposed by Hu et al. [12] take 9.9s e 8.2s, respectively. Besides, our approach can solve all anomalies, while the algorithms of Hu et al. reach lower efficiency (e.g., 86% and 67% for the previously mentioned policy).

The remainder of this paper is structured as follows. Section II discusses related work, highlighting their limitations that our proposal aims to overcome. Section III provides a comprehensive overview of the approach that we propose for both firewall policy anomaly analysis and resolution. Section IV formalizes all the operations of the algorithm. Section V describes the implementation of the proposed approach, and discusses the results of its validation. Section VI discusses possible limitations in the applicability of the proposed algorithm. Finally, Section VII draws conclusions and outlines future work.

II. Related Work

This section discusses related work about firewall policy management, describing the main limitations of related studies about anomaly analysis (Subsection II-A) and resolution (Subsection II-B), and highlighting how the approach proposed in this paper aims to overcome them. It also discusses other formal network management approaches based on the atomic predicate concept, underlining how they leverage that principle to address different problems (Subsection II-C). Many considerations included in this section are based on the findings of the survey carried out by Jabal et al. in [4], which represents the most complete investigation of this literature field.

A. *Firewall policy anomaly analysis*

The milestone study for firewall policy anomaly analysis is the paper by Al-Shaer and Hamed [5], which lays the foundations for all next anomaly detection strategies. Their study proposes a categorization of all anomalies that may arise among intra-firewall policy rules, i.e., filtering rules that are part of the configuration of the same firewall instance. Specifically, it identifies four intra-firewall policy anomaly classes: shadowing, correlation, generalization, and redundancy anomalies. This preliminary study was extended by Al-Shaer et al. [6], so as to propose an anomaly classification scheme that may also apply to inter-firewall policy rule anomalies, so as to detect sub-optimizations and conflicts that may arise due to the configuration of multiple serially-connected firewalls.

Both studies propose simple algorithms to identify firewall anomalies for which they present the respective classification schemes. Therefore, next studies tries to improve them, alongside with alternative classification schemes. The next most significant studies which propose anomaly detection techniques are [7], [8], [15]. First, the static analysis approach proposed by Yuan et al. [7], named FIREMAN, models firewall rules using binary decision diagrams. It can thus use them to execute a symbolic model checking technique covering every path of those diagrams, so that the firewall configuration is analyzed for all possible IP packet classes. Second, Golnabi et al. [8] extend Al-Shaer et al.'s analysis using a data mining technique, named association rule mining. The anomaly detection based on this mining technique exposes many hidden but not detectable anomalies by analyzing only the firewall policy rules. In particular, such approach allows the identification of two new non-systematic misconfiguration anomalies: blocking existing service and allowing traffic to non-existing service anomalies. The first misconfiguration case blocks legitimate traffic from a trusted network to an "existing" service, while the other case of the misconfiguration permits traffic destined for a non-existing service. Then, Bouhoula et al. [15] propose a different approach, based on an inference system to detect intra-firewall policy anomalies. They use the inference system to construct a tree representation of the policy, so that this process stops the construction of a specific branch when no anomaly is found.

The main limitations of all these studies are that their analysis techniques are very complex, even if accurate and detailed, and that they are not paired with anomaly resolution strategies. As these studies were published between 2003 and 2008, they were the first attempt at addressing problems related to firewall policy management and could solve it only partially. Specifically, after the strategies proposed in these studies identify an anomaly, they are not removed or solved from the original policy. Therefore, human administrators would still be responsible for solving the detected anomalies, with the risk of introducing others. Besides, the extensions introduced by all studies that followed [6] were not extensive and did not have the same impact, so [6] remains a main reference for anomaly analysis (but not for resolution).

Our approach aims to overcome all these limitations. On the one hand, it is based on a simpler analysis technique. The simplicity derives from the fact that it can be applied to a simpler representation of the firewall policy rules, based on the atomic predicate concept, where only duplications and contradictions may be present. On the other hand, our proposed approach also embeds multiple variants of an anomaly resolution algorithm, so that either the anomalies are automatically solved or human administrators are assisted in their resolution.

B. Firewall policy anomaly resolution

A second class of studies ([11], [12], [16]–[21]) also include anomaly resolution for traditional firewall policies, to go beyond proposing a simple variant or minimal extension of existing anomaly analysis techniques. However, all of them have shortcomings that our approach aims to overcome.

Some of these approaches ([16], [17]) can only detect and solve sub-optimizations. On the one hand, Liu et al. [16] address the problem of identifying and removing two sub-optimization categories, named upward redundant rules and downward redundant rules. Upward redundant rules are rules that are never matched, whereas downward redundant rules are rules that are matched but enforce the same action as rules with lower priority. Their technique uses a model based on a data structure named firewall decision diagram. On the other hand, Jeffrey et al. [17] suggest using a SAT solver for redundancy analysis. Their problem formulation reduces complexity, and achieves higher performance. Differently from this group of studies, our approach can also identify and solve contradictions which arise when the conditions of rule with different actions match the same packets. Nonetheless, contradictions also represent the most dangerous anomaly type for a firewall configuration.

Instead, Cuppens at al. [18] propose an anomaly resolution approach for solving two intra-firewall policy anomaly types among the ones that were originally defined in [5]: shadowing and redundancy anomalies. Their technique has also been later extended to support inter-firewall anomaly analysis [19]. The same technique has been adopted by the same authors in MIRAGE [20], where they address the resolution problem only for shadowing and redundancy anomalies. The authors motivate their choice stating that those two cases are the ones they consider as serious errors within firewall configurations. However, as extensively demonstrated by the literature about firewall anomaly analysis discussed in Subsection II-A, limiting a study to those two anomaly classes may hinder a correct and efficient behavior for a firewall. Besides, the resolution strategy proposed in MIRAGE [18]–[20] is fully automated, and therefore it does not allow a human user to take decisions in solving the anomalies. Differently from them, our approach aims to address all the possible firewall policy anomalies, and also allows human users to be assisted in solving them if they require it.

The proposal by Hu et al. [11], [12] represents the main proposal among the ones about traditional firewall policy anomaly resolution for its impact on this research area. In fact, by proposing a rule-based segmentation technique, a grid-based representation to identify policy anomalies, and a policy reordering algorithm for anomaly resolution, this study represents a significant step ahead with respect to the previous literature. Specifically, it broadens the addressed anomaly categories to all the ones defined in [5], and it introduces the idea of dividing the packet spaces representing rule conditions into sub-spaces to identify possible intersections. Nevertheless, this study has two main limitations to respect to our proposal. First, the anomaly resolution algorithm does not directly used the information derived from the segmentation of the rule condition packet spaces, which thus has a validity restricted to anomaly analysis. Second, their reordering algorithm does not guarantee that all policy conflicts and sub-optimizations are actually removed, because there may be cases where no ordering can be found to achieve a complete anomaly resolution.

More recently, also Li et al. [22] has proposed a priority-based anomaly-free mechanism to resolve anomalies by adjusting the priority of existing rules instead of rewriting them. However, despite being published more recently, their study shares the same limitations as the one by Hu et al. [11], [12] (i.e., reordering the rules of an existing policy does not guarantee that all anomalies are solved), and its applicability is less broad because only validated for 5G-based networks.

With the advent of network virtualization, Software-Defined

Networking (SDN) reshaped the traditional vision of network infrastructures, providing higher flexibility and dynamism in the creation of service function graphs, and improved mechanisms for traffic control [23]. In this context, SDN switches may be configured by network controllers with protocols, such as OpenFlow, to behave as firewalls. Therefore, the joint problem of anomaly analysis and resolution has also been investigated for this type of filtering function. However, the proposed solutions in literature are not as feature-complete as our approach. On the one hand, some proposed techniques can only solve specific classes of anomalies. Specifically, the FlowGuard algorithm discussed by Hu et al. [24] can only solve conflicts with overlapped IP address spaces, but cannot manage conflicts among multiple packet header fields. Instead, Li et al. [25] only focus on the resolution of conflicts related to covert channel attacks, while Cui et al. [26] address the problem for transaction conflicts. On the other hand, few other studies cover a broader range of anomaly types, but still in a limited way. In the approach proposed by [27], the resolution of interpretive conflicts leads to loss of information. Instead, the method by Asif et al. [28] bases the resolution algorithm on merging rule conditions and is more feature-complete than the other alternatives. Still, it does guarantee that, in the end, the computed rules are disjoint as there are no atomic rules, but represents a step ahead in literature with respect to previous studies about SDN switch anomalies.

In summary, to the best of our knowledge, our approach is characterized by manifold features which overcome existing limitations of the state of the art. First, it proposes a joint algorithm for firewall policy anomaly analysis and resolution, instead of focusing on a single operation. Second, it simplifies anomaly analysis, thanks to the policy representation based on atomic predicates. Third, the proposed resolution algorithm aims to solve all anomalies, differently from the other strategies which only order rules of an existing policy without guaranteeing that all anomalies are removed. Fourth, our approach is efficient in terms of performance, as also experimentally shown in Subsection V-C.

C. Formal network management approaches based on the concept of atomic predicates

The concept of atomic predicates was originally proposed by Yang and Lam in [13], [14]. According to their definition, given a set of initial predicates, it is possible to apply an algorithm on them to compute a derived set of atomic predicates, which is proved to be minimum and unique. The main property of this set is that each given predicate is equal to the disjunction of a subset of atomic predicates. Besides, as each atomic predicate is unique, it can be assigned an integer number, and each initial predicate can be stored and represented as a set of integers identifying the atomic predicates that compose it. Through this property, the operations of conjunction and disjunction of two predicates become easier, because they can be computed respectively as the intersection and union of two sets of integers, i.e., the ones representing the atomic predicates that compose the two predicates.

Yang and Lam initially apply the atomic predicate concept to real-time verification of network properties such as loop detection, black hole detection, network slice isolation, and required waypoints [13], and they also show the feasibility of their proposal to networks composed of packet transformers [14]. Following studies leverage this concept to solve the same problem, i.e., verification of data plane properties, but by introducing some incremental features. In particular, Berardi et al. [29] use atomic predicates to verify that a 5G network could be resistant against hijacking and denial-of-service attacks, Zhang et al. [30] use them to detect a security threat existing in software-defined networks called firewall bypass. Some proposals by Zhang et al. [31] and Guo et al. [32] aim to improve the speed through which the verification of network properties is performed. Instead, three other studies try to leverage atomic predicates to solve peculiar problems: service function chaining [33], network security policy refinement [34], and to improve network management tasks, such as dataplane refactoring and network debugging [35].

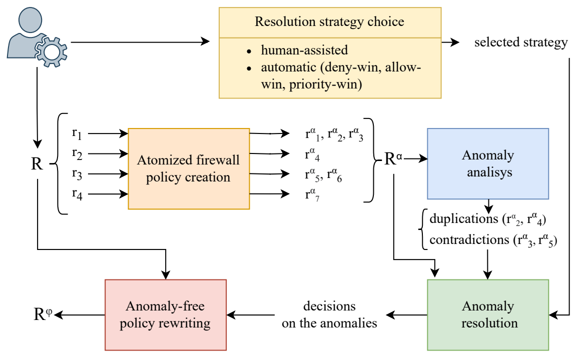

Resolution strategy choice
human-assisted
automatic (deny-win, allow-win, priority-win)
selected strategy
R
r1
r2
r3
r4
Atomized firewall policy creation
rα1, rα2, rα3
rα4
rα5, rα6
rα7
Rα
Anomaly analisys
duplications (rα2, rα4)
contradictions (rα3, rα5)
Anomaly resolution
decisions on the anomalies
Anomaly-free policy rewriting
Rφ

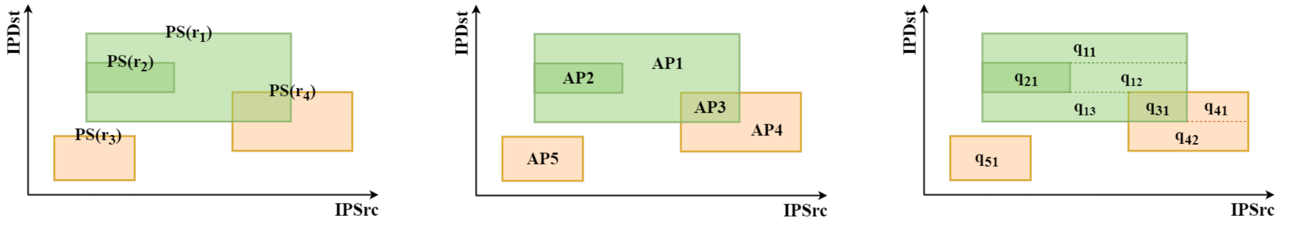
Fig. 1: Workflow of the proposed approach

Most of these studies share the basic algorithm for atomic predicate computation, but then they leverage the actual predicates to solve different problems. We also used atomic predicates as a starting point for our proposal. However, on the one hand, we had to customize the atomic predicate computation algorithm to work with formal models of firewall policy rules. On the other hand, we used them to address a problem, that is anomaly analysis and resolution, for which atomic predicates have never been previously used.

III. The Atomizing Approach for Firewall Policy Anomaly Analysis and Resolution

This section provides a complete overview of the proposed approach, whose workflow is illustrated in Fig. 1. In particular, it describes how it applies the concept of atomic predicates for the representation of firewall policies (Subsection III-A), how it performs firewall policy anomaly analysis (Subsection III-B) and resolution (Subsection III-C), and how it rewrites the firewall policy in an anomaly-free representation (Subsection III-D). Besides, the applicability of the proposed approach to mainstream firewall solutions, and possible extensions for the management of policy anomalies related to other firewall types or for the management of inter-firewall anomalies are also discussed (Subsection III-F).

TABLE I and Fig. 2a are used to provide a clarifying example about all the operations of the proposed approach.



(a) Packet spaces of the firewall rules (b) Atomic predicates of rule conditions (c) Predicates compliant to firewall rules

Fig. 2: Geometric representation of firewall rule packet spaces and atomic predicates

TABLE I: *Original firewall policy R*

#	Action	IPSrc	IPDst	pSrc	pDst	Proto
r_1	Allow	[130.11.2.16, 130.11.2.100]	[42.0.2.32, 42.0.2.86]	*	*	*
r_2	Allow	[130.11.2.16, 130.11.2.42]	[42.0.2.42, 42.0.2.60]	*	*	*
r_3	Deny	[130.11.2.84, 130.11.2.146]	[42.0.2.22, 42.0.2.42]	*	*	*
r_4	Deny	[130.11.2.4, 130.11.2.26]	[42.0.2.2, 42.0.2.28]	*	*	*

TABLE II: *Atomized firewall policy R^α*

# (original)	# (atomic)	Action	Condition
r_1	r_1^α	Allow	AP1
	r_2^α	Allow	AP2
	r_3^α	Allow	AP3
r_2	r_4^α	Allow	AP2
r_3	r_5^α	Deny	AP3
	r_6^α	Deny	AP4
r_4	r_7^α	Deny	AP5

TABLE III: *Atomic predicate division in sub-rectangles*

Atomic predicate	Division in sub-rectangles
AP1	$q_{11} \vee q_{12} \vee q_{13}$
AP2	q_{21}
AP3	q_{31}
AP4	$q_{41} \vee q_{42}$
AP5	q_{51}

#	IPSrc	IPDst	pSrc	pDst	Proto
q_{11}	[130.11.2.16, 130.11.2.100]	[42.0.2.60, 42.0.2.86]	*	*	*
q_{12}	[130.11.2.42, 130.11.2.100]	[42.0.2.42, 42.0.2.60]	*	*	*
q_{13}	[130.11.2.16, 130.11.2.100]	[42.0.2.32, 42.0.2.42]	*	*	*
q_{21}	[130.11.2.16, 130.11.2.42]	[42.0.2.42, 42.0.2.60]	*	*	*
q_{31}	[130.11.2.84, 130.11.2.100]	[42.0.2.32, 42.0.2.42]	*	*	*
q_{41}	[130.11.2.100, 130.11.2.146]	[42.0.2.32, 42.0.2.42]	*	*	*
q_{42}	[130.11.2.84, 130.11.2.146]	[42.0.2.22, 42.0.2.32]	*	*	*
q_{51}	[130.11.2.4, 130.11.2.26]	[42.0.2.2, 42.0.2.28]	*	*	*

TABLE IV: *Rewritten anomaly-free firewall policy R^ϕ*

#	Action	IPSrc	IPDst	pSrc	pDst	Proto
r_1^ϕ	Allow	[130.11.2.16, 130.11.2.100]	[42.0.2.60, 42.0.2.86]	*	*	*
r_2^ϕ	Allow	[130.11.2.16, 130.11.2.42]	[42.0.2.42, 42.0.2.60]	*	*	*
r_3^ϕ	Allow	[130.11.2.42, 130.11.2.100]	[42.0.2.42, 42.0.2.60]	*	*	*
r_4^ϕ	Allow	[130.11.2.16, 130.11.2.84]	[42.0.2.32, 42.0.2.42]	*	*	*
r_5^ϕ	Deny	[130.11.2.4, 130.11.2.26]	[42.0.2.2, 42.0.2.28]	*	*	*
r_6^ϕ	Deny	[130.11.2.84, 130.11.2.146]	[42.0.2.22, 42.0.2.42]	*	*	*

TABLE I lists four rules belonging to the same firewall policy for which some anomalies exist, whereas Fig. 2a graphically depicts the packet spaces matching their conditions. They can be geometrically represented as penteracts, which are five-dimensional hypercubes, as also shown in studies such as [36]–[38]. Each dimension of the hyperspace where these penteracts lie corresponds to one of the five fields of the IP 5-tuple over which the firewall rule condition is defined, i.e., source and destination IP addresses (IPSrc, IPDst), source and destination ports (pSrc, pDst), protocol type (Proto). Here, for sake of visualization simplicity, the conditions of all rules specified in TABLE I only select packets depending on the values of their source and destination IP addresses. In this way, it is possible to represent their packet spaces as rectangles in a 2-space.

A. *Atomized firewall policy creation*

The first step of our approach, shown in Fig. 1, consists in writing the original firewall policy, symbolized as R, in an alternative representation, named atomized policy and symbolized as R^α. In order to achieve this objective, it is first required to model the packet spaces represented by the firewall rule conditions as predicates. Then, these potentially complex predicates are split into disjoint simpler predicates, named atomic predicates, where each one represents a specific packet sub-class. Each one of these simpler predicates represents either the intersection among the predicates representing rule conditions, or the packet sub-space that each rule condition does not have in intersection with others. After these preliminary operations, the atomized policy R^α can be finally created by decomposing the original rules of R into multiple rules, named atomic rules. In particular, for each atomic predicate into which the condition of a rule of R is split, an atomic rule is created in R^α, having that atomic predicate as condition and the same action of the rule from which it derives.

The creation of this atomized firewall policy is inspired by the idea of atomic predicates originally proposed in [13], [14]. According to those studies, given a set of predicates on packet fields, it is possible to compute the set of totally disjoint and minimal predicates, named atomic predicates, such that each predicate can be expressed as a disjunction of a subset of atomic ones. Applying this concept to a firewall policy, it is possible to split each complex predicate representing a rule condition into a disjunction of simpler and minimal atomic predicates, each one representing a sub-space of the packet space identified by the condition itself. Each atomic predicate can thus be used as condition of an atomic rule. This guarantees a fine granularity level, as each packet sub-space considered for anomaly analysis is the minimal one. As all atomic predicates are disjoint for definition, each one can also be associated with an integer number that uniquely identifies it, thus simplifying all operations on them. Besides, this transformation based on atomic predicates decomposes complex predicates into simpler predicates, preserving all original information without adding any more or removing any. Therefore, its application does not sacrifice soundness or completeness.

The atomized policy that is created starting from the example of TABLE I and Fig. 2a is reported in TABLE II and graphically shown in Fig. 2b. The original rule r_1 is divided into three atomic rules: r_1^α, r_2^α, and r_3^α. The condition of r_1^α is the atomic predicate AP1, which represents the packet sub-space of the condition of r_1 that is not intersected with any rule condition. The condition of r_2^α is the atomic predicate AP2, which represents the packet sub-space of the condition of r_1 that is intersected with the condition of r_2. Similarly, the condition of r_3^α is the atomic predicate AP3, which represents the packet sub-space of the condition of r_1 that is intersected with the condition of r_3.

The condition of rule r_2 is totally represented by a single atomic predicate, AP2, because it is entirely intersected with the condition of r_1. Therefore, a single atomic rule r_4^α having AP2 as condition derives from r_2. Similarly as for r_1, multiple atomic rules derives from r_3: the atomic rule r_5^α with condition AP3, representing the intersection with the condition of r_1, and the atomic rule r_6^α with condition AP4, representing the non-intersected packet sub-space. Finally, the condition of rule r_4 has no intersections, so a single atomic rule r_7^α with the atomic predicate AP5 as condition is derived.

B. Firewall policy anomaly analysis

After computing all atomic rules, it is possible to identify the presence of an anomaly for a pair of them if their conditions are represented by the same atomic predicate. In fact, this means that there is an intersection between the packet spaces representing the packets that match the conditions of the original rules from which the two atomic ones derive.

Depending on the rule actions, only two anomaly types satisfying this definition exist: (i) a *duplication*, if the two atomic rules have the same atomic predicate as condition and have the same action (i.e., both actions are “allow“ or “deny“); (ii) a *contradiction*, if the two atomic rules have the same atomic predicate as condition and have different actions (i.e., an action is “allow“, the other is “deny“).

Thanks to the previous creation of the atomized firewall policy, the method for anomaly analysis is much easier than state-of-the-art alternatives, and understanding how firewall rules are sub-optimal or contradicting is more intuitive for human administrators.

For example, considering again Fig. 2b, which depicts the sub-spaces related to the atomic predicates representing the conditions of the atomic rules of TABLE II, two anomalies can be easily identified. On the one hand, the atomic predicate AP2 represents the condition of both atomic rules r_2^α and r_4^α. The anomaly associated to the relationship between these two atomic rules is a duplication, because the actions of both rules are “allow”. On the other hand, the atomic predicate AP3 represents the condition of both atomic rules r_3^α and r_5^α. In this case, the associated anomaly is a contradiction, because the action of r_3^α is “allow”, while the action of r_5^α is “deny”.

C. Firewall policy anomaly resolution

After all duplications and contradictions have been identified, the next step of the proposed methodology consists in resolving them, by deciding which action must be applied to any packet sub-space represented by an atomic predicate which is the condition of multiple atomic rules. The output of these decisions will be used to rewrite the original firewall policy so as to make it anomaly free.

The decision process to establish the action to be applied on a packet sub-space represented by an atomic predicate which is the condition of a pair of atomic rules varies depending on the anomaly type. If the anomaly is a duplication, the decision is trivial, because the action is the same as those of the original rules. Instead, if the anomaly is a contradiction, the resolution is not immediate and it requires a specific resolution strategy selected by the user. In particular, two alternative classes of resolution strategies are envisioned in our method: human-assisted and automatic resolution. In the former, the user is asked to take a decision for each contradiction, while duplications are automatically solved as previously explained. This strategy is named human-assisted, because all decisions related to contradiction resolution are manually taken by a human. In the latter, all decisions are automatically taken, according to a guideline specified by the user. Examples of automatic resolution strategies are: (i) deny-win, if the “deny” action must be applied on all the packets related to an anomaly; (ii) allow-win, if the “allow” action must be applied on all the packets related to an anomaly; (iii) priority-win, if the action to be applied on all the packets related to an anomaly is the action of the rule having higher priority. Our approach is flexible enough to support other automatic resolution strategies, which can be introduced without altering the flow of the methodology (e.g., the action that is enforced is the one of the firewall rule that has been introduced in the configuration earliest). Besides, the simplicity of this anomaly resolution technique is precisely one of the strengths of our proposal. This degree of simplicity is possible because atomic predicates are used to create atomic rules.

For example, let us consider again the atomized policy of Fig. 2b and TABLE II, and suppose that a deny-win resolution strategy is selected by the user. The atomic predicate AP2 is associated with an anomaly of type duplication, and therefore the action that must be enforced on the packets represented by AP2 is simply “allow”, which are the same as those of the two duplicated atomic rules r_2^α and r_4^α. Instead, the two atomic rules that have the atomic predicate AP3 as condition, r_3^α and r_5^α, are in contradiction, because their actions are opposite. In view of the selected strategy, the “deny” action must be thus enforced on the packets identified by AP3.

D. Firewall anomaly-free policy rewriting

As shown in Fig. 1, the final operation of the approach consists in rewriting the original firewall policy so as to make it anomaly free, avoiding that a packet sub-space matches with the condition of more than one rule of the rewritten set. This operation employs the output of the previous decisions about which action must be applied to any packet sub-space represented by an atomic predicate. The resulting firewall policy is symbolized as R^{ϕ}.

This operation should ensure that, for each packet sub-space identified by an atomic predicate, only the action decided in the previous step is performed, and it is part of the condition of at most one rule. In particular, if all the packet sub-spaces represented by the atomic predicates intersecting with an original rule condition are still managed with its action and neither of them is included in the condition of a rule of the new policy that is being created, then the original rule is simply preserved in the new anomaly-free policy. Otherwise, for each packet sub-space of the original rule condition that is not already included in the condition of a rule of the new policy that is being created, a separate new rule is created, characterized by the corresponding atomic predicate as condition and the action decided in the previous step.

However, it is not always possible to create rules with atomic predicates themselves as conditions because some may represent packet sub-spaces that cannot be expressed as well-formed firewall conditions. Fig. 2c and TABLE III are here used to clarify this concept, as they show how each atomic predicate can be divided into rectangular sub-spaces. In fact, an atomic predicate can be used as a rule condition if its shape is rectangular, as it means that its source and destination IP addresses can be expressed as two intervals of contiguous values. Considering, for example, the sub-spaces expressed by atomic predicates AP2, AP3 and AP5 of Fig. 2b, their division in sub-rectangles reported in Fig. 2c and TABLE III show that each one of them is a rectangle by itself (q_{21}, q_{31}, and q_{51}, respectively). Consequently, they may be directly used as well-formed rule conditions. Instead, considering the sub-spaces expressed by atomic predicates AP1 and AP4 of Fig. 2b as another example, they are not rectangular. As graphically depicted in in Fig. 2c and shown in TABLE III, the packet sub-space of AP1 can be divided into three sub-rectangles q_{11}, q_{12}, and q_{13}, while the the packet sub-space of AP4 can be divided into two sub-rectangles q_{41} and q_{43}. It is thus possible to create firewall rules with these sub-rectangles as conditions.

Even if this algorithm may create a larger policy than the original one, nowadays this does not represent a serious issue for some modern implementations of packet filtering firewalls, which do not use a linear search anymore, but strategies that are even more efficient if applied on disjoint rules [39], [40]. At the same time, as expanding the number of rules excessively would require a larger amount of memory, which is a scarce resource in firewalling devices, as previously explained, the proposed algorithm for final policy rewriting tries to keep the original aggregated rule whenever possible, i.e., if all the packet sub-spaces defined by atomic predicates intersecting with that original rule are still associated with its action, and none are already covered by another rule in the new policy.

TABLE IV reports the rewritten anomaly-free firewall policy R^{ϕ}, supposing again that a deny-win strategy has been selected to solve the anomalies of the atomized policy in TABLE II. According to the result of anomaly resolution, the packet sub-space represented by the atomic predicate AP2 is associated with the "allow" action, while the packet sub-space represented by the atomic predicate AP3 is associated with the "deny" action. Therefore, the original rules r_2 and r_4 are directly preserved in the rewritten policy, where they are r_2^{ϕ} and r_6^{ϕ}, because their conditions are fully represented by AP2 ane AP3 respectively, and their actions are the same as those decided in the resolution step. Then, as r_3 does not have any anomaly with other rules and its condition is fully represented by the single atomic predicate AP5, then it can be preserved as well, and it is included in the anomaly-free policy as r_5^{ϕ}. The only packet sub-space that still needs to be covered in the final policy R^{ϕ} is the one represented by the atomic predicate AP1. However, this sub-space is not well formed for representing a rule condition, as it is not a rectangle. Therefore, three separate rules (r_1^{ϕ}, r_3^{ϕ}, and r_4^{ϕ}) are created, having as conditions the packet sub-spaced represented by q_{11}, q_{12}, and q_{13}.

E. Contributions of the designed approach

Even if the proposed approach has been designed using the atomic predicate computation algorithm as a starting point, it provides several contributions to the related literature, which can be clarified now that all the steps of the designed workflow have been described.

The algorithm for atomic predicate computation is used only for a single operation included in the first stage of the workflow depicted in Fig. 1, i.e., for transforming the predicates representing the initial firewall policy conditions into the corresponding atomic predicates. Then, after this preliminary step, all the next steps of our approach have been designed by us without leveraging or customizing any other existing algorithm of the literature. In other words, the rest of the method for the creation of the atomized policy (Subsection III-A), the simplified anomaly analysis strategy based on the new anomaly classification scheme characterized by contradictions and duplications (Subsection III-B), the anomaly resolution method with the resolution strategy choices involving the human user (Subsection III-C), and the anomaly-free policy rewriting (Subsection III-D) are entirely new and represent the main contributions offered by our design. Therefore, the atomic predicates computation algorithm, used at the first stage, simply represents a starting point, allowing us to design a completely new methodology, which goes beyond what the literature described in Section II offers. Moreover, also the design for the atomic predicate computation algorithm has been customized, as it will be explained and formalized in Section IV-A.

F. Applicability and possible extensions of the proposed approach

The proposed approach has been designed to be applied successfully to practical mainstream packet filtering firewall products.

Typically, those firewalls are characterized by a resolution strategy (e.g., First Matching Rule, Deny Takes Precedence, Most Specific Takes Precedence) determining which action must be applied to packets matching the conditions of multiple rules. Thanks to this, even if a rule ordering is afflicted by anomalies (e.g., a rule is shadowed by another one with higher priority), the application of a specific resolution strategy may sometimes hide the existence of those anomalies, which will not thus impact the filtering outcome. For this reason, the literature defines them as "potential anomalies", because they may potentially impact the correct decisions of a firewall, and thus become "real anomalies", only depending on the used resolution strategy. However, if there are potential anomalies, then the problem of understanding what resolution strategies could be used arises. Therefore, the best thing to do is to remove all potential anomalies as well. In this way, through the application of our proposed approach that can solve all anomalies, also real anomalies will be surely removed.

Additionally, the proposed approach can also be extended to support other kinds of firewall solutions, such as application-layer firewalls and SDN switches, and to work within inter-firewall anomalies.

1) Extension to application-layer firewalls or SDN switches: This approach for anomaly analysis and resolution can be extended to solve the same problem, but related to firewall types whose rule conditions are characterized by more than 5 dimensions, such as application-layer firewalls and SDN switches. This extension would require to model other fields (e.g., HTTP method, web domain, MAC addresses, VLAN tags) as additional predicates, to be included among the predicates composing the model of a firewall rule condition. If for a packet filtering firewall the condition is modeled as a conjunction of five predicates, for an application-layer firewall or an SDN switch it would thus be modeled as a conjunction of more predicates. Then, all the other algorithms, including the one for firewall policy atomization, could still be applied to the modified model.

2) Extension to inter-firewall anomalies: This approach can also be extended to inter-firewall scenarios, where each instance of a distributed firewall architecture has its own policy. In those scenarios, anomalies may be relationships between policy rules of different firewall instances, crossed by the same traffic flow. Some changes need to be applied to the steps of the proposed methodology to adapt it to that case.

First, the atomic predicate computation must be performed simultaneously on the rule conditions of the policies of all instances of the distributed firewall. This change is required so as to consider all possible intersections among rules not only within the same policy, but also belonging to different policies. After that, each single policy can be atomized independently from the other ones, using the atomic predicates computed simultaneously over all firewall rules. Second, the anomaly analysis can still identify duplications and contradictions in rule pairs. The only difference is that the anomalies may be intra-firewall duplications and contradictions if they afflict rules belonging to the same policy, or inter-firewall duplications or contradictions if they afflict rules belonging to different policies. Third, for what concerns anomaly resolution, in view of the complexity of this problem for inter-firewall anomalies, the strategy that is suggested is the human-assisted. The reason is that, for an automatic resolution of inter-firewall policy anomalies, it would be necessary to know the direction of each possible traffic flow, as also discussed in [6]. However, such knowledge cannot be known a-priori when analyzing a distributed firewall policy offline. Fourth, each firewall instance policy is simply rewritten as already discussed for the intra-firewall case, by using the atomic predicates computed simultaneously over all firewall rules.

At the same time, a main challenge that would arise in this extension relates to the size of the overall rule set. If, as observed in literature, a centralized firewall might have a few hundred rules, a distributed firewall might be characterized by thousands of them in order to handle malicious traffic generated by potentially millions of different sources. Consequently, in addition to the abovementioned changes, parallelization strategies might be necessary for good performance.

IV. Algorithm

This section formalizes the approach that is followed for the proposed anomaly and resolution strategy. First, it introduces the model used for the representation of a firewall policy (Subsection IV-A). Then, it describes the formalization of the four operations composing the methodology: the atomized firewall policy creation (Subsection IV-B), the firewall policy anomaly analysis and resolution (Subsection IV-C), and the final operation of rewriting the original rule list into an anomaly-free policy (Subsection IV-D).

A. Firewall policy model

A firewall policy is modeled as an (ordered) rule list:

$$R = [r_1, r_2, ..., r_{|R|}] \tag{1}$$

where each rule is a tuple $r_i = (c, a)$. In the proposed model, the '.' notation, when applied to a tuple, is used to retrieve a specific tuple element. As the policy is ordered, if r_{i_1} precedes r_{i_2}, the firewall checks if a packet satisfies $r_{i_1}.c$ (and, if so, it applies $r_{i_1}.a$) before checking the satisfaction of $r_{i_2}.c$.

The element $r_i.c$ is the rule condition that allows to identify the packet classes to which the action $r_i.a$ must be applied. Specifically, the rule condition is modeled as the conjunction of five predicates, one for each field of the IP 5-tuple:

$$r_i.c = r_i.c.x_1 \land r_i.c.x_2 \land r_i.c.x_3 \land r_i.c.x_4 \land r_i.c.x_5 \tag{2}$$

The two predicates $r_i.c.x_1$ and $r_i.c.x_2$ respectively express conditions on the source and destination IP addresses. Each of them can identify a single IP address (e.g., $r_i.c.x_1$ = 127.10.22.3) or a range of contiguous IP addresses (e.g., $r_i.c.x_1 = [r_i.c.x_{1,begin}, r_i.c.x_{1,end}]$ = [127.10.22.0, 127.10.22.255]). Instead, the two predicates $r_i.c.x_3$ and $r_i.c.x_4$ respectively express conditions on the source and destination port numbers. Each of them can identify a single port number (e.g., $r_i.c.x_3$ = 80) or a range of contiguous port numbers (e.g, $r_i.c.x_3 = [r_i.c.x_{3,begin}, r_i.c.x_{3,end}]$ = [22, 79]). Finally, $r_i.c.x_5$ identifies a specific protocol type (e.g., TCP or UDP). For each field of the IP 5-tuple, the set of all the possible values is concisely symbolized by the wildcard $*$.

The element $r_i.a$ is the rule action that is enforced onto the packets satisfying the rule condition. The rule action can be "allow" if the packets matching its conditions must be forwarded by the firewall to the next network node, whereas "deny" if they must be discarded.

Finally, the last rule of the firewall policy is expressed as $r_{|R|} = (*, a)$. The action $r_{|R|}.a$ is commonly named firewall default action, as it is applied to all the packets that do not satisfy the condition of any other policy rule.

B. Atomized firewall policy creation

The creation of the atomized firewall policy from the original policy R requires two preliminary steps, which are (1) computing the atomic predicates related to all rule conditions predicates, and (2) creating atomic rules, where the condition of each one is represented by a single atomic predicate. These two steps are formalized in Algorithm 1.

First, the algorithm computes the set of atomic predicates C related to all rule conditions, by applying function $\mathcal{A}$ to the set of all condition predicates $\mathcal{P}$ (lines 1-4). $\mathcal{A}$ is a standard function, described as Algorithm 3 in [14], that, given a set of predicates, splits them according to their mutual intersections, so as to compute atomic predicates that are disjoint to each other. This function represents the only algorithmic part we have mutated from the literature. All the rest of Algorithm 1 and all the next Algorithms here presented are entirely novel. Besides, the original version of $\mathcal{A}$ works on predicates representing packet classes defined such that each predicate is a Boolean formula with a variable for each packet header bit. Instead, the model that we use, illustrated in (2), defines the predicate representing a rule condition and, consequently, a packet class as a conjunction of five sub-predicates, one for each field of the IP 5-tuple. This different modeling choice is motivated by the fact that, differently from [14], we explicitly work with packet filters and do not need to have a complexity as significant as the one that characterizes their model. So we simply adapted that algorithm to work with this different predicate model by applying all operations there included (e.g., intersections and unions) not to single bits, but to packet fields, while keeping the same algorithmic logic.

Each atomic predicate c_k of the output set C is the disjunction of multiple predicates q_{kh}, where each q_{kh} expresses conditions related to the IP 5-tuple fields of network packets, and it is similarly modeled as $r_i.c$:

$$c_k = \bigvee_{h=1}^{H} q_{kh}, \text{with } q_{kh} = \bigwedge_{l=1}^{5} q_{kh}.x_l \tag{3}$$

It is possible that an atomic predicate c_k is equal to a single q_{kh}, i.e., when $H = 1$. Moreover, given a q_{k_1h} associated to an atomic predicate c_{k_1}, the packet classes that it identifies cannot be identified by any other q_{k_2h} associated to a different atomic predicate c_{k_2}.

Second, the algorithm creates an alternative representation of R, where each rule has a condition expressed by a single atomic predicate (lines 5-13). The notation used for this version, named *atomized* policy or rule list, is R^α. Starting from a rule r_i of the original policy, the algorithm checks if its condition $r_i.c$ intersects any atomic predicate $c_k \in C$. For each intersecting atomic predicate c_k, it creates a new rule r_j^α, having c_k as condition and $r_i.a$ as action. The algorithm also keeps track of the the connection between each created atomic rule and the original policy rule from which it derives, by means of the o array. In particular, $o[j]$ is set to the integer number i if the atomic rule r_j^α derives from r_i.

Algorithm 1 for creating the atomized policy

Input: the rule list R
Output: the rule list R^α, the array o

```
 1: $\mathcal{P} \leftarrow \{\text{false}\}$
 2: for $i = 1, 2, ..., |R|$ do
 3:     $\mathcal{P} \leftarrow \mathcal{P} \cup \{r_i.c\}$
 4: $C \leftarrow \mathcal{A}(\mathcal{P})$
 5: $R^\alpha \leftarrow []$
 6: $j \leftarrow 1$
 7: for $i = 1, 2, ..., |R| - 1$ do
 8:     for $k = 1, 2, ..., |C|$ do
 9:         if $r_i.c \wedge c_k$ then
10:             $r_j^\alpha \leftarrow (c_k, r_i.a)$
11:             $R^\alpha \leftarrow R^\alpha + [r_j^\alpha]$
12:             $o[j] \leftarrow i$
13: return $R^\alpha, o$
```

The worst-case time complexity of Algorithm 1 can be estimated as the sum of the time complexities of three sequential code blocks. Lines 2-3 have $O(|R|)$ complexity, because $O(1)$ operations are performed on each one of the $|R|$ input rules. Line 4 has $O(\mathcal{A})$ complexity, where $\mathcal{A}$ is the external function that is called to compute the atomic predicates of a given set of input predicates. Referring to the $\mathcal{A}$ presented in [14], this function is linear in the number of input predicates in $\mathcal{P}$. Here, in the worst case, the size of $\mathcal{P}$ is equal to the size of R, so $O(\mathcal{A})$ would be $O(|R|)$. Lines 7-12 have $O(|R| \cdot |C|)$ complexity because the algorithm has to iterate over the entire set of atomic predicates for each rule of the original firewall policy. Therefore, the overall worst-case time complexity of Algorithm 1 is $O(|R| \cdot |C|)$, which is the dominant term. However, this algorithm can be parallelized, potentially by associating one thread to each original rule that must be atomized. This parallelization can thus practically counterbalance the theoretical worst-case complexity.

C. Firewall policy anomaly analysis and resolution

The creation of the atomized policy R^α allows the actual anomaly identification. Given two atomic rules $r_{j_1}^\alpha$ and $r_{j_2}^\alpha$ with $j_1 \neq j_2$, only two anomaly types may afflict them:

- a *duplication* if $r_{j_1}^\alpha.a = r_{j_2}^\alpha.a$ and $r_{j_1}^\alpha.c = r_{j_2}^\alpha.c$;
- a *contradiction* if $r_{j_1}^\alpha.a \neq r_{j_2}^\alpha.a$ and $r_{j_1}^\alpha.c = r_{j_2}^\alpha.c$;

If $r_{j_1}^\alpha.c \neq r_{j_2}^\alpha.c$, no anomaly afflicts the two atomic rules.

Then, the anomaly resolution algorithm has the objective of removing all the duplications and contradictions, as previously defined, that afflict an atomized firewall policy R^α.

As already discussed in Section III, the user of this methodology can express their preference among a human-assisted resolution strategy and fully automated resolution strategies, such as deny-win, allow-win, and priority-win. In the former, the user is asked to take a decision for each contradiction, while duplications are automatically solved. In the latter, all decisions are automatically taken, according to guidelines derived from the selected strategy.

In all cases, the objective of the policy anomaly resolution algorithm is to compute two sets, identified by the A and

Algorithm 2 for anomaly resolution (human-assisted strategy)

```
Input: the rule list $R^\alpha$, the array $o$
Output: the sets $A$ and $D$, the array $altered$
 1: $A \leftarrow \emptyset$, $D \leftarrow \emptyset$
 2: for $i = 1, 2, ..., |R| - 1$ do
 3:     $altered[i] \leftarrow$ false
 4: for $j = 1, 2, ..., |R^\alpha|$ do
 5:     if $r^\alpha_j.a$ = allow $\wedge\ r^\alpha_j.c \notin A$ then
 6:         $A \leftarrow A \cup \{r^\alpha_j.c\}$
 7:     else if $r^\alpha_j.a$ = deny $\wedge\ r^\alpha_j.c \notin D$ then
 8:         $D \leftarrow D \cup \{r^\alpha_j.c\}$
 9:     else
10:         $altered[o[j]] \leftarrow true$
11: for each $c_k \in (A \cap D)$ do
12:     $decision \leftarrow$ input(Select 'A' or 'D':)
13:     if $decision$ = 'A' then
14:         $D \leftarrow D \backslash \{c_k\}$
15:         for $j = 1, 2, ..., |R^\alpha|$ do
16:             if $c_k = r^\alpha_j.c \wedge r^\alpha_j.a$ = deny then
17:                 $altered[o[j]] \leftarrow$ true
18:     else if $decision$ = 'D' then
19:         $A \leftarrow A \backslash \{c_k\}$
20:         for $j = 1, 2, ..., |R^\alpha|$ do
21:             if $c_k = r^\alpha_j.c \wedge r^\alpha_j.c$ = allow then
22:                 $altered[o[j]] \leftarrow$ true
23: return $A$, $D$, $altered$
```

Algorithm 3 for anomaly resolution (deny-win or allow-win)

```
Input: the rule list $R^\alpha$, the array $o$, the winning action $a^W$
Output: the sets $A$ and $D$, the array $altered$
$A \leftarrow \emptyset$, $D \leftarrow \emptyset$
for $i = 1, 2, ..., |R| - 1$ do
    $altered[i] \leftarrow$ false
for $j = 1, 2, ..., |R^\alpha|$ do
    if $r^\alpha_j.a = a^W$ then
        if $a^W$ = allow $\wedge\ r^\alpha_j.c \notin A$ then
            $A \leftarrow A \cup \{r^\alpha_j.c\}$
        else if $a^W$ = deny $\wedge\ r^\alpha_j.c \notin D$ then
            $D \leftarrow D \cup \{r^\alpha_j.c\}$
        else
            $altered[o[j]] \leftarrow$ true
for $j = 1, 2, ..., |R^\alpha|$ do
    if $r^\alpha_j.a \neq a^W$ then
        if $a^W$ = allow $\wedge\ r^\alpha_j.c \notin (A \cup D)$ then
            $D \leftarrow D \cup \{r^\alpha_j.c\}$
        else if $a^W$ = deny $\wedge\ r^\alpha_j.c \notin (A \cup D)$ then
            $A \leftarrow A \cup \{r^\alpha_j.c\}$
        else
            $altered[o[j]] \leftarrow$ true
return $A$, $D$, $altered$
```

D letters, which respectively contain the atomic predicates related to packet classes that must be allowed or blocked by the firewall. As previously defined, each atomic predicate is associated with a representative integer number, so that A and D are sets of integers. This representation guarantees that each $c_k \in C$ can appear once in a set, thus removing any duplication. Besides, the algorithm must also ensure that, given an atomic predicate $c_k \in C$, it can appear only in one of the two sets, i.e., $c_k \in (A \cup D) \wedge c_k \notin (A \cap D)$. If it appeared in both sets, a contradiction would still be present. Besides, the algorithm computes a Boolean array named *altered*. Specifically, $altered[i]$ = true if there is at least an atomic rule r^α_j, such that $o[j] = i$, whose condition $r^\alpha_j.c$ has not been included in either A or D. Both sets of atomic predicates and the *altered* array will be used to rewrite the anomaly-free policy, as we will show in Subsection IV-D.

Even if all the proposed anomaly resolution strategies may be jointly designed as a single algorithm, we preferred defining a separate algorithm for each one of them. Each algorithm can be specifically optimized to minimize the number of operations that must be executed to satisfy the user-selected guideline, thus maximizing the overall performance. In the following, we describe the formalization of the algorithms for the different anomaly resolution strategies.

Human-assisted strategy: Algorithm 2 formalizes the human-assisted resolution strategy. Initially, the algorithm automatically removes the existing duplications. Specifically, for each atomic rule $r^\alpha_j \in R^\alpha$, the atomic predicate corresponding to its condition $r^\alpha_j.c$ is included into the A or D set, depending on the rule action (lines 1-10). This step ensures removal of all duplications, as the integer number representing a predicate c_k cannot appear more than once in a set. Then, for each predicate c_k that is included in both A and D sets, the user is asked to decide if the action that must be enforced on the packets represented by that predicate is "allow" or "deny". At that point, c_k is removed from the opposite set, i.e., D if the selected action is "allow", A if the selected action is "deny" (lines 11-22). This human-assisted operation guarantees that all contradictions are solved as the user actually requires. The Boolean array *altered* is accordingly updated, depending on the original policy rules from each each atomic rule derives r^α_j. For this purpose, the previously computed o array is used to properly set the values of the *altered* array.

The worst-case time complexity of Algorithm 2 can be estimated as the sum of the time complexities of three sequential code blocks. Lines 2-3 have $O(|R|)$ complexity, because $O(1)$ operations are performed on each one of the $|R|$ input rules. Lines 4-10 have $O(|R^\alpha|)$ complexity, because $O(1)$ operations are performed on each one of the $|R^\alpha|$ atomic rules. Lines 11-22 have $O(|A \cap D| \cdot |R^\alpha|)$ complexity, because it is a nested loop, where the external iterates on the elements in common to the A and D sets, the internal on the $|R^\alpha|$ atomic rules. Overall, the computational complexity is dominated by this last term, i.e., $O(|A \cap D| \cdot |R^\alpha|)$, as the number of atomic rules is not smaller than the number of original rules.

Deny-win and allow-win strategies: Algorithm 3 jointly formalizes the deny-win and allow-win resolution strategies in a parametric way, as their approaches are similar. In particular, a^W is a parameter named winning action, that is "allow" or "deny" depending on the selected strategy. The algorithm iterates twice on the atomized firewall policy. In the first iteration, it considers only the atomic rules with the same action as a^W, and it includes its condition $r^\alpha_j.c$ in the appropriate set, i.e., A if a^W = allow, D if a^W = deny. If $r^\alpha_j.c$ is already included in that set, then $altered[o[j]]$ is set to true, to specify

Algorithm 4 for anomaly resolution (priority-win)

Input: the rule list R^{α}, the array o
Output: the sets A and D, the array *altered*

```
 1: $A \leftarrow \emptyset$, $D \leftarrow \emptyset$
 2: for $i = 1, 2, ..., |R| - 1$ do
 3:     $altered[i] \leftarrow$ false
 4: for $j = 1, 2, ..., |R^{\alpha}|$ do
 5:     if $r^{\alpha}_{j}.a =$ allow $\wedge\ r^{\alpha}_{j}.c \notin (A \cup D)$ then
 6:         $A \leftarrow A \cup \{r^{\alpha}_{j}.c\}$
 7:     else if $r^{\alpha}_{j}.a =$ deny $\wedge\ r^{\alpha}_{j}.c \notin (A \cup D)$ then
 8:         $D \leftarrow D \cup \{r^{\alpha}_{j}.c\}$
 9:     else
10:         $altered[o[j]] \leftarrow$ true
11: return $A$, $D$, $altered$
```

that the duplication has been successfully removed (lines 1-11). Instead, in the second iteration, the algorithm analyzes the atomic rules with the opposite action with respect to a^{W}. Their condition $r^{\alpha}_{j}.c$ is included in corresponding set (D if a^{W} = allow, A if a^{W} = deny) only if it is not already present in the other set. In that case, it is not included so as to remove the contradiction, and $altered[o[j]]$ is set to true (lines 12-19). Differently from Algorithm 2, each contradiction is solved automatically, because the decision about the action to be enforced on the packet classes related to that policy anomaly is derived from the guideline selected by the user. The worst-case time complexity of Algorithm 3 can be estimated as the sum of the time complexities of three sequential code blocks. Lines 2-3 have $O(|R|)$ complexity, because $O(1)$ operations are performed on each one of the $|R|$ input rules. Lines 4-11 and also lines 12-19 have $O(|R^{\alpha}|)$ complexity each, as they iterate $O(1)$ operations on the $|R^{\alpha}|$ atomic rules. As these two terms are equal and dominate the first one, the overall time complexity is $O(|R^{\alpha}|)$.

Priority-win strategy: Algorithm 4 formalizes the priority-win strategy. Here, for each atomic rule $r^{\alpha}_{j} \in R^{\alpha}$, its condition $r^{\alpha}_{j}.c$ is included into the A or D set, depending on the rule action, if it is not already part of one of the two sets (lines 1-10). Indeed, if $r^{\alpha}_{j}.c$ such that $r^{\alpha}_{j}.a$ = allow already belongs to the A set, a duplication is thus removed. Instead, if it already belongs to the D set, that means that the packets are managed by a rule with higher priority and "deny" action, and a contradiction is automatically removed. Similar reasoning applies to a predicate $r^{\alpha}_{j}.c$ such that $r^{\alpha}_{j}.a$ = deny. For all the cases in which $r^{\alpha}_{j}.c$ is not included in the A and D sets in this algorithm, $altered[o[j]]$ is set to true, similarly as for the other strategies. The worst-case time complexity of Algorithm 4 can be estimated as the sum of the time complexities of two sequential blocks: the loop on the R set at lines 2-3 with $O(|R|)$ complexity and the loop on the R^{α} set at lines 4-10 with $O(|R^{\alpha}|)$ complexity. As already explained, the second term is dominant and coincides with the algorithm time complexity.

Algorithm 5 for writing the anomaly-free policy

Input: the rule list R, the sets A and D, the arrays o and *altered*
Output: the rule list R^{ϕ}

```
 1: $R^{\phi} \leftarrow []$, $C^{T} \leftarrow \emptyset$
 2: if $r_{|R|}.a =$ allow then
 3:     $C^{T} \leftarrow D$
 4: else if $r_{|R|}.a =$ deny then
 5:     $C^{T} \leftarrow A$
 6: for $i = 1, 2, ..., |R| - 1$ do
 7:     if $r_{i}.a \neq r_{|R|}.a \wedge altered[i] =$ false then
 8:         $R^{\phi} \leftarrow R^{\phi} + [r_{i}]$
 9:         for each $c_{k} \in C^{T}$ do
10:             if $c_{k} \wedge r_{i}.c$ then
11:                 $C^{T} \leftarrow C^{T} \backslash \{c_{k}\}$
12: for each $c_{k} \in C^{T}$ do
13:     for each $q_{kh} | c_{k} = \bigvee_{h=1}^{H_{k}} q_{kh}$ do
14:         $R^{\phi} \leftarrow R^{\phi} + [(\{\text{allow}, \text{deny}\} \backslash \{r_{|R|}.a\}, q_{kh})]$
15: $R^{\phi} \leftarrow R^{\phi} + [r_{|R|}]$
16: return $R^{\phi}$
```

D. Firewall anomaly-free policy rewriting

The last step of the algorithm consists in rewriting an anomaly-free policy version, by using the information computed by the resolution strategy. Algorithm 5 formalizes the algorithm that produces the anomaly-free policy R^{F}.

First, the default action of the initial policy R is kept in the anomaly-free policy R^{ϕ}. Given this assumption, the definition of the other filtering rules requires analyzing just one of the two sets A and D, i.e., the set containing the atomic predicates related to packet classes on which the opposite action of the default one must be enforced (lines 2-5). The selected set (i.e., D if the default action is allow, A if the default action is deny), used in the next steps of the algorithm, is denoted as C^{T}.

Second, the algorithm checks if some filtering rules of R can be directly reintroduced in R_{ϕ}. This is feasible for rule $r_{i} \in R$ if two conditions are satisfied: i) the rule action is the opposite of the default action, and ii) the previous algorithm for anomaly resolution establishes that, even after removing duplications and contradictions among atomic rules, the action of the original rule must be still applied to the packet classes represented by its condition, i.e., $altered[i]$ = false. If these conditions hold, r_{i} is directly included in R^{ϕ}, and all atomic predicates that intersect with $r_{i}.c$ are removed from C^{T}, because they do not have to be processed in the final step (lines 6-11).

Third, for each predicate q_{kh} composing each remaining atomic predicate $c_{k} \in C^{T}$, a rule having that predicate as rule condition and the opposite of the default action as rule action is created and included in R^{ϕ} (lines 12-14). Finally, the rule enforcing the default action is actually included at the end of the produced rule list (line 15). In the computation of this policy, there is no need to enforce a specific priority among the rules. The reason is that the configuration is anomaly-free and, for any pair of policy rules, there is no intersection among the packet space represented by their conditions, thanks to the resolution algorithm based on the atomic predicate computation. The anomaly-free policy R^{ϕ} is thus the final output of the resolution strategy, as all anomalies have been successfully removed, and it can be installed on real packet filter implementations.

The worst-case time complexity of Algorithm 5 can be estimated as the sum of the time complexities of two nested loops. Lines 6-11 have $O(|R| \cdot |C^T|)$ complexity, because the predicates in C^T are iterated for each firewall rule. Instead, lines 12-14 has $O(|C^T| \cdot \max_{k \in K} H_k)$ complexity, because for each predicate in C^T some $O(1)$ operations are executed on each sub-predicate composing it, and in the worst-case the one with the higher number of sub-predicates must be considered.

V. Implementation and Validation

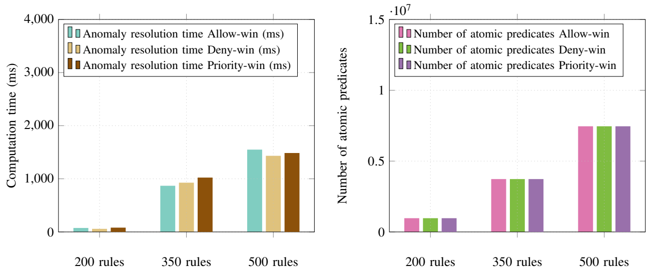
The proposed approach for firewall policy anomaly analysis and resolution has been implemented as a Java framework. The firewall policies to be analyzed can be specified by the user in XML or JSON format. The same format is used for the representation of the output, i.e., the rewritten anomaly-free policy. The framework exposes a set of REST APIs, so that it can interact with the user or with other applications. A set of translators have been also developed to translate the XML or JSON representation into the concrete configuration of real-world firewall implementations. The firewall types that the tool already supports are the most commonly used in modern networks: iptables, ipfirewalls, eBPF-based firewalls, OpenVSwitch. However, it is possible to create translators for any other kind of packet filter, with minor changes to the ones that have been already developed. The code of the tool is publicly available in the GitHub repository at the following link: https://github.com/netgroup-polito/firewall-anomalies.

The remainder of this section is structured as follows. On the one hand, it discusses the experimental setup that has been created to validate the framework (Subsection V-A), and then it presents the results of the performance validation of the developed tool (Subsection V-B). On the other hand, the framework is compared with relevant proposals of the related literature [6], [12], [28] (Subsection V-C).

A. Experimental Setup for Performance Validation

The experimental setup used to carry out all tests related to the framework validation consists in a machine with an Intel i7-6700 CPU running at 3.40 GHz and 32GB of RAM. In particular, this setup was used to validate the performance of our framework, by computing its execution time. This evaluation metric has also been adopted by other state-of-the-art studies such as [6], [12], [28] to validate their proposals related to firewall anomaly analysis and resolution. A motivation is that fast times are nowadays required by modern networks, such as the virtualized ones. Any issue in a firewall configuration must be solved as fast as possible, so as to stop and prevent possible cyberattacks, or to avoid service disruptions. Moreover, as the firewall configuration updates have becoming progressively more frequent, it is essential to have anomaly resolution methods whose performance is in line with those update times.

In this experimental setup, the performance of the Java framework has been evaluated by applying it to solve the anomalies in firewall policies of different sizes and characteristics. These policies are synthesized by a policy generator, that we have written in Java and whose code can be found in the same Github repository linked above. This generator produces different firewall policies according to the values assigned to a set of different configurable parameters: (i) the number of filtering rules composing the firewall policy, (ii) the number of anomalies (i.e., duplications and contradictions) afflicting the firewall policy rules, and (iii) the percentage of firewall policy rules that impose specific conditions also on source and destination ports, in addition to IP addresses.



(a) Computation time chart (b) Atomic predicates chart

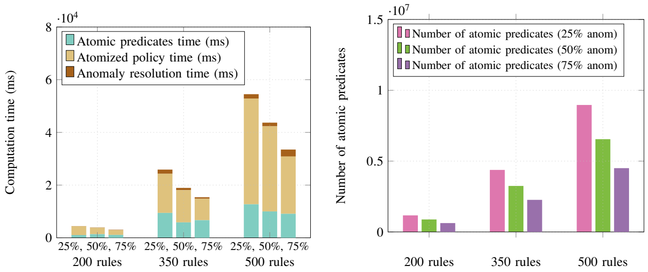
Fig. 3: Scalability versus resolution strategy

After each configurable parameter is assigned a value, the generator works as follows. First, it creates an initial subset of random firewall rules, while checking that no anomalies occur. Next, the remaining rules of the whole policy, whose size is specified as parameter (i), are created with an internal logic of the generator, so that each one has an anomaly with a rule of the initial subset. This generation continues until the number of anomalies, expressed as parameter (ii), is reached. At the same time, the generator enforces the conditions heterogeneity of the generated rules by ensuring that the percentage of firewall policy rules that impose specific conditions also on source and destination ports (i.e., those conditions fields are not $*$) is equal to the specified parameter (iii). Furthermore, concerning the anomalies that afflict each analyzed policy, by default the firewall policy generator equally distributes them among the four categories considered by Al-Shaer et al. in [6] (shadowing, correlation, generalization, redundancy). This helps us cover the full range of possible anomalies that, according to the literature, may occur in a firewall configuration.

B. Performance Validation

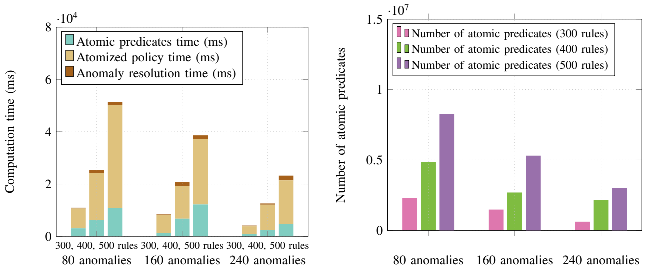
The strategy employed for text execution was the following. First, we investigated how the choice of the resolution strategy affects the results, checking if there is one that performs better or if the three strategies have comparable time. Then, we proceeded to analyze how the numbers of rules and anomalies influence the computation times (varying parameters (i) and (ii)). In doing so, for each analyzed test case, we present two charts, one showing the computational time and the other representing the number of generated atomic predicates for each considered firewall policy. The aim was to check if there exists a link between the resolution time and the number of generated atomic predicates. Additionally, we investigated how the heterogeneity of rule conditions may impact the performance of the framework (varying parameter (iii)).

In the following, we discuss the results achieved from test execution.



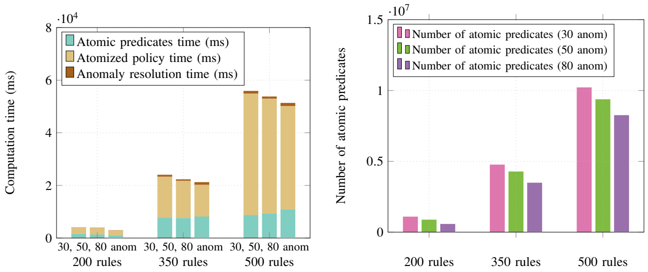
(a) Computation time chart (b) Atomic predicates chart

Fig. 4: Scalability versus number of rules maintaining constant the percentage of rules with anomalies



(a) Computation time chart (b) Atomic predicates chart

Fig. 6: Scalability versus number of anomalies



(a) Computation time chart (b) Atomic predicates chart

Fig. 5: Scalability versus number of rules maintaining constant the number of anomalies

1) Impact of the chosen anomaly resolution strategy: As anticipated, we investigated how the execution time changes with respect to the chosen automatic resolution strategy, i.e., deny-win, allow-win and priority-win. For these tests, we consider only the time related to Algorithms 3 and 4 proposed in our model (i.e., the time required to solve the anomalies once policy rules are atomized). We did not report the times of the previous operations of the workflow, because they are in common to all strategies. However, we will report and analyze them in the context of the next tests.

Fig. 3 shows the results obtained analyzing firewalls with 200, 300 and 500 rules. In this case, the percentage of rules with anomalies is always constant and set at 40% of the total number of rules. As it can be seen from Fig. 3a, the resolution time remains fairly constant regardless of the chosen strategy and increases with the number of rules. Therefore, there is not an automatic resolution strategy that performs better than the others. The number of generated atomic predicates, instead, is exactly the same in each test case and increases with the number of rules as well (Fig. 3b). This was expected, since the atomic predicate computation phase is a preliminary step common to all the strategies (and thus independent of the selected one).

Since the resolution strategy choice does not influence in a significant way the computation time results, for simplicity we decided to use only one resolution strategy, the deny-win one, for all next tests.

2) Impact of increasing numbers of policy rules and anomalies: In a second series of tests, we investigated how the total number of rules and anomalies specifically impact the computation times. To this purpose, we considered three different test case scenarios:

- in the first test case scenario, the numbers of rules and anomalies increase simultaneously, and there is a fixed ratio between them (Fig. 4);
- in the second test case scenario, only the number of rules progressively increases, while the number of anomalies is kept constant (Fig. 5);
- in the third test case scenario, only the number of anomalies progressively increases, while the number of rules is kept constant (Fig. 6).

Together, these three scenarios cover a large number of cases, with an exhaustive comparison of the possible impact the two parameters may have.

Again, for each test, we present two charts. The first one refers to the computation time, that is the time required by the tool to analyze the firewall policy and solve the anomalies. Each bar of this chart is composed of three components with different colors, representative of the time to execute specific sub-algorithms of our methodology: i) time required to compute all the atomic predicates (lines 1-4 of Algorithm 1); ii) time required to create the atomized policy by using the previously computed atomic predicates (lines 5-12 of Algorithm 1); iii) time required for anomaly resolution (Algorithm 3 or 4, depending on the selected strategy) and for writing the final anomaly-free policy (Algorithm 5). These two times were joined, because the latter is negligible as it is time to iterate once over all input rules and once over all atomic predicates in the worst case. Therefore, it would not be visible if represented alone in the bar. The second chart, instead, shows the number of generated atomic predicates.

For what concerns the impact of the rule number, we can state that an increasing number of rules leads to a longer execution time. This is true in both the first and second test case scenarios, i.e., when the number of anomalies increases with the number of rules (Fig. 4a) and when it remains constant (Fig. 5a). In particular, all three times grow. An increasing number of rules requires longer time to compute the set of atomic predicates representative for all rules, and it also leads to a higher number of predicates (Fig. 4b and 5b). A larger set of predicates then increases the time spent to atomize the policy rules, as there are more rules to atomize and a larger set of atomic predicates to consider. The anomaly resolution time increases as well, but it has a less significant impact. The

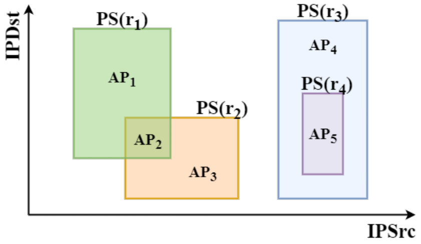
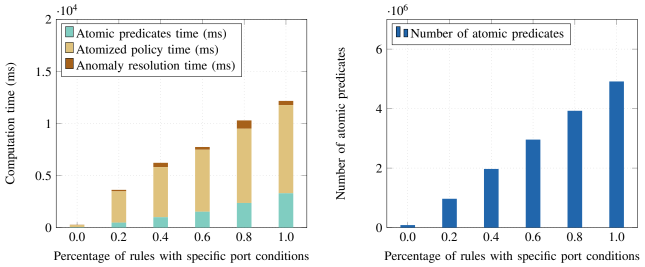
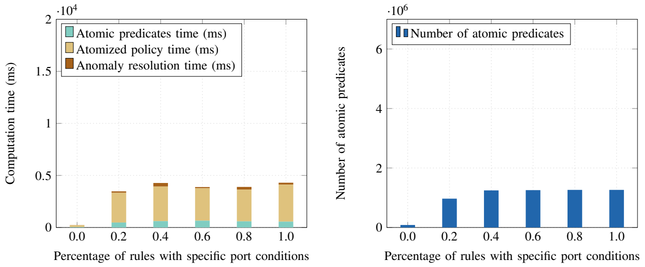


Fig. 7: Atomic Predicates generated by an intersection (AP1, AP2, AP3) and by an inclusion (AP4, AP5)



(a) Computation time chart (b) Atomic predicates chart

Fig. 8: Scalability versus rule conditions heterogeneity (uncommon case)



(a) Computation time chart (b) Atomic predicates chart

Fig. 9: Scalability versus rule conditions heterogeneity (common case)

major contribution is given by the time spent to atomize the policy rules. The reason is that the tool has to iterate, for each original rule, over the entire set of atomic predicates. However, this process can be parallelized, potentially one thread to atomize each original rule. In our tests, we run the tool using 8 threads. Increasing the number of threads, performance would be better. Besides, we would like to remark that, even in the worst case that has been analyzed here, the framework can successfully solve all anomalies which are present among 500 rules in less than 60 seconds.

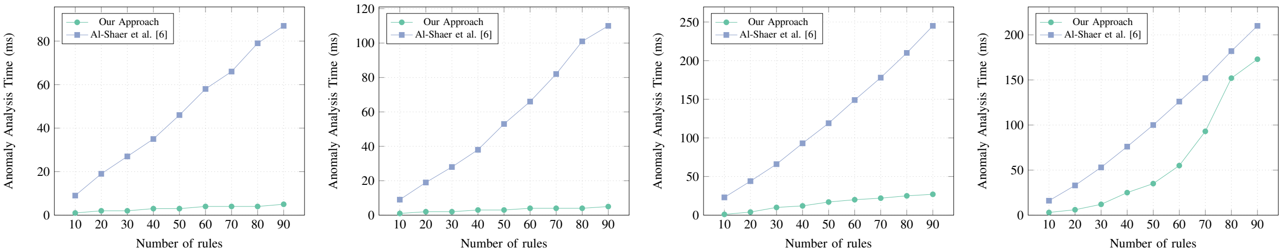
For what concerns the impact of the anomaly number, as we can see in Fig. 4a, 5a and more specifically in Fig. 6a, we can state that an increasing number of anomalies among firewall rules decreases the total resolution time. This apparently paradoxical result is actually reasonable and expected, since a greater number of anomalies reduces the number of generated atomic predicates (Figs. 4b, 5b, 6b). This is particularly true when the packet space of a rule condition is included in the space of another rule condition. Fig. 7 exemplifies this concept. In this example, the two overlapping predicates presenting the packet spaces of rules r_1 and r_2 (i.e., PS(r_1) and PS(r_2)) generate three atomic predicates, while predicates that are one the subset of the other (such as PS(r_3) and PS(r_4) generate only two atomic predicates. This two-dimensional exemplification gives the idea that the fewer the anomalies, the higher the number of generated atomic predicates. As we have seen in all the previous charts, there is a direct positive correlation between the number of generated predicates and the increase of the times, i.e., the fewer the anomalies, the higher the resolution times. This result is interesting, as our approach can be efficiently solve quite complex problems where the number of anomalies is very high.

3) Impact of rule conditions heterogeneity: The heterogeneity of rule conditions represents a complexity factor for the algorithms defined for atomic predicates computation and atomized policy creation. In particular, the more heterogeneous the values selected by the condition fields are among the rules of the same policy, the more atomic predicates are generated. This relationship is explained by the fact that different rule conditions represent different packet sub-spaces of the 5-dimensional hyperspace, and therefore different atomic predicates are associated to those sub-spaces and their intersections.

In order to assess this potential impact onto the performance, we have executed our framework on policies with peculiar characteristics. Each rule of those policies imposes specific different values for the condition fields related to the source and destination IP addresses, so as to make the analyzed policies sufficiently complex. Then, the policies have a progressively increasing percentage (from 0% to 100%) of rules that impose specific conditions on port fields, and consequently a decreasing percentage (from 100% to 0%) of rules that set the port fields to the wildcard value $*$. Policy that have more rules with specific conditions on port fields are thus more heterogeneous, and more complex to analyze, than the other ones.

First, we have considered an uncommon case, where port numbers are randomly selected among all possible 65535 ones (Fig. 8). The execution time and the atomic predicates number increase progressively with the percentage of rules that impose specific conditions on port fields, as it can be seen in Figs. 8a and 8b. Specifically, the largest increase is given by the time to compute the atomic predicates and the time to atomize the original rules, since the algorithm has to iterate over a larger set of predicates. This is an uncommon case, because it is quite rare that firewall policies have rules so heterogeneous, as the commonly used port numbers belong to a restricted set. Anyhow, this test showed that, despite the impact that rule conditions heterogeneity has onto the performance, our framework can still analyze and solve all anomalies in a little more than 10 seconds even for the worst case considered.

Then, we have considered a more common case, where port numbers are selected from a restricted set of 30 "well-known" ports, e.g., 80, 443, 22 (Fig. 9). Even if more atomic predicates are generated as the percentage of rules with specific conditions on port fields increases, their increase is limited



(a) Test Case A

(b) Test Case B

(c) Test Case C

(d) Test Case D

Fig. 10: Comparison with the approach by Al-Shaer et al. [6]

since only a limited set of possible port numbers is employed for policy creation. While in the previous example, Fig. 8, ports were randomly selected from the set of all possible port numbers (i.e, 0 to 65535) thus generating a higher combination of atomic predicates. Fig. 9b shows that, in the first two test cases, the number of generated atomic predicates increases. Starting from the third case, instead, it remains constant. The number of atomic predicates, in this case, saturates at the maximum possible value, that is the one that has a specific predicate for each port in the set of "well-known" ports. This helps to reduce the resolution times (see Fig. 9a) and is also reasonable in real case scenarios, where the firewall rules hardly have port numbers different from well-known values.

C. Comparison with related approaches

The proposed methodology has been compared with three proposals of the related literature: [6] [12] [28].

1) Comparison with Al-Shaer et al.: The study by Al-Shaer et al. [6] was selected as a baseline among the works that only address anomaly analysis (i.e., the ones discussed in Subsection II-A) because its anomaly categorization is mutated by all other studies of that class, which were excluded because they did not provide an equally impactful contribution, and the description of their performance validation is more limited.

Fig. 10 shows a comparison of our anomaly analysis algorithm with the one proposed by Al-Shaer et al. in [6]. All the times reported in this figure are related to the computation time required for anomaly analysis because the approach by Al-Shaer et al. cannot resolve the identified anomalies. For a fair comparison, our anomaly analysis algorithm has been executed on the same four test cases which were used by Al-Shaer et al. to validate their technique:

- Test Case A is based on firewall policies whose rules are different in the destination address only.
- Test Case B is based on firewall policies whose rules are different in the source address only.
- Test Case C is based on firewall policies where each rule is a superset match of the preceding rule, i.e., the packet space of the rule condition includes the packet space of the preceding rule condition.
- Test Case D is based on firewall policies where the rules are randomly selected from the three previous test cases.

For each test case, nine different firewall policies have been used, with progressive number of rules from 10 to 90. Even though our approach can analyse bigger policies as shown in Subsection V-B, the paper by Al-Shaer et al. did not report execution times related to bigger firewall configurations.

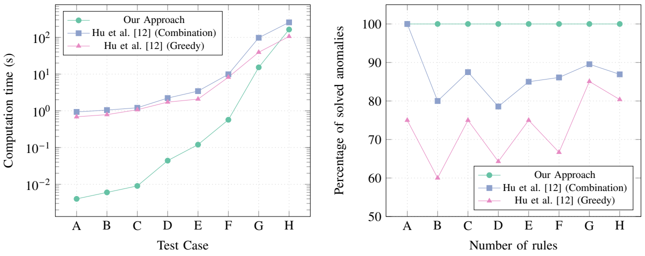
Test Cases A and B represent the best case scenario for the approach by Al-Shaer et al., because they require the minimum time to navigate the policy rule tree that they build to search for possible anomalies. However, they are also the best cases for our anomaly analysis strategy. In fact, the number of atomic predicates that is generated is very low, as it is directly equal to the number of policy rules. Therefore, our anomaly analysis times are always lower by one or two magnitude orders than the ones by Al-Shaer et al., and our technique is extremely efficient, because it identifies all anomalies in few milliseconds.

Test Case C represents the worst case scenario for the approach by Al-Shaer et al., because each rule requires a complete navigation of the policy tree in order to analyze the entire rule set. Our approach can deal efficiently also with this peculiar test case. As each rule condition includes the condition of the previous one, the rules have several common intersections, which can be expressed with a single atomic predicate for each one. This guarantee that, in less then 30 ms, our approach can successfully identify all anomalies for a firewall configuration whose analysis required 245 ms for the strategy by Al-Shaer et al..

Test Case D represents the average case scenario for the approach by Al-Shaer et al.. Instead, it is the worst case among these four for our approach. Indeed, the number of atomic predicates is higher than the ones in the previous scenarios, because multiple different intersections among rule condition may occur due to the randomness of rule selection. However, for all nine firewall policies, including the one with 90 rules, our approach still proved to be always more efficient.

In summary, this comparison shows that our anomaly analysis time is always lower than the one by Al-Shaer et al. for all the test cases which they described in [6]. Besides, we would like to remark that their approach cannot solve the identified anomalies, differently from ours.

2) Comparison with Hu et al.: The study by Hu et al. [12] was selected as a comparison reference in the class of studies about traditional firewall anomaly resolution (i.e., the ones discussed in Subsection II-B) because it is widely acknowledged in the firewall policy management research community, and it addresses the firewall problem broadly. Instead, the other ones were excluded because published earlier, less relevant, or in



(a) Computation time (b) Resolution efficiency

Fig. 11: Comparison with the approach by Hu et al. [12]

TABLE V: *Test case characteristics for the comparison with the approach by Hu et al. [12]*

	Our Approach		Approach of Hu et al.	
# Test Case	#Rules	#Anomalies	#Rules	#Anomalies
A	12	4	12	4
B	18	6	18	5
C	27	9	25	8
D	54	18	52	14
E	81	27	83	20
F	132	44	132	36
G	354	118	354	67
H	927	309	926	107

the case of [22] too domain-specific.

Fig. 11 shows a comparison of our approach with this proposal by Hu et al. [12]. The evaluation of the two approaches was carried out in eight use cases, described in their paper. Those eight scenarios are based on firewall policies, with increasing numbers of policy rules and anomalies. These characteristics allow assessing the differences between the two strategies as the policy complexity varies.

The two validation parameters used for the comparison were the computation time and the anomaly resolution efficiency, expressed as the percentage of solved anomalies. Even if it was not possible to execute their code as it is not publicly available, we could compute the total execution times by summing the segmentation, correlation, and resolution times that the authors reported in their paper for those eight use cases. The characteristics of those eight use cases, in terms of the number of rules and anomalies, are reported in the right part of TABLE V. We decided not to report the results of their third proposed strategy, named permutation algorithm, because it has much worse performance than the other two ones, and it is not able to compute a solution for complex instances of the anomaly resolution problem in finite time, as declared by the authors themselves.

For this comparison, our proposed algorithm was applied to eight use cases that are the most similar as possible to the ones described by Hu et al. in their paper, and are described in the left part of TABLE V. Specifically, we used eight firewall policies whose numbers of rules and anomalies are in the same magnitude order as those described in [12], and at the same time we kept a constant ratio of 3:1 between the number of rules and the number of anomalies. For what concerns the other parameters that are relevant to the performance of our algorithm, we decided to keep the values for the percentages of firewall policy rules that impose specific conditions on ports and transport-level protocols the same as the one defined for most of our previous tests, i.e., 20% and 50%. Then, we use the deny-win strategy for anomaly resolution, as we already proved that all proposed automatic resolution strategies have anyway the same performance.

A significant conclusion that can be drawn by Fig. 11b about anomaly resolution efficiency is that our approach can actually solve all anomalies afflicting any firewall policy. This achievement is feasible because we reformulate the original firewall policy into an anomaly-free version, rewriting the original rules when needed by means of the computed atomic predicates. Instead, both the combination and greedy resolution strategies proposed by Hu et al. cannot solve all anomalies. In the worst case where those strategies were tested, the percentages of solved anomalies could go down to 78% and 60%, respectively. This difference is motivated by the fact that their resolution strategies consists in rule reordering algorithms. However, as experimentally shown by the authors with the values reported in this table, there may not exist a rule reordering that allows to solve all anomalies.

Another relevant consideration, which can be drawn from Fig. 11a (whose y axis is in logarithmic scale) about computation time comparison, is that the performance of our approach is significantly better than both the strategies by Hu et al. in most of the analyzed test cases. For the simpler six test cases, the computation time of our approach is better by even a magnitude order, as it is inferior to one second. For the worst analyzed case, it is still better than the combination resolution algorithm by Hu et al., while it is slightly inferior to the computation time taken by their greedy algorithm. Nevertheless, their greedy algorithm can solve a reduced number of anomalies. Indeed, the authors themselves considers the combination algorithm their most valuable algorithm, as it represents higher efficiency and effectiveness in conflict resolution. With respect to that, our approach is better with respect to both time performance and anomaly resolution.

Additionally, if we sum the execution times of the frameworks described in [6] and [12] and we compare them with the total execution time of ours, our framework would be able to outperform their straightforward combination.

In conclusion, these comparisons further show the benefits, previously discussed in Section II, that our proposal can bring over to the state of the art.

3) Comparison with Asif et al.: The study by Asif et al. was selected as a baseline among the studies about SDN switch anomalies because, as explained in Subsection II-B, all the others are not as feature complete as this study.

Fig. 12 shows a comparison of our approach with the proposal by Asif et al. [28]. For this comparative experiment, we used three real firewall policies that used to be installed in the backbone of the Stanford networks, and whose configuration has been made publicly available by past Stanford researchers publicly available in the GitHub repository at the following link: https://github.com/eastzone/atpg/tree/master. Specifically, we reused the same three firewall policies on which Asif et al. validate their proposals according to their paper. These policies are named Stanford1, Stanford2 and Stanford3, and are respectively composed of 203, 43 and 53 rules.

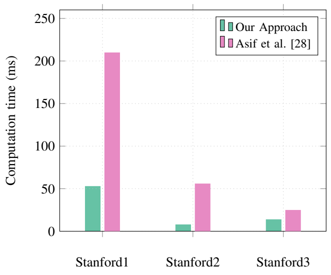


Fig. 12: Comparison with the approach by Asif et al. [28]

The experimental results reported in Fig. 12 are directly derived from applying our framework to the three policies on the previously described experimental setup for our approach, and are taken from the paper [28] for Asif et al.'s approach. In all three cases, our solution achieves a complete resolution of all the existing anomalies in less time than the other one. Besides, for the most complex policy, i.e., Stanford1, the execution time of our framework is an order of magnitude smaller. This comparison has a twofold value. On the one hand, it shows that our proposed approach has performance that is also in line with the times requested to address the anomaly management problem in SDN-based virtual networks, for which the compared solution of Asif et al. was designed. On the other hand, this comparative evaluation allowed us to successfully apply our approach to real firewall policies, thus proving its feasibility in real-world networks.

VI. Limitations

In the following, some limitations of the proposed approach are highlighted.

First, parallelization has been used to improve the performance of the designed algorithms, such as Algorithm 1. However, the full benefits of parallel computing can be exploited by using hardware tailored to parallelization, e.g., multi-core processors. For the performance validation of the developed framework, we used the Intel i7-6700, a 4-core processor that supports up to eight parallel threads thanks to the hyper-threading technology. Indeed, we were able to achieve the best performance in that experimental setup when using eight threads. If a process with more cores were used, then better performance would also be achieved. This may represent a limitation, as it requires the users to have hardware suitable to execute parallel code. At the same time, nowadays most commercial processors, including the ones used in personal computers, have at least four cores. Therefore, this hardware requirement is not very restrictive.

Second, the proposed approach has been validated to solve the anomaly analysis and resolution problems for packet filtering firewall policies, whose rules define conditions over the fields of the IP 5-tuple. An extension to other filtering controls, such as application-layer firewalls and SDN switches, may be theoretically feasible, but it may impact the performance of the approach. The reason is that the packet space matching the conditions of an application-layer firewall or SDN switch rule would be an N-dimensional hypercube, with N greater than five. The parallelization strategies that have been already implemented would decrease the impact, but they may not be enough. Therefore, new strategies should be investigated.

VII. Conclusions

This paper proposes a novel approach for firewall policy anomaly analysis and resolution. This approach is based on modeling rule conditions as potentially complex predicates, and splitting them into simpler ones, named atomic predicates. Their computation allows to identify intersections among the packet spaces represented by rule conditions, and to create an alternative representation of the firewall policy, where each rule has an atomic predicate as condition. For these atomic rules, the anomaly analysis is much easier and more intuitive with respect to other approaches. To solve the detected anomalies, we propose multiple resolution strategies, including human-assisted or automatic ones. All these strategies allow to solve all anomalies in fast times. This achievement has been experimentally proved with tests assessing the performance of the framework implementing our approach, with respect to multiple parameters and to related proposals of the literature.

As future work, we plan to integrate our approach into an existing open-source framework for firewall configuration, Batfish, so as to have a comprehensive tool for the management of firewall policies, inclusive of a consistency check of the produced anomaly-free policy. We also plan to research how the rules of the final anomaly-free policy output by our approach may be further aggregated when needed, without introducing new anomalies. The objective of this study would be to improve the performance of firewall implementations that still use linear search algorithms when deciding which rule matches a received packet. Besides, we will investigate if this approach based on atomic predicates can be used to solve other problems related to policy-based management, such as security policy-based verification and configuration.

References

[1] P. P. Mukkamala and S. Rajendran, "A survey on the different firewall technologies," *Inter. J. of Engin. Appl. Scien. and Tech.*, vol. 5 (1), 2020.

[2] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "Automation for network security configuration: State of the art and research trends," *ACM Comput. Surv.*, vol. 56, no. 3, pp. 57:1–57:37, 2024.

[3] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Automated firewall configuration in virtual networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 22, no. 2, 2023.

[4] A. A. Jabal, M. Davari, E. Bertino, C. Makaya, S. Calo, D. Verma, A. Russo, and C. Williams, "Methods and tools for policy analysis," *ACM Comput. Surv.*, vol. 51, no. 6, 2019.

[5] E. S. Al-Shaer and H. H. Hamed, "Firewall policy advisor for anomaly discovery and rule editing," in *Proc. of the IEEE Eighth International Symposium on Integrated Network Management.*, 2003, pp. 17–30.

[6] E. Al-Shaer, H. H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," *IEEE J. on Sel. Areas in Commun.*, vol. 23, no. 10, 2005.

[7] L. Yuan, H. Chen, J. Mai, and C.-n. Chuah, "FIREMAN: A Toolkit for FIREwall Modeling and ANalysis," in *Proc. of the IEEE Symposium on Security and Privacy*, 2006, pp. 199–213.

[8] K. Golnabi, R. K. Min, L. Khan, and E. Al-Shaer, "Analysis of firewall policy rules using data mining techniques," in *Proc. of the 10th IEEE/IFIP Network Operations and Management Symposium*, 2006, pp. 305–315.

[9] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège, "A formal approach to specify and deploy a network security policy," in *Proc. of the 2nd IFIP TC1 WG1.7 Workshop on Formal Aspects in Security and Trust*, 2004.

[10] F. Valenza and M. Cheminod, “An optimized firewall anomaly resolution,” *J. Internet Serv. Inf. Secur.*, vol. 10, no. 1, pp. 22–37, 2020.

[11] H. Hu, G.-J. Ahn, and K. Kulkarni, “FAME: a firewall anomaly management environment,” in *Proc. of the 3rd ACM workshop on Assurable and usable security configuration (SafeConfig10)*, October 2010, pp. 17–26.

[12] H. Hu, G. J. Ahn, and K. Kulkarni, “Detecting and resolving firewall policy anomalies,” *IEEE Trans. Dependable Sec. Comput.*, vol. 9, no. 3, pp. 318–331, 2012.

[13] H. Yang and S. S. Lam, “Real-time verification of network properties using atomic predicates,” *IEEE/ACM Trans. on Net.*, vol. 24, no. 2, 2016.

[14] ——, “Scalable verification of networks with packet transformers using atomic predicates,” *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2900–2915, 2017.

[15] T. Abbes, A. Bouhoula, and M. Rusinowitch, “An inference system for detecting firewall filtering rules anomalies,” in *Proc. of the ACM symposium on Applied computing*, 2008, pp. 2122–2128.

[16] A. X. Liu and M. G. Gouda, “Complete Redundancy Detection in Firewalls,” in *Proc. of the 19th Working Conference on Data and Applications Security (IFIP WG 11.3)*. Springer, 2005, pp. 193–206.

[17] A. Jeffrey and T. Samak, “Model Checking Firewall Policy Configurations,” in *Proc. of the IEEE International Symposium on Policies for Distributed Systems and Networks*, 2009, pp. 60–67.

[18] F. Cuppens, N. Cuppens-Boulahia, and J. Garcia-Alfaro, “Detection and Removal of Firewall Misconfiguration,” in *Proc. of the 2005 IASTED International Conference on Communication, Network and Information Security*, 2005, pp. 154–162.

[19] ——, “Detection of network security component misconfiguration by rewriting and correlation,” in *Proc. of the Conf. on Security in network Architectures and Security of Information Systems*, 2006.

[20] J. García-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and S. Preda, “MIRAGE: A management tool for the analysis and deployment of network security policies,” in *Proc. of the 5th Intern. Workshop, Data Privacy Management and Autonomous Spontaneous Security*, 2010.

[21] S. Ferraresi, S. Pesic, L. Trazza, and A. Baiocchi, “Automatic conflict analysis and resolution of traffic filtering policy for firewall and security gateway,” in *Proc. of the IEEE International Conference on Communications*, 2007, pp. 1304–1310.

[22] G. Li, H. Zhou, B. Feng, G. Li, H. Zhang, and T. Hu, “Rule anomaly-free mechanism of security function chaining in 5g,” *IEEE Access*, vol. 6, pp. 13 653–13 662, 2018.

[23] G. Li, M. Dong, K. Ota, J. Wu, J. Li, and T. Ye, “Deep packet inspection based application-aware traffic control for software defined networks,” in *Proc. of 2016 IEEE GLOBECOM*, 2016, pp. 1–6.

[24] H. Hu, W. Han, G. Ahn, and Z. Zhao, “FLOWGUARD: building robust firewalls for software-defined networks,” in *Proc. of the 3rd workshop on Hot topics in software defined networking*, 2014, pp. 97–102.

[25] Q. Li, Y. Chen, P. P. C. Lee, M. Xu, and K. Ren, “Security policy violations in SDN data plane,” *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1715–1727, 2018.

[26] J. Cui, S. Zhou, H. Zhong, Y. Xu, and K. Sha, “Transaction-based flow rule conflict detection and resolution in SDN,” in *Proc. of the 27th International Conference on Computer Communication and Networks*, 2018, pp. 1–9.

[27] S. Pisharody, J. Natarajan, A. Chowdhary, A. Alshalan, and D. Huang, “Brew: A security policy analysis framework for distributed sdn-based cloud environments,” *IEEE Trans. Dependable Secur. Comput.*, vol. 16, no. 6, pp. 1011–1025, 2019.

[28] A. B. Asif, M. Imran, N. Shah, M. Afzal, and H. Khurshid, “ROCA: auto-resolving overlapping and conflicts in access control list policies for software defined networking,” *Int. J. Commun. Syst.*, vol. 34, no. 9, 2021.

[29] D. Berardi, F. Callegati, A. Melis, and M. Prandini, “Technetium: Atomic predicates and model driven development to verify security network policies,” in *Proc. of the IEEE 17th Annual Consumer Communications & Networking Conference*, 2020, pp. 1–6.

[30] Y. Zhang, J. Li, S. Kimura, W. Zhao, and S. K. Das, “Atomic predicates-based data plane properties verification in software defined networking using spark,” *IEEE J. Sel. Areas Commun.*, vol. 38, no. 7, pp. 1308–1321, 2020.

[31] P. Zhang, X. Liu, H. Yang, N. Kang, Z. Gu, and H. Li, “Apkeep: Realtime verification for real networks,” in *Proc. of the 17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020*, 2020, pp. 241–255.

[32] D. Guo, J. Luo, K. Gao, and Y. R. Yang, “Poster: Scaling data plane verification with throughput-optimized atomic predicates,” in *Proc. of the ACM SIGCOMM 2023 Conference*, 2023, pp. 1141–1143.

[33] X. Li and C. Qian, “An NFV orchestration framework for interference-free policy enforcement,” in *Proc. of the 36th IEEE International Conference on Distributed Computing Systems,*, 2016, pp. 649–658.

[34] D. Bringhenti, S. Bussa, R. Sisto, and F. Valenza, “A two-fold traffic flow model for network security management,” *IEEE Transactions on Network and Service Management*, 2024.

[35] C. Leet, R. Soulé, Y. R. Yang, and Y. Zhang, “Flow algebra: Towards an efficient, unifying framework for network management tasks,” in *Proc. of the 40th IEEE Conference on Computer Communications, INFOCOM 2021*, 2021, pp. 1–10.

[36] S. P. Morrissey and G. G. Grinstein, “Visualizing firewall configurations using created voids,” in *Proc. of the 6th International Workshop on Visualization for Cyber Security*, 2009, pp. 75–79.

[37] C. Basile, A. Cappadonia, and A. Lioy, “Network-level access control policy analysis and transformation,” *IEEE/ACM Trans. Netw.*, vol. 20, no. 4, pp. 985–998, 2012.

[38] A. Voronkov, L. H. Iwaya, L. A. Martucci, and S. Lindskog, “Systematic literature review on usability of firewall configuration,” *ACM Comput. Surv.*, vol. 50, no. 6, pp. 87:1–87:35, 2018.

[39] M. Kuzniar, P. Peresíni, and D. Kostic, “What you need to know about SDN flow tables,” in *Proc of. Passive and Active Measurement - 16th International Conference*, ser. Lecture Notes in Computer Science, vol. 8995. Springer, 2015, pp. 347–359.

[40] J. Daly, V. Bruschi, L. Linguaglossa, S. Pontarelli, D. Rossi, J. Tollet, E. Torng, and A. Yourtchenko, “Tuplemerge: Fast software packet processing for online packet classification,” *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1417–1431, 2019.

Daniele Bringhenti received the M.Sc. degree (summa cum laude) and the Ph.D. degree (summa cum laude) in computer engineering from the Politecnico di Torino, Torino, Italy, in 2019 and 2022 respectively, where he is currently a fixed-term Assistant Professor. His research interests include novel networking technologies, automatic orchestration and configuration of security functions in virtualized networks, formal verification of network security policies.

Simone Bussa received the M.Sc. degree (summa cum laude) in computer engineering from the Politecnico di Torino, Turin, Italy, in 2021, where he is currently working toward the Ph.D. degree in control and computer engineering. His research interests include distributed systems security and formal verification applied in the field of cyber-physical systems.

Riccardo Sisto received the Ph.D. degree in Computer Engineering in 1992, from Politecnico di Torino, Italy. Since 2004, he is Full Professor of Computer Engineering at Politecnico di Torino. His main research interests are in the area of formal methods, applied to distributed software and communication protocol engineering, distributed systems, and computer security. He has authored and co-authored more than 100 scientific papers. He is a Senior Member of the ACM.

Fulvio Valenza received the M.Sc. degree (summa cum laude) and the Ph.D. degree (summa cum laude) in computer engineering from the Politecnico di Torino, Torino, Italy, in 2013 and 2017, respectively, where he is currently a Tenure-Track Assistant Professor. His research activity focuses on network security policies, orchestration and management of network security functions in SDN/NFV-based networks, and threat modeling.