Autonomous attack mitigation through firewall reconfiguration

International Journal of Network Management

WILEY

SPECIAL ISSUE PAPER OPEN ACCESS

Autonomous Attack Mitigation Through Firewall Reconfiguration

Daniele Bringhenti | Francesco Pizzato | Riccardo Sisto | Fulvio Valenza

Dipartimento di Automatica e Informatica, Politecnico di Torino, Turin, Italy

Correspondence: Daniele Bringhenti (daniele.bringhenti@polito.it)

Received: 30 July 2024 | **Revised:** 9 September 2024 | **Accepted:** 24 September 2024

Funding: This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union—NextGenerationEU.

Keywords: attack mitigation | firewall configuration | security automation

ABSTRACT

Packet filtering firewalls represent a main defense line against cyber attacks that target computer networks daily. However, the traditional manual approaches for their configuration are no longer applicable to next-generation networks, which have become much more complex after the introduction of virtualization paradigms. Some automatic strategies have been investigated in the literature to change that old-fashioned configuration approach, but they are not fully autonomous and still require several human interventions. In order to overcome these limitations, this paper proposes an autonomous approach for firewall reconfiguration where all steps are automated, from the derivation of the security requirements coming from the logs of IDSs to the deployment of the automatically computed configurations. A core component of this process is React-VEREFOO, which models the firewall reconfiguration problem as a Maximum Satisfiability Modulo Theories problem, allowing the combination of full automation, formal verification, and optimization in a single technique. An implementation of this proposal has undergone experimental validation to show its effectiveness and performance.

1 | Introduction

The growing size and complexity of modern computer networks, designed around virtualization principles such as *network functions virtualization* (NFV) and *software-defined networking* (SDN), made impractical the traditional approaches for network security configuration. According to the most recent Data Breach Investigations Report by Verizon, misconfiguration was seen in approximately 10% of breaches [1]. The main reason is that old-fashioned configuration strategies were manual, so they could be applied successfully only to static networks, where every component was under the direct control of the network administrator. In order to tackle the dynamic and evolutionary nature of next-generation networks, automatic approaches for network security configuration have recently become popular as a possible solution to this management problem. Nowadays, many solutions adopt policy-based management approaches, where the administrators simply specify the security requirements that must be enforced in their network (e.g., they specify which traffic flows should be blocked before reaching their destination because potentially malicious), and then, an automated process computes the configuration of the required network security functions [2].

Among the automated approaches for security configuration proposed in the literature, many of them address this problem for packet filtering firewalls because they represent the most effective response to a large number of possible cyber attacks. The solutions investigated for firewalls are quite rich, especially the ones for distributed firewalls. Indeed, configuring them automatically in a virtual network means both establishing how firewall instances must be allocated in the

logical topology and computing their filtering rules. Besides, those proposals have been progressively enriched with other features. For example, some of those solutions embed formal methods to address the problem, ensuring solution correctness by construction. This is crucial because it allows the avoidance of possible misconfigurations, which is essential to provide adequate network security [3, 4].

Despite the different advantages achieved by these solutions for firewall configuration, such as avoiding trial-and-error human operations and improving security management through automation, most of them have two common weaknesses related to the fact that they are not explicitly designed to address the reconfiguration problem.

On the one hand, whenever there is a change in the set of security networks to be enforced as a consequence of a cyber attack, most of the approaches proposed so far need to be re-executed from scratch to synthesize a new valid configuration, resulting in a wasteful process, both in terms of computation power and time. Moreover, this may produce a significantly different configuration with respect to the original one. Consequently, in order to deploy the updated configuration, a large part of the network needs to be shut down, updated, and restarted, adding another time delay that is not negligible (e.g., OpenStack requires more than 5 s for the deployment of a single machine [5], and Open Source MANO, a well-known NFV orchestrator, requires a delay of 134 s to deploy a virtual function [6]). This also clashes with the trend for modern network attacks, as reported by various sources [7, 8]. Cloudflare reported [7] that in 2022, 96% of DDoS attacks remain below 500 Mbps, with 52% of the total attacks lasting less than 10 min. Also, the news of the attack undergone by Proton in 2022 [8] showed evidence of burst DDoS attacks, with multiple attack vectors and rapid changes within minutes.

On the other hand, these approaches proposed in literature still require a large number of interactions with human administrators, and this represents a two-fold issue in terms of time and correctness. Concerning time, these interactions significantly delay the moment in which the automatically computed configuration can actually be deployed in the physical infrastructure of the virtual networks. If the objective is to address cyber attacks as soon as possible, then it is not reached. Concerning correctness, these interactions may lead to introducing new errors in firewall configurations. For example, administrators must still analyze the logs produced by intrusion detection systems (IDSs) to get information about ongoing cyber attacks, and they must create a new set of security requirements. If some parts of the logs are misinterpreted, then the policies may not be coherent. Similarly, administrators must manually issue all the configuration commands produced by automatic approaches to make firewalls operative. Unfortunately, also this operation may be prone to decision-making errors, due to the variety of real-world firewall products.

This paper aims to overcome these existing limitations by proposing a novel approach whose objective is to automatically reconfigure a distributed firewall in virtual computer networks whenever an IDS alerts about a detected cyber attack by writing a new log entry. This approach is meant to be autonomous and continuously active to provide prompt automatic reactions whenever required. After the initial input specification, the human administrator should no longer be required to interact with our proposed process. In particular, the administrator does not have to define new security requirements to specify how the attack must be mitigated and does not have to issue the low-level configuration commands personally. This solution is thus suitable for the timing of modern attacks that can reconfigure the network quickly and adapt to evolving attack scenarios.

The proposed approach includes multiple steps dedicated to the automated operations: firewall configuration translation and deployment, IDS log monitoring and requirement extraction, merging of old requirements with the extracted ones, and firewall reconfiguration computation. The last step is the most complex one because understanding how firewall instances should be repositioned and reconfigured to satisfy new security requirements implies a high degree of versatility for decision-making. For the resolution of the problem involved in this step, the proposed approach includes an algorithmic methodology, named Reactive VErified REFinement and Optimized Orchestration (React-VEREFOO), which represents a primary component. React-VEREFOO performs an optimized security reconfiguration of distributed packet filters for an already deployed network within a short computation time. Its novelty lies in the coexistence of three important features that, to the best of our knowledge, are not supported by any other reconfiguration approach in literature: complete automation, optimization, and formal correctness assurance. This is made possible by the adopted technique, which is based on the resolution of a partial weighted *Maximum Satisfiability Modulo Theories* (MaxSMT) problem. This type of problem allows, with a carefully designed model of the network configuration and the desired security policies, the computation of a solution that correctly enforces the given requirements while seeking additional optimality goals, going beyond what is achievable with commonly used approaches based on heuristics. Such an automated approach based on formal methods thus proves to be a valid solution to address the continuous generation of network attacks, being able to quickly defend against incoming attacks through the optimized computation of an updated firewall configuration that blocks the attacker within a short time delay, while ensuring formal correctness with respect to all security policies in place.

This paper is an extension of the conference paper published in the proceedings of the 2024 IEEE/IFIP Network Operations and Management Symposium (NOMS 2024) [9]. That conference paper only presented the React-VEREFOO component. Instead, this extension describes the design of all the other components of the process, that is, the modules related to translating and deploying the firewall configuration produced by React-VEREFOO, monitoring IDS logs and extracting new requirements, and merging the extracted requirements with the previous ones. In this way, this paper casts React-VEREFOO into a more complete approach, where all steps of attack mitigation through firewall reconfiguration are automated. Besides, it also describes new validation tests that were carried out to assess the effectiveness of the whole process.

The remainder of this paper is structured as follows. Section 2 contains a summary of the related work. Section 3 describes the proposed approach. Section 4 introduces some key formalisms used to represent the network and the security requirements, but it also discusses the design of the main algorithms and of the MaxSMT problem employed in React-VEREFOO. Section 5 discusses the results of the validation and performance tests conducted on the implementation of the proposed approach. Finally, Section 6 outlines the conclusion and future work.

2 | Related Work

Previous related work can be divided into three main categories: (1) approaches that pursue a similar idea but are applied to a different issue with different characteristics and needs, that is, the problem of routing management (Section 2.1); (2) approaches that are designed for the same problem, that is, reconfiguration of firewalls, but lacking some of the features with respect to our approach (Section 2.2); and (3) approaches for the configuration of distributed firewalls with a similar set of features, namely, automation, formal correctness assurance, and optimality, but without the support for a specific reconfiguration procedure (Section 2.3).

2.1 | Optimized Reconfiguration for Routing Problems

A small number of studies [10–12] adopt an approach similar to the one presented in this paper but address a distinct issue, that is, routing management. Indeed, they deal with routers, routing algorithms, and forwarding policies instead of network security functions and policies. In greater detail, Gember-Jacobson et al. [10] describe the design of the control plane repair algorithm, an approach based on a MaxSMT problem to automatically compute correct and minimal repairs for network control planes. The solution is based on a carefully crafted formal model for the network, the routing protocols, and the exchanged traffic. It supports minimizing the lines written in the configuration as an optimization goal. Abhashkumar et al. [11] outline another synthesis tool, named AED, that formally models the network and its configuration into a MaxSMT problem. The optimality goals considered in the resolution are more refined, allowing the operator to specify different management objectives, such as maintaining structural similarity across devices or minimizing the number of modified devices. Finally, Tian et al. [12] present JINJING, an approach for the automatic and correct update of routing configuration on the base of intents expressed using an ad hoc language. This approach models the network as an SMT problem, leaving as open variables all the elements causing the inconsistencies between the current configuration and desired policies while keeping the other elements fixed. Optimality, in this case, is not present. Moreover, the approach could produce redundancy in the computed rules as it requires a postprocessing task to minimize the lines of the computed configuration, and it just focuses on traditional networks, not allowing the modification of the topology of the control plane but only its configuration. Overall, while these studies share similarities with our work, such as combining a similar set of features (automation, formal verification, and optimization) and focusing on reconfiguration, they operate within a different context.

2.2 | Automatic Fixing of Firewall Configurations

Other studies [13–17] investigate the problem considered in this paper, that is, automatic reconfiguration of firewalls, but they address it partially, as their proposed solutions lack some of the features that are included in our proposal. Chen et al. [13] propose five algorithms to automatically reconfigure a faulty firewall after five corresponding issues (wrong rule order, missing rules, wrong condition predicates, wrong decision actions, and wrong extra rules). This approach uses samples of misclassified packets as input for the reconfiguration, proposing a fix that tries to maximize the number of solved misclassifications detected through samples of misclassified packets used as input of the reconfiguration process. Adi et al. [15] use a dedicated calculus to formally verify if the configuration is compliant with the security policies defined by the user and, if not, to automatically generate the optimal and correct configuration repair. The repair is computed using the adopted calculus and a quotient operator that can compute which changes are needed to reach the desired state and which must comply with the defined security policies. Cheminod et al. [17] illustrate a methodology for configuration refinement, formal verification, and, if needed, the automatic computation of a fixing strategy in case the current configuration does not correctly enforce the user-defined policies. This is based on an SMT model and, for the fixing, on a constraint refinement approach, which keeps everything fixed in the configuration, but the elements causing the anomalies are recomputed by the refinement process. Youssef and Bouhoula [14] compute, whenever a misconfiguration is detected, a formally correct fixing action by resolving a carefully designed SMT problem, and use a formal model for the security policies and the network configuration. Whenever a misconfiguration is detected, it computes a formally correct fixing action solving a carefully designed SMT problem. Hallahan et al. [16] present another approach based on formal models and the design of an SMT problem. It follows a repair-by-example paradigm, providing a set of user-defined examples of the desired filtering behavior as input for the reconfiguration process. Also, in this case, the model can be solved to synthesize a formally correct reconfiguration. Note that this approach does not model the desired security policies, but it follows a repair-by-example paradigm, considering examples of the desired behavior that the user provides. The approach automatically synthesizes new firewall rules for the existing configuration so that the new set of rules respects the provided examples.

Concerning their limitations, Chen et al. [13] and Hallahan et al. [16] cannot guarantee the formal correctness of the configuration with respect to a set of security policies, because they do not model all the traffics but either only those provided in the examples or those involved in a detected misclassification, and so they cannot guarantee the correctness for all traffics. Almost all approaches [13–16] are not designed for distributed firewalls, but they support only single firewall instances. Moreover, they do not support the synthesis of new services from scratch but can only modify those already deployed. Youssef and Bouhoula [14] and Hallahan et al. [16] adopt limited or no optimization for

the computation of the new configuration. The approach proposed by Cheminod et al. [17] is the most complete one in terms of features, but its focus is primarily on access control instead of reachability policies, and it is mostly a description of a possible approach rather than a fully functional solution.

2.3 | Automatic, Formal, and Optimal Firewall Configuration

Finally, some studies propose automatic techniques for allocating or configuring distributed firewall systems with all the features we are considering. Among all the ones that are reported in a state-of-the-art survey about automatic security configuration [2], the most relevant ones are ConfigSynth [18] and VEREFOO [19, 20]. The former automates the generation of the firewall allocation scheme (but not of the configuration) with an optimized and formal approach based on the definition of an iterative SMT problem. The idea is that the architecture is tuned at each step of the algorithm until it properly enforces all the security properties. In this case, the optimization criteria is the minimization of the network security functions allocated in the network. The latter proposes the definition of a MaxSMT problem to model the network and its configuration. The formal assurance is provided with a correctness-by-construction approach, and the involved optimality criteria are the minimization of the number of allocated firewalls and the number of firewall rules so as to reduce the amount of consumed resources.

Despite the relevance of these two studies and other related ones in the same category, they do not provide an optimized procedure for reconfiguration. As they simply regenerate the allocation scheme or configuration from scratch every time, the result may have significant differences with respect to the previous configuration status (e.g., the position of the firewalls is significantly modified or several new firewalls are allocated [21]). Consequently, these differences would lead to a significant delay required to instantiate many new virtual firewalls or to change several rule sets of the preserved ones. Unlike them, our proposed approach has been specifically designed to optimize firewall reconfiguration by minimizing the number of the required changes to satisfy new security requirements.

3 | The Proposed Approach

The proposed methodology aims to provide automatic firewall reconfiguration after attack detection in virtual computer networks through an autonomous process of operations that minimizes the number of interactions with the human administrator. As shown in Figure 1, the approach that we designed involves multiple steps, which are defined for the different tasks of attack mitigation through firewall reconfiguration: initial input specification, firewall configuration translation and deployment, IDS log monitoring and requirement extraction, merging of old requirements with the extracted ones, and firewall reconfiguration computation. The remainder of this section will detail all of them more precisely.

3.1 | Initial Input Specification

The main interaction point between the proposed approach and the human administrator occurs at the beginning of the process, when no firewalls are operative yet in the computer network. At this stage, the administrator must specify the initial inputs that kick off the autonomous loop of mitigation operations. In greater detail, two inputs are required: a service graph (SG) and an initial set of network security requirements (NSRs).

The SG consists in a description of the logical topology of the virtual computer network where security must be enforced. As such, it describes how the network nodes interconnect with each other and provides information about the configuration of network service functions such as network address translators and load balancers. The way these service functions work may impact the satisfaction of security requirements, as they may modify the received traffic. Consequently, their behavior must be known by our automatic approach so that it can later understand where firewalls should be allocated and how they should be configured. For example, a network address translator may change the source or destination IP address of a received packet. Therefore, depending on whether a firewall is allocated before or after the network address translator in the path followed by a traffic flow, it will analyze packets with different values for those address fields so that it will need different filtering rules.

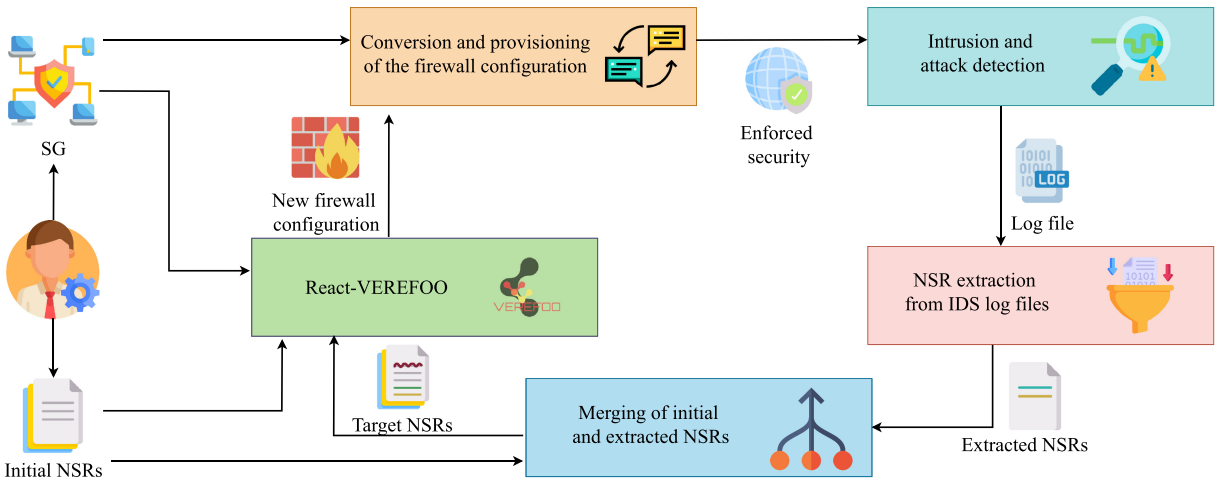


FIGURE 1 | The proposed approach for attack mitigation through firewall reconfiguration.

The input SG also provides information about the configuration of the already present firewall, which will have to be reconfigured in the future. This configuration comprises the allocation scheme, specifying the positions in the network where the firewall instances are allocated, and the filtering rule set for each instance. Additionally, an assumption about the input SG is that all the virtual network functions (VNFs) composing its logical topology are already deployed in the physical network infrastructure, with the exception of firewall instances, because allocating and configuring them is another task of our autonomous process.

The initial set of NSRs describes the security requirements (or policies) that are already enforced in the input SG. As our approach works with packet filtering firewalls, whose objective is to block malicious traffic and enable desired communications, the NSRs that are envisioned in this study are connectivity requirements, that is, policies that provide information about how and if network nodes can communicate with each other. From this point of view, two possible NSR types may exist: isolation requirements if they request that certain packet classes are blocked before reaching their destination because potentially malicious and reachability requirements if they request that certain packet classes must reach their destination to ensure a connectivity service. The packet classes of interest are identified by the condition of each NSR, which expresses the values (or range of values) that each field of the IP 5-tuple must have so that its related packet is considered to belong to the class of interest for a certain NSR.

Both the SG description and the initial set of NSRs are provided by the human administrator through a medium-level policy specification language, which is independent of the specific low-level settings of the network service and security function configuration. In the literature, several languages exist for policy specification to provide different abstraction levels. For this level, XML or JSON representations can be used, so as to abstract from the technicalities of the specific firewall implementations.

Moreover, the definition of the firewall configuration included in the SG and required to satisfy the initial set of input NSRs do not necessarily require manual operators to be performed by the human administrator. In fact, it may be computed automatically with state-of-the-art tools described in Section 2.3 such as VEREFOO [20]. In this way, the administrator simply has to feed the automatic tool with the information needed for firewall configuration.

In general, this operation of initial input specification represents the only interaction point where the human administrator must actively do something to provide information to the automatic approach, but this task cannot be avoided because all autonomous processes need a starting point.

3.2 | Conversion and Provisioning of the Firewall Configuration

The firewall configuration, inclusive of its allocation scheme and filtering rule sets, is not already deployed in the active virtual network. Therefore, a specific module of the methodology is dedicated to the operation of deploying them.

Both provisioning and conversion operations are not very complex. On the one hand, the provisioning operation, that is, deploying the VNFs in the network, is a simple operation because it simply requires the module to interact with the APIs of the specific VNF orchestrator that is used by the administrator to handle VNF management. On the other hand, the configuration that is provided by the user cannot be directly pushed onto the deployed VNFs because it is a medium-level implementation-agnostic language. Consequently, all firewall rule sets must be converted from XML/JSON representations to the specific commands of different firewall implementations, such as iptables, ipfirewall, eBPF firewall, and Open vSwitch. The module envisioned for this task must be flexible enough to be extended easily to other technologies, thus providing forward compatibility. Anyway, this conversion is simply a translation, as it moves the same information from a structured format to another structured format without adding other information or removing any. Therefore, overall, the module dedicated to this operation does not require an intelligent engine to work differently from other modules of the proposed autonomous process. At the same time, it contributes to avoiding that these simple operations are performed manually by a human administrator, who may introduce easily avoidable mistakes.

3.3 | Intrusion and Attack Detection

After the provisioning of the firewall configuration has been completed, the network must be able to identify possible cyber attacks that were not taken into account in the initial set of NSRs. For this purpose, among all the deployed VNFs, some of them must be IDSs. These monitoring functions have the objective of analyzing all packets crossing them to understand if they may possibly belong to a malicious traffic flow. The literature on IDS technologies and behaviors is extremely rich. To mention a few examples, passive IDSs evaluate the received packets against a set of rules expressing the conditions under which the IDS should assume that a specific attack is occurring. Instead, active IDSs try to understand the presence of an attack depending on statistics, such as the occurrence of packets with the same structure and field values in the past. Several active IDS are also enhanced with intelligent algorithms, such as data-driven techniques and Artificial Intelligence strategies based on Support Vector Machines.

Considering the richness of the literature on this topic and the continuous progress in it, it is not the purpose nor contribution of this paper to introduce new strategies for intrusion detection. Therefore, any kind of IDS can be used and work with our proposed approach, as long as the IDS produces a log file that the next steps of the proposed methodology can later exploit.

3.4 | Extraction of Security Requirements from IDS Log Files

Whenever an IDS writes a new log entry, it may notify the occurrence of a cyber attack, for which new NSRs should be

defined, and a new firewall configuration should be produced. Consequently, the first step of such mitigation consists of monitoring the log files, understanding if a new entry is about an attack, and extracting a new NSR for its management.

A specific module of the proposed methodology is responsible for these tasks. Specifically, it continuously monitors log files written by the IDSs deployed in the network. Whenever a new entry is written, it parses it to understand if it is related to a cyber attack. In that case, it extracts all the valuable pieces of information from that entry, identifying the packet class related to the attack from the values of the IP 5-tuple fields. Next, starting from this information, the module identifies all traffic flows related to the identified packet class. In this study, a traffic flow represents how a specific packet class is forwarded and transformed within its path, so multiple flows can exist for each class (e.g., it is simply possible that the packets of this malicious class can come from multiple sources). After identifying the flows, a new NSR is created for each one of them. This new NSR has a condition whose source IP address and port are the ones of the packet class at the beginning of the flow, and whose destination IP address and port are the ones of the packet class at the end of the flow. Those condition fields cannot be directly mutated from the packet class extracted from the log entry because some packet fields may have been modified by intermediate nodes, such as network address translators, while the conditions of an NSR should be defined over an end-to-end communication.

After concluding all these operations, a new set of NSRs is produced, called extracted set of NSRs.

3.5 | Merging of Initial and Extracted Requirements

All the NSRs belonging to the extracted set require satisfaction in an updated version of the firewall configuration to be deployed in the virtual network because they are necessary to stop the cyber attack identified by the monitoring agents and reported in their log files. At the same time, the initial set of NSRs should also be preserved, as long as those NSRs do not conflict with the target ones. For instance, some flows that were permitted by an initial reachability NSR may not have to be blocked because of a target isolation NSR. As each NSR may be associated with multiple flows, partial conflicts are possible, so the problem of conflicting NSRs cannot be solved just by removing the initial conflicting NSR, which instead must be opportunely modified.

A tailored module of the proposed methodology is responsible for performing this merging operation in an enhanced way to create a merged set, also named target set of NSRs, because this is the set of all the NSRs that will have to be actually satisfied in the network, in place of the previous initial NSRs.

The module in charge of creating the target set works according to a strategy based on the following sequential steps:

1. First, it includes all the extracted NSRs into the target set, as they are required to block the detected cyber attack.
2. Then, for each initial NSR, it checks if it conflicts with any extracted NSR (already included in the target set) to decide if and how to include it. In particular, two NSRs are conflicting if they have different actions and (at least partially) overlapping conditions.
3. The initial NSR may conflict or not with some extracted NSRs.
   - If the initial NSR does not conflict with any extracted NSR, it is directly included in the target set.
   - If the initial NSR conflicts with an extracted NSR, the merging module must modify its condition so that it does not include the overlapped part anymore, and then, it can put the modified NSR into the target set. For example, if an initial reachability NSR was previously applied to the TCP traffic targeting all ports in the range [50,800, 50,900] while an extracted isolation NSR imposes that TCP traffic to 50,900 must be blocked because it is used by a cyber attack, then the initial reachability NSR must be modified so that the condition on the destination port is defined on the restricted interval [50,800, 50,899]. Clearly, if there is a complete overlapping between the conditions of the initial and extracted NSRs, the initial NSR is completely removed.

3.6 | React-VEREFOO

After the merging operation has been concluded, the new firewall configuration can finally be produced. This operation is performed by a central module of the proposed methodology, that is, React-VEREFOO.

As shown in Figure 2, React-VEREFOO works on two inputs. The first input is the SG that is currently deployed, that is, the logical topology of a virtual network with an already existing distributed firewall configuration, composed of the allocation scheme of its instances and their filtering rules. The second input is a pair of NSR sets: the initial set of NSRs, including the old NSRs already satisfied by the existing firewall configuration, and the target set of NSRs, coinciding with the previously computed merged set and therefore including the newly extracted NSRs to be enforced in the updated network configuration. The produced outputs are the updated allocation scheme and the reconfigured filtering rules of the firewall.

React-VEREFOO achieves this outcome by combining automation, formal correctness, and optimization. This achievement is feasible because the firewall reconfiguration problem is modeled in React-VEREFOO as a MaxSMT problem. A MaxSMT problem differs from an SMT problem because it allows the definition of two types of clauses: the *hard constraints* that are compulsory and the *soft constraints* that are optional and have an associated weight. The selected solution is the one that satisfies all the hard constraints and maximizes the sum of the weights of the satisfied soft constraints. Hard constraint satisfaction contributes to achieving formal correctness as long as all problem components are formally modeled. The attempt of the solver to achieve soft constraint satisfaction contributes to the optimization of the result.

The approach pursued by React-VEREFOO avoids the need to recompute the entire network from scratch, significantly

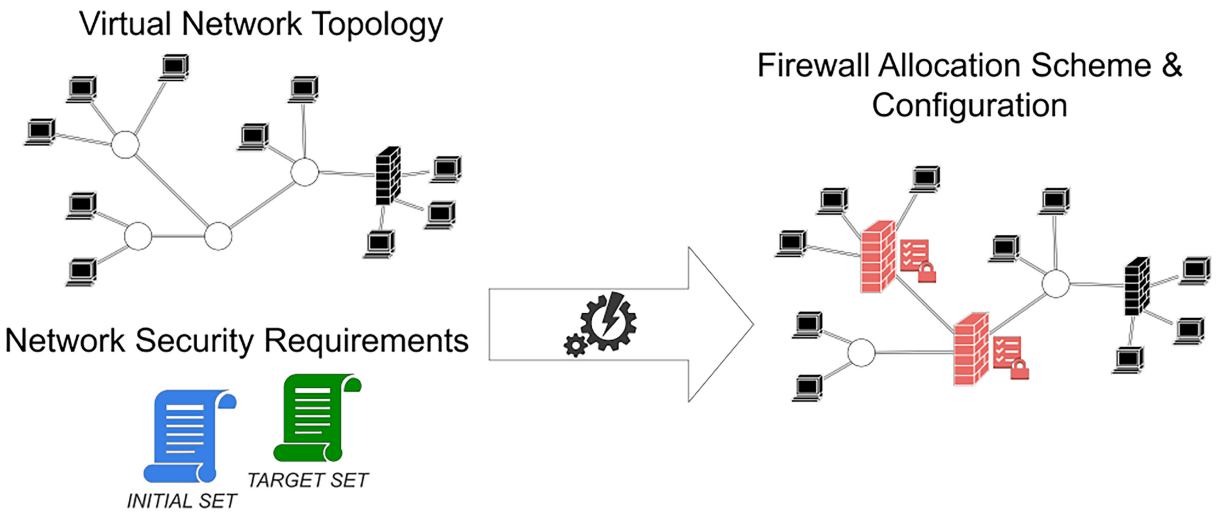


FIGURE 2 | Approach of React-VEREFOO.

reducing computation time. It speeds up the process by narrowing down the space of possible solutions that are analyzed, keeping certain parts of the configuration as fixed, and providing optimality-related clauses, which are provided to the solver, to have a faster convergence toward the optimal solution. Specifically, the optimality objectives are the minimization of resource usage and the preference for reusing the original firewall configuration.

As React-VEREFOO is a main contribution of this study, Section 4 will be dedicated to exploring the details about the formal models used in it, the algorithms to identify the network areas to reconfigure, and the formulation of the MaxSMT problems.

3.7 | Repetition of the Approach

After the new firewall configuration, including the allocation scheme and filtering rules, is computed, the process envisioned in our approach is repeated without any active intervention by the human administrator. The configuration is translated and provisioned in the network. Whenever an IDS writes a new log entry related to an attack that was not considered in the previous target set of NSRs, which have now become the initial set, a new set of NSRs is extracted and is merged with the previous one, so that React-VEREFOO can be executed again.

4 | React-VEREFOO

The approach of React-VEREFOO is composed of multiple steps. The first one involves the definition of a complete and formal model that represents the network, the configuration of the different network functions, and the traffic exchanged (Section 4.1). Then, a central part of this proposal, representing a main novelty introduced here, is the design of an algorithm able to identify the network areas that must be reconfigured based on the intersection of the two sets of NSRs provided as input (Section 4.2). Having computed this intersection, it is possible to discern which NSRs have been added, kept, or deleted in the new set of NSRs with respect to the original one. The algorithm identifies, for all the "added" requirements, that is, those relevant for the reconfiguration scenario, which elements of the provided network should be modified because in conflict with the new set of security requirements. The configuration of these elements is therefore put under question. Finally, the approach formulates a MaxSMT problem whose resolution allows to generate the new allocation graph (AG) and configuration rules of the needed firewalls (Section 4.3). This approach starts from the hypothesis that the configured rules for each firewall are anomaly-free and without overlapping. This is not restrictive because there are techniques that allow to generate such set from nondisjoint rules. Given this hypothesis, the order of rules within each firewall is irrelevant, as each packet will match at most one rule.

4.1 | Formal Models

The formal models used in this paper stem from VEREFOO [19, 20], a policy-based approach for automatic firewall configuration, which has all the features (automation, formal verification, and optimization) we are interested in. Here, we report the main features of those models that are required to understand the remainder of the section, focusing on the ones that are different with respect to the ones already presented in the VEREFOO papers [19, 20].

The logical topology of the input network is modeled as a directed graph, named SG, whose nodes represent all the network functions and endpoints (e.g., web clients, web servers, firewalls, and NATs) and whose edges represent directed connections. However, the SG model is not directly used by the next steps of the proposed approach, but it is preliminarily preprocessed to create an alternative representation, named AG. The main difference is that the AG model is characterized by an extra node type, named *allocation place* (AP), representing a placeholder node that can be used by the MaxSMT solver to potentially allocate a firewall. The transformation of the SG model into the AG consists in adding an AP only in-between pairs of network nodes that do not contain any function that could be reused, for

example, firewalls, as the idea is to reconfigure previously allocated firewall instances whenever it is possible, rather than placing additional ones in other APs.

The packets that may cross the AG are grouped in classes, depending on the values of their header fields. Each packet class, also called traffic in this paper, is represented as a predicate computed over some variables representing the header fields. Packets whose fields have the same values belong to the same traffic, represented by the same predicate, and are therefore managed in the same way by all nodes crossed in the network. The predicate representing the formal model of each traffic is a conjunction of subpredicates, one for each considered packet field. Because this approach works with packet filters, the modeled fields are the five ones composing the IP 5-tuple, that is, source and destination IP addresses, source and destination ports, and protocol type. Each subpredicate can represent a single value, a range of values, or the range of all possible values, denoted with the "*" symbol. For example, 10.22.34. $*$ used as IP address subpredicate stands for 10.22.34.0/24, while $*$ used as port subpredicate stands for the range [0, 65,535]. The set of all different traffics, that is, packet classes, crossing the AG is denoted as T. The generated firewall rules are modeled with the same predicate representing the IP 5-tuple. In this way, the approach produces a set of rules that is independent of the specific implementation, allowing to easily convert the computed configuration into a specific kind of packet filter system (e.g., iptables, ipfirewall, and eBPF firewall) with a translator.

The way each node composing the AG handles each input packet class is then modeled in a way that is as lightweight as possible, by considering only the parameters that are actually relevant for the security reconfiguration problem. In particular, it is modeled by means of two functions, representing respectively the forwarding and transformation behaviors. On the one hand, the function modeling the forwarding behavior of node n_i is $deny_i : T \rightarrow \mathbb{B}$. This function maps an ingress traffic t to true, if and only if n_i blocks all the packets of that traffic. For simplicity, for each node n_i, we denote $\mathcal{D}_i$ the set of denied traffics and $\mathcal{A}_i$ the set of allowed traffics. On the other hand, the function modeling the transformation behavior of node n_i is $\mathcal{T}_i : T \rightarrow T$. This function maps an input traffic t to the corresponding output traffic, after the possible modifications that it may apply. For several function types (e.g., forwarders, traffic monitors, and firewalls), $\mathcal{T}_i$ is the identity function, as they cannot modify the input traffic. Instead, for functions such as NATs and load balancer, it provides the information related to the changes applied to the 5-tuple fields. Note that for the sake of conciseness and performance, we do not model single packets but only packet classes or traffics.

These models (i.e., the ones of packet classes and network node behavior) allow to introduce the concept of *traffic flow*. A *traffic flow* represents how a specific packet class is forwarded and transformed within its path. A flow $f \in F$ is modeled as a list of alternating nodes and packet classes $[n_s, t_{sa}, n_a, t_{ab}, n_b, \ \dots, n_z, t_z d, n_d]$, where each node n_i in the list represents a node crossed by the flow, whereas the traffic t_{ij} is a predicate representing the class of packets transmitted from node n_i to node n_j. The definition of the traffic flows crossing the AG may differ, depending on how single packets are grouped into the corresponding classes. From this point of view, we decided to adopt the grouping strategy named atomic flow (AF) [22]. The reason is that, according to that study, it is the technique that provides more benefits and better performance for solving an automatic (re)configuration problem. In greater detail, this grouping strategy is based on the *atomic predicate* concept, proposed by Yang and Lam [23] for the network reachability problem. 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 initial predicate is equal to the disjunction of a subset of atomic predicates. The idea was to apply this concept to computer networks so that, given a set of predicates of the network, it is possible to compute a set of corresponding atomic predicates that are minimal, unique, and fully representative of the initial set. Then, a flow $f = [n_s, t_{sa}, n_a, t_{ab}, n_b, \ \dots, n_z, t_z d, n_d]$ can be defined atomic if each traffic t_{ij} is an atomic predicate. To compute the set of atomic predicates, and then the set of AFs, we consider the "interesting" predicates extracted from the NSRs and the network configuration. The algorithms are not reported here, because they are already described in literature [22].

Lastly, the security requirements that must be enforced in the AG are modeled as the combination of two elements: a set of specific NSRs R and a general behavior. On the one hand, each specific NSR $r \in R$ is formally modeled as a tuple (a, C), where a is the action that must be applied on packets matching with the condition predicate C. The NSR is defined isolation requirement if the action a is deny, reachability requirement if instead the action a is allow. The set of specific NSRs provided by the user is assumed to be anomaly-free, which is not a limitation because there are many well-known anomaly analysis techniques [24, 25] that easily allow deriving anomaly-free policy sets. On the other hand, the general behavior adopted in this proposal is a "don't care" approach, which allows users to define both reachability and isolation requirements without imposing any restriction on other packet classes (i.e., users are not concerned about the reachability and isolation of packets for which they have not specified a specific NSR).

The formalization of the specific reachability and isolation NSRs in terms of traffic flows management is as follows: an isolation NSR is satisfied if for any associated traffic flow there is at least one node with an allocated function configured to block the traffic in input for that flow, as shown in Equation (2), whereas a reachability NSR is satisfied if there is at least one associated traffic flow that is not blocked from source to destination by any of the crossed nodes, as shown in Equation (1). These will be modeled in the MaxSMT problem as hard constraints, making their satisfaction mandatory. Note that the reported equations are using some utility functions whose meaning is as follows: $\pi(f)$ returns the nodes belonging to flow f (excluding the source), $allocated(n)$ returns true if there is a firewall allocated in node n, and finally, $\tau(f, n)$ returns the traffic in input to node n for flow f. Instead, F_r represents the set of AFs that respect the condition of NSR r.

$$\exists f \in F_r.\ \forall i.\ (n_i \in \pi(f) \wedge allocated(n_i)\ \Rightarrow\ \neg deny_i(\tau(f, n_i))), \quad (1)$$

$$\forall f \in F_r.\ \exists i.\ (n_i \in \pi(f) \wedge allocated(n_i) \wedge deny_i(\tau(f, n_i))). \quad (2)$$

Moreover, the input NSR set R is composed of two subsets: the *Initial set* R_i, including the NSRs that are already enforced in

the existing network modeled by the AG, and the *Target set* R_t, including the new NSRs to be enforced in the updated network configuration.

4.2 | Algorithm for Detection of Network Area to Reconfigure

Starting from the formal models of all the input components (i.e., network topology, function behavior, traffic flows, and NSRs), our approach envisions the execution of an algorithm, designed to detect the network areas that actually require firewall reconfiguration for the satisfaction of the Target NSRs included in R_t.

First, this algorithm classifies each NSR $r \in \{R_t \cup R_i\}$ to one of the following groups: (*i*) $\mathcal{R}_d = \{r \in R_i | r \notin R_t\}$, the "deleted" NSRs which are no more needed in the final configuration but are present in the initial one, (*ii*) $\mathcal{R}_a = \{r \in R_t | r \notin R_i\}$, the "added" NSRs that should be enforced in the final configuration and are not present in the initial one, and (*iii*) $\mathcal{R}_k = \{r \in R_t | r \in R_i\}$, the "kept" NSRs that are already configured in the provided network and should continue to be enforced in the final configuration.

Second, the algorithm detects, for all "added" requirements $\mathcal{R}_a$, that is, those relevant to the reconfiguration scenario, which elements of the provided network should be modified because in conflict with at least a new requirement. Due to the different formulations of the two requirement types, this part of the algorithm is differently formulated for the case of isolation requirements, and the case of reachability requirements. Anyhow, it is important to underline that the algorithm selects as reconfigurable all the nodes which can potentially be used to fulfill the NSRs in $\mathcal{R}_a$, because it cannot decide a priori which is either the optimal node for blocking a traffic or the optimal traffic flow which must be allowed from source to destination.

Considering a new isolation requirement $r \in \mathcal{R}_a$ and a given AG $\mathcal{G}_A$, the procedure to compute the network elements to be reconfigured is presented in Algorithm 1. This procedure starts considering for each isolation requirement, r, all the correlated AFs, F_r. The algorithm checks whether there is a node along the flow path that is currently blocking the incoming traffic for that flow (Lines 3–7). If no such node is found for a given flow, then all the nodes crossed along the flow path are designated as eligible for reconfiguration (Lines 9–11). This would allow the solver to subsequently decide in which node a firewall should be allocated to enforce the isolation requirement r. Figure 3 clarifies this with an example, where the algorithm takes as inputs the two sets of Initial and Target NSRs, along with a partially configured network that comprises a firewall and two forwarders. The new requirement that should be enforced is the isolation requirement for the traffic from the web client 10.0.0.1 to the web server 20.0.0.1. In this case, there are two paths associated with this requirement. A path crosses only the forwarders, whereas the other one crosses also the firewall. As the two paths are characterized by different network node lists, there are two AFs, one for each path, because each flow includes the crossed node list in its model. These two flows are labeled A and B, respectively. The algorithm analyzes each flow to determine if there exists a node blocking the traffic. In this scenario, Flow B encounters a firewall that blocks all traffic through its default action, consequently the condition is satisfied. Instead, Flow A does not cross any network function that is blocking the traffic. So the nodes belonging to its path must be added to the set of nodes to be reconfigured, $N_{reconfigure}$ (i.e., the two forwarders in this case).

Considering a new reachability requirement $r \in \mathcal{R}_a$ and an AG $\mathcal{G}_A$, the procedure for selecting the network elements to be reconfigured is presented in Algorithm 2. In this case, the satisfiability condition is the logical negation of the prior scenario. The algorithm has to look for the existence of an AF that is not blocked by any firewall from the source up to the destination. If such a flow is found to satisfy the reachability condition, the algorithm may terminate before having checked all flows. For all the flows F_r correlated with the reachability requirement r, the algorithm examines whether the nodes along the path are blocking the incoming traffic for the given flow. If this is the case, these nodes are added to a temporary list (Lines 4–8). In the end, if the algorithm does not find an AF satisfying the reachability condition, all the nodes in the temporary list are selected for reconfiguration and added to the set $N_{reconfigure}$. Considering the example in Figure 4,

Algorithm 1 Algorithm for selecting network area to reconfigure for each added isolation requirement.

Input: an isolation requirement r, and an AG $\mathcal{G}_A$
Output: nodes to be reconfigured $N_{reconfigure}$

1: **for** $f \in F_r$ **do**
2: $found \leftarrow False$
3: **for** $n_i \in \pi(f) = [n_1, n_2, \dots, n_d]$ **do**
4: **if** $allocated(n_i)$ & $deny_{n_i}(\tau(f, n_i))$ **then**
5: $found \leftarrow True$
6: break
7: **end if**
8: **end for**
9: **if** $found == False$ **then**
10: $N_{reconfigure} \leftarrow \pi(f)$ ▷ All nodes in the path should be reconfigured
11: **end if**
12: **end for**
13: **return** $N_{reconfigure}$

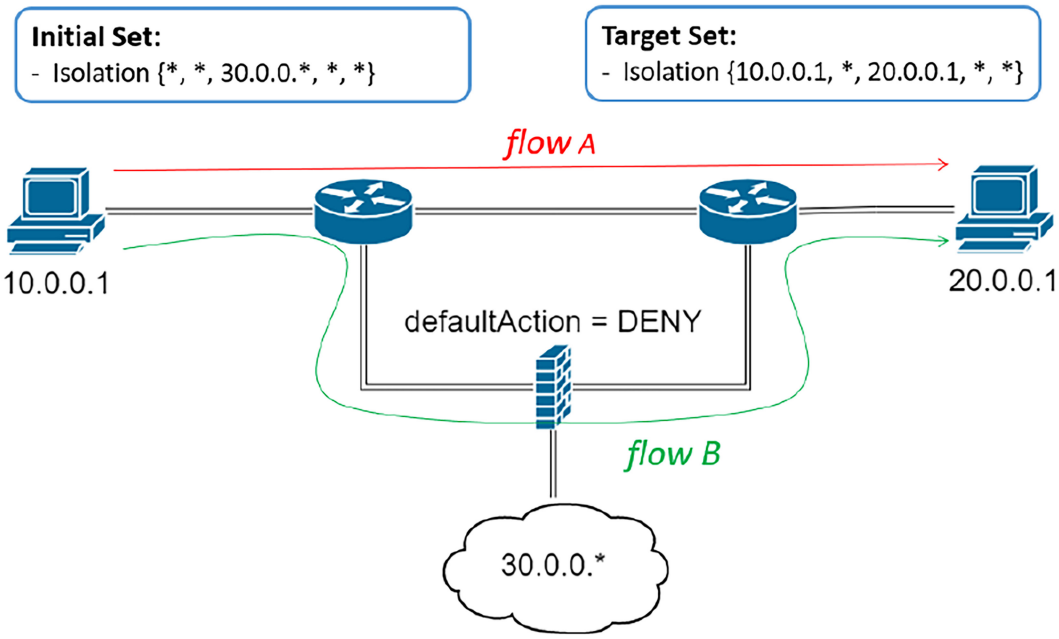


FIGURE 3 | Example of addition of an isolation requirement.

Algorithm 2 Algorithm for selecting network area to reconfigure for each added reachability requirement.

Input: a reachability requirement r, and an AG $\mathcal{G}_A$
Output: nodes to be reconfigured $N_{reconfigure}$

```
tmpReconfigured ← ∅
for f ∈ F_r do
    tmpFlow ← ∅
    for n_i ∈ π(f) = [n_1, n_2, ..., n_d] do
        if allocated(n_i) & deny_{n_i}(τ(f, n_i)) then
            tmpFlow ← n_i
        end if
    end for
    if tmpFlow.isEmpty() then
        tmpReconfigured ← ∅
        break
    else
        tmpReconfigured ← tmpFlow
    end if
end for
return N_reconfigured ← tmpReconfigured
```

we encounter a situation analogous to the previous scenario but with different inputs. In this case, the network consists of two firewalls configured in whitelisting mode and a forwarder. The new requirement that should be added is the reachability from node 10.0.0.1 to node 20.0.0.1, encompassing all possible ports and protocol types. This requirement is associated with two paths, each with an associated AF. These are referred as A and B. In this instance, the algorithm checks whether at least one of these flows is not blocked by any firewall. If this is not the case, the algorithm proceeds to select for reconfiguration, in each flow, the nodes that are blocking the traffic. In this specific case, both flows encounter a firewall that is blocking their traffic. As a result, the algorithm selects the nodes that are blocking both flows, as the solution would be to reconfigure either FW1 or FW2. This ensures that at least one AF can reach the destination, thus meeting the reachability requirement condition.

4.3 | MaxSMT Problem Formulation

After the algorithm has identified all the firewall instances that may potentially require reconfiguration, this information is used, jointly with the formal models of the input, for the formulation of the MaxSMT problem. The core of this formulation is

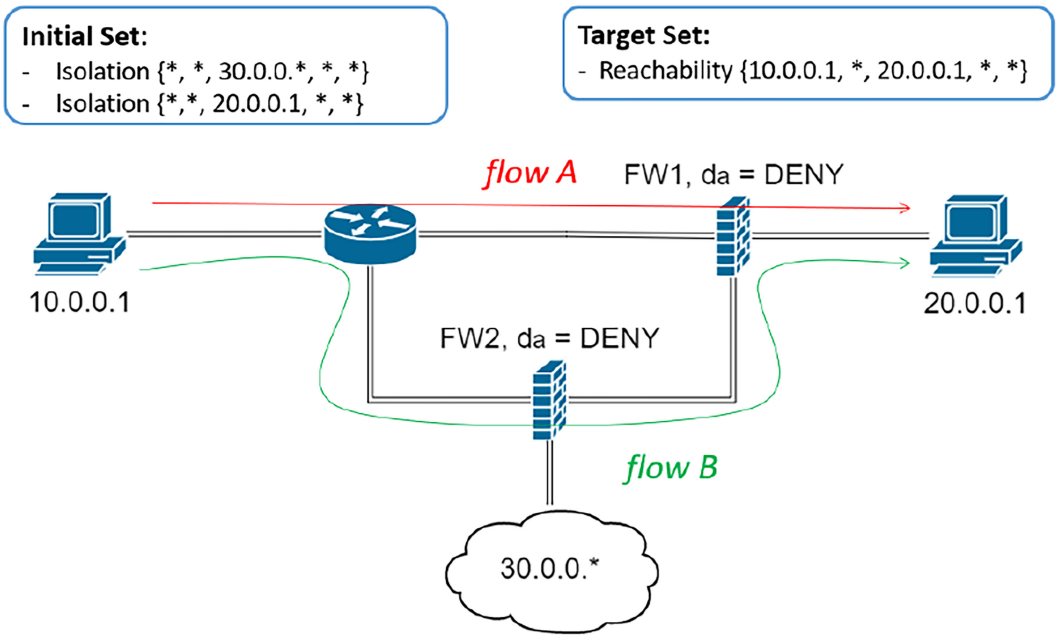


FIGURE 4 | Example of addition of a reachability requirement.

mutated from the one proposed for automatic firewall configuration from scratch [19, 20]. However, some key changes have been introduced to contextualize that formulation to the optimized reconfiguration scenario.

A first difference is that the approach proposed in this paper restricts the set of APs that are available for the solver to allocate firewall instances and keeps some of them as static elements, which cannot be updated within the security configuration. Moreover, to further optimize this approach, also the soft constraints involved in the optimization phase have been adjusted, leading to an additional performance improvement. The guiding principle is that the optimal reconfiguration is the one that does not require any update to the network, thus causing the lowest possible delay. Following this idea, the soft constraints have been updated to prefer an already allocated firewall instead of deploying a new one, as well as an already configured rule must be preferred with respect to a newly generated one.

The same classes of soft constraint have been employed to express both objectives, namely, the minimization of resource usage and the preference of reusing the original firewall configuration. However, the weights associated with these constraints differ in relation to the considered subject, whether it is an empty AP or a reconfigured firewall, or a newly generated firewall rule with respect to an already configured one. This difference is such that the usage of a reconfigured node, as any of its rules, would result in a higher sum of weight and a preferred solution. As a result, the final configuration will be the one that not only minimizes the number of consumed resources in general but also the one that produces the smallest number of changes with respect to the initial configuration.

In particular, the soft constraint regulating the allocation of a firewall for each node n in the set of APs $\mathcal{A}$ is formulated as in Equation (3). This instructs the solver to prefer a solution that does not allocate a firewall for any APs. Indeed, the nonallocation soft constraint produces a contribution to the sum of weights equal to c_k.

$$\forall a_i \in \mathcal{A}.\ Soft(\neg allocated(a_i), c_k). \tag{3}$$

Similarly, the soft constraint regulating the configuration of firewall rules is presented in Equation (4). In this case, a different weight c_{ki}, smaller than the previous, is assigned to the non-configuration of each potential firewall rule p_i. The set P_k contains all the rules that should be potentially configured in a firewall if it is allocated.

$$\forall p_i \in P_k.\ Soft(\neg configured(p_i), c_{ki}). \tag{4}$$

In the approach optimized for reconfiguration, the weights associated to these soft constraints are modified for the nodes in $N_{reconfigured}$. The first soft constraint has been modified in such a way that the nonallocation of each firewall in $N_{reconfigured}$ produces a contribution c_k^R to the sum of weights, such that $c_k^R < c_k$. In this way, the nonallocation of an empty AP would be preferred to the nonallocation of a reconfigured firewall because it has a higher weight. The same principle is applied for the second soft constraint. In this case, for each firewall in $N_{reconfigured}$, any of the configured rules p_i has an associated weight c_{ki}^R, such that $c_{ki}^R < c_{ki}$. As before, this implies that the non-configuration of a new potential rule is preferred with respect to the non-configuration of a previously used one.

Finally, it is worth noting that the proposed approach may only produce a solution that is optimal with respect to the network areas identified as to be reconfigured and not an optimal solution in a global sense. Nevertheless, this limitation is compensated by the improved computation time, which represents a more critical parameter in the proposed scenario of a cybersecurity attack.

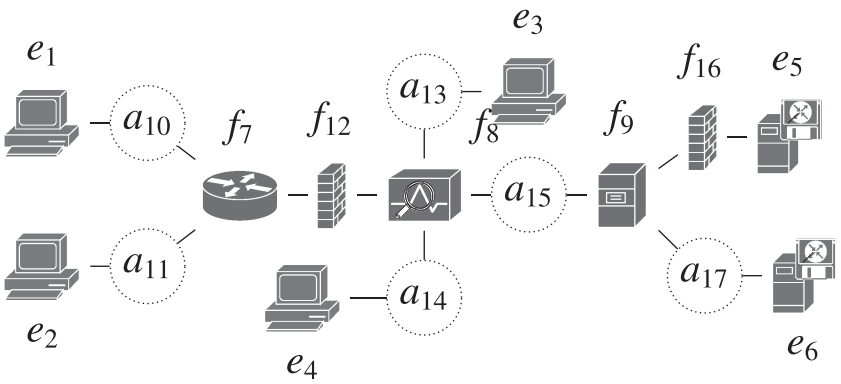


FIGURE 5 | Initial service graph.

TABLE 1 | Current NSRs.

Action	IPSrc	IPDst	pSrc	pDst	tProto
Allow	57.73.0. *	84.20.2.1	*	*	TCP
Allow	84.20.2.1	57.73.0.1	*	*	*
Allow	232.61.10.2	84.20.2.1	*	37894	UDP
Allow	84.20.2.1	232.61.10.2	37894	*	UDP
Allow	232.61.10.2	84.20.2.1	*	37894	UDP
Allow	84.20.2.1	232.61.10.2	37894	*	UDP
Allow	84.20.2.2	232.61.10.2	37895	*	TCP
Allow	232.61.10.2	84.20.2.2	*	37895	TCP
Allow	232.61.10.2	232.61.10.1	*	*	*
Allow	232.61.10.1	232.61.10.2	*	*	*
Deny	232.61.10.1	57.73.0.1	*	*	*
Deny	232.61.10.2	57.73.0.2	*	*	*
Deny	57.73.0.1	232.61.10.1	*	*	*
Deny	57.73.0.2	232.61.10.2	*	*	*
Deny	84.20.2.1	84.20.2.2	*	*	*
Deny	84.20.2.2	84.20.2.1	*	*	*
Deny	232.61.10.1	57.73.0.1	*	*	*

5 | Implementation and Validation

The implementation of the proposed approach was carried out by developing prototypes for the different modules composing its approach:

- The module for firewall configuration translation has been implemented with the Java language. As input medium-level policy specification language, this module can work with one of the most commonly known languages in the literature, that is, the Medium Security Policy Language (MSPL), based on an XML format. As output, it can produce configuration rules for multiple types of firewall solutions: iptables, ipfirewall, Open vSwitch, and eBPF firewall are already supported, but the framework is flexible enough to be extended to support other firewall technologies. For the conversion from MSPL documents to the low-level settings of these firewalls, the JAXB library is used to convert XML documents to Java objects internally (and vice versa). This internal conversion eases the final translation to the low-level firewall configuration settings.
- The module for NSR extraction from IDS log files has been developed with the python3 language, and it is a continuously active framework that monitors IDS log files, to see if new entries are added after the detection of a cyber attack. Currently, the implemented prototype for this module already supports the analysis of two state-of-the-art IDS solutions, which are OSSEC3.7 and Snort3. However, also this module has been implemented in a way to ease extension to other IDS technologies. The extracted NSRs are similarly

TABLE 2 | Current firewall filtering rules.

#	Action	IPSrc	IPDst	pSrc	pDst	tProto
Firewall f_{12}						
1	Allow	232.61.10.2	84.20.2.1	*	37894	UDP
2	Allow	84.20.2.1	232.61.10.2	37894	*	UDP
3	Allow	84.20.2.2	232.61.10.2	37895	*	TCP
4	Allow	232.61.10.2	84.20.2.2	*	37895	TCP
D	Deny	*. *. *. *	*. *. *. *	*	*	*
Firewall f_{16}						
1	Allow	57.73.0. *	84.20.2.1	*	*	TCP
2	Allow	84.20.2.1	57.73.0.1	*	*	*
3	Allow	232.61.10.2	84.20.2.1	*	37894	UDP
4	Allow	84.20.2.1	232.61.10.2	37894	*	UDP
D	Deny	*. *. *. *	*. *. *. *	*	*	*

expressed in MSPL, to provide forward compatibility to all other modules of the whole framework.

- The module for NSR merging has been developed with the python3 language, and works with two sets of NSRs (initial and extracted sets of NSRs) to produce a target set of NSRs. All NSRs are still specified in MSPL.
- React-VEREFOO has been implemented as a Java-based framework, and it employs the open-source Z3 theorem prover [26], developed by Microsoft Research, to solve the formulated MaxSMT problem.
- All these modules have interfaces for their communications (e.g., firewall reconfiguration is produced by React-VEREFOO after a request is sent to its REST interface by the module dedicated to NSR merging).

```
<rule id=''100009'' level=''1''>
      <options>no_log</options>
      <decoded_as>iptables</decoded_as>
      <description>TCP SYN request detected</description>
</rule>
<rule id=''100010'' level=''10'' frequency=''20'' timeframe=''60''>
      <if_matched_sid>100009</if_matched_sid>
      <decoded_as>iptables</decoded_as>
      <description>TCP SYN port scan detected</description>
      <same_source_ip />
</rule>
```

Listing 1 OSSEC configuration

The framework has been validated both in terms of effectiveness (Section 5.1) and performance (Section 5.2) with a series of tests. All the MaxSMT instances have been solved on a machine with an Intel i7-6700 CPU at 3.40 GHz, 32 GB of RAM, and Z3 version 4.8.5.

5.1 | Effectiveness Validation

The effectiveness and efficacy of the framework have been validated on some use cases, where attacks were simulated and a new firewall configuration had to be produced. In this subsection, we report a use case example to show how the framework reacts to stop an ongoing attack of Denial of Service (DoS). In this use case, a TCP SYN port scan is simulated. This kind of attack can discover the state of TCP ports without establishing a full connection, and it can thus understand if there are open ports exploitable to carry out an attack to the victim host.

The network that was used for the execution of this experiment was realized with Dockers and its topology, inspired from our research laboratory computer network, is graphically depicted in Figure 5. In this network, the currently enforced NSRs are the ones listed in Table 1. Besides, there are two already installed firewalls, f_{12} and f_{16}, configured as shown in Table 2, and there is an IDS f_8 with OSSEC3.7 configured as reported in Listing 1.

Rule 100009 matches any TCP SYN request without logging it. This rule must work jointly with rule 100010, which is activated when at least 20 TCP SYN requests from the same source IP are identified within 60 s, notifying a potential port scan.

The attack simulation starts by accessing the Docker representing end point e_4 with a shell:

```
sudo docker exec -it e4 /bin/sh
```

The port scan is simulated with `nmap`:

`nmap -p 1-25 84.20.2.1` This command sends 25 TCP SYN requests to e_5, triggering the rule in OSSEC and resulting in the following alert, represented by the log entry listed in Listing 2.

```
{
      ``rule: {
            ``comment: ``TCP SYN port scan detected,
            ``sidid: 100010,
            ``frequency: 20,
      },
      ``protocol: ``TCP,
      ``srcip: "57.73.0.2",
      ``dstip: "84.20.2.1",
      ``agent_name: ``f8,
      ``timestamp: ``2024 July 24 08:26:31,
      ``logfile: "/var/log/ulog/syslogemu.log"
}
```

Listing 2 Log entry notifying the attack detection

When the module focusing on NSR extraction sees that a new log entry has been written by the IDS, it automatically parses it and extracts an isolation NSR written formatted as the XML object reported in Listing 3.

```
<PropertyDefinition>
<Property graph=``0'' name=``IsolationProperty''
src="57.73.0.2" dst="84.20.2.1" lv4proto=``TCP''/>
</PropertyDefinition>
```

Listing 3 Extracted NSR

At this point, the merging module merges the extracted NSR with the initial set of NSRs. The only conflicting initial NSR is the one appearing at the top of Table 1. That reachability NSR requested that the TCP traffic from all endpoints with IP addresses in the 57.73.0.0/24 range must be able to contact the IP address 84.20.2.1. However, this is not acceptable anymore, because the TCP traffic from 57.73.0.2 must now be stopped. Therefore, the merging module modifies that initial NSR so that the only acceptable source IP address is 57.73.0.1, and introduces the extracted isolation NSR in that list.

When the target set of NSRs is thus created, React-VEREFOO uses it to formulate a MaxSMT problem and compute the new optimized firewall reconfiguration. In particular, it finds the reconfiguration solution where no new firewalls are added, thus avoiding delays related to the deployment of new VNFs. Instead, in this solution, the filtering rules of the firewall f_{16} are just modified. The new rules are reported in Table 3.

Then, the conversion module converts the medium-level configuration produced by React-VEREFOO to iptables commands, producing the script reported in Listing 4.

```
#!/bin/sh
cmd=``sudo iptables''
${cmd} -F
${cmd} -P INPUT DROP
${cmd} -P FORWARD DROP
${cmd} -P OUTPUT DROP
${cmd} -A FORWARD -p tcp -s 57.73.0.1/32 -d 84.20.2.1/32 --dport ACCEPT
${cmd} -A FORWARD -s 84.20.2.1/32 -d 57.73.0.1/32 --dport ACCEPT
${cmd} -A FORWARD -p tcp -s 232.61.10.2/32 -d 84.20.2.1/32 --dport 37894 -j ACCEPT
${cmd} -A FORWARD -p tcp -s 84.20.2.1/32 -d 232.61.10.2/32 --sport 37894 -j ACCEPT
```

Listing 4 Translated iptables configuration

As soon as the configuration of f_{16} is updated, the attack is successfully stopped. This result has been experimentally confirmed by reapplying the command to perform a port scan from e_4 to e_5. This time, the attack turns unsuccessful.

5.2 | Performance Validation

The framework has been extensively tested to assess its correctness and its performance improvement with respect to the traditional approach for firewall configuration from scratch. The validation was carried out on synthetic networks of increasing sizes, generated as extensions of the network shown in Figure 5, and under various reconfiguration scenarios. The performance tests were designed to evaluate how much the results obtained with the proposed optimized reconfiguration approach deviate from those obtained with a state-of-the-art approach lacking support for optimized reconfiguration. For this purpose, the comparison has been done with the official implementation of VEREFOO [27]. The evaluation also covered the achievement of the optimality goals, quantifying the deviations of the proposed approach from the global optimum in terms of resource consumption. As mentioned in 4.3, this approach considers a limited subset of the solution space, thus it may compute a configuration that is locally optimal concerning the reconfigured nodes but not in a global sense.

The main parameters used in the different test cases are the number of NSRs, the number of endpoints, and number of NATs (which introduces an additional complexity factor, and they can modify the crossing traffic). Another important parameter is *PercReqKept*, which represents the percentage of

TABLE 3 | Updated firewall filtering rules.

#	Action	IPSrc	IPDst	pSrc	pDst	tProto
Firewall f_{16}						
1	Allow	57.73.0.1	84.20.2.1	*	*	TCP
2	Allow	84.20.2.1	57.73.0.1	*	*	*
3	Allow	232.61.10.2	84.20.2.1	*	37894	UDP
4	Allow	84.20.2.1	232.61.10.2	37894	*	UDP
D	Deny	*. *. *. *	*. *. *. *	*	*	*

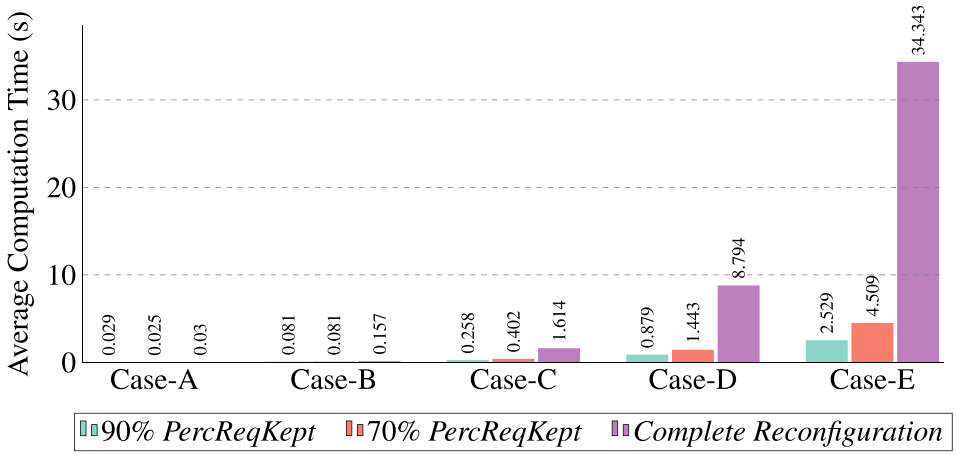


FIGURE 6 | Performance tests.

requirements in the set $\mathcal{R}_k$ with respect to the complete set of NSRs. In other words, it represents the proportion of kept requirements with respect to the total number of defined NSRs. Clearly, if the percentage of requirements maintained between the Initial and Target sets of NSRs increases, then the number of NSRs in the added ($\mathcal{R}_a$) and deleted ($\mathcal{R}_d$) groups will consequently decrease.

The first analysis, shown in Figure 6, compares the performance of the algorithm in five different types of networks, differentiated by an increasing number of endpoints, NSRs and NATs, and with two reconfiguration scenarios, differentiated by the value of the *PercReqKept* parameter. In particular, the five different classes of networks that have been tested are *Case-A* with 10 NSRs, 60 endpoints, and 5 NATs; *Case-B* with 15 NSRs, 80 endpoints, and 10 NATs; *Case-C* with 20 NSRs, 100 endpoints, and 15 NATs; *Case-D* with 25 NSRs, 120 endpoints, and 20 NATs; and *Case-E* with 30 NSRs, 140 endpoints, and 25 NATs. Then, the application of the framework to each of these network classes has been tested in two different reconfiguration scenarios, each characterized by a decreasing value of *PercReqKept*, and, as a consequence, an increased number of modified NSRs. The values adopted for this parameter are 90% and 70%. Note that the tested scenarios are concentrated on higher values of *PercReqKept*, as justified by multiple sources. For instance, it is reported [28] that updates in the Facebook infrastructure affect on average 157 lines of configuration, considering only the backbone, or 738 lines, if also data centers and edge servers are considered. Both are relatively small numbers when compared to the huge scale of their network. Instead, other researchers [29] questioned different online service providers and found out that, for 75% of the networks operated by them, the median change includes only three devices.

For each value of *PercReqKept*, and for each type of network, the algorithm has been executed 100 times. Moreover, this analysis aimed to highlight the improvement versus an unoptimized approach, which is referred in the tests as the *Complete Reconfiguration* case, and it corresponds to the vanilla version of VEREFOO.

The observed trend is that the computation time is directly proportional to the number of endpoints and NSRs and inversely proportional to the percentage of kept requirements. Every considered reconfiguration scenario achieves an average computation time significantly lower than the approach adopted in the vanilla VEREFOO. The obtained results highlight that the main parameters increasing the computation time are the number of endpoints, the number of NSRs, and just for the reconfiguration case, the percentage of NSRs which are maintained, that is, the *PercReqKept* parameter. Indeed, the Initial and Target NSR sets,

once overlapped, form a shared region representing the group $\mathcal{R}_k$. The larger this area, the smaller the sets $\mathcal{R}_a$ and $\mathcal{R}_d$, representing the added and deleted NSRs, and fewer NSRs must be processed, making the reconfiguration process less computationally expensive.

These results show that the highest advantage in terms of computation time is obtained when the reconfiguration regards a small subset of the total NSRs. This is an expected result. In fact, the optimization improvement of the presented approach is mainly achieved by limiting the solution space that is considered by the solver. This reduction is achieved by fixing the configurations of some network elements which are unaffected by the new NSRs, shrinking the set of variables whose values must be determined with the resolution of the problem. If the modified NSRs represent a major part of the whole set, then the unaffected area is reduced, and the optimization effect is limited. This validation phase also assessed that the impact of the designed algorithm could be considered irrelevant when compared to the overall computation time, because its contribution always ranged between 0 and 100 ms, with most of the runs being under 10 ms. In general, the results confirm the feasibility of the proposed approach and its relevant advantages in terms of computation time when compared to the previous solution, based on a complete reconfiguration of the whole network.

In this phase, also the optimality of the solution has been analyzed. The results show that the reconfiguration approach achieves a slightly higher number of allocated firewalls and configured rules. The extent of this difference changes depending on the ratio between the weight assigned to a reconfigured node and that used for a new one. This is demonstrated by the additional tests shown in Figure 7. Two different cases are represented here, one in which the ratio between the weight assigned to a new AP and the weight assigned to a reconfigured node is equal to 2, in Figure 7a, and another case in which the same ratio is equal to 10, Figure 7b. As we can see, increasing this ratio results in an increase for the number of generated firewall rules, and the same applies to the number of firewalls (even if not represented here). The suboptimality of the result is due to two factors: first, the reduction of the solution space for the solver which is limited to the subset of APs that could be modified, and second, the soft constraints which force the preference of reusing old configuration elements even if a completely new configuration would result in a slightly better optimality. Note that the results for performance and scalability have been conducted using the value 2 for the ratio, which allows to reduce the computation time while achieving a nearly optimal usage of resources. All the values assigned to the weights of each soft constraint are the results of different tests that have been conducted.

The second validation analysis tested the approach with larger networks and considering just a single reconfiguration scenario in which 70% of NSRs are kept from the Initial to the Target set. Figure 8 compares the obtained average computation time for the proposed approach compared with the previous one. The considered network types have increasing sizes, specifically the considered cases are from left to right: 200 NSRs and 40 endpoints, 300 NSRs and 60 endpoints, with 400 NSRs and 80 endpoints, and 500 NSRs and 100 nodes. As we can see, even considering the high variability of the obtained values, the reconfiguration approach performs well when compared to the previous one also in terms of scalability. In this specific case, which can be considered as a sort of upper bound, the average computation time decreases by a factor of 60% over a similar nonoptimized approach. When the update involves a smaller portion, the gain is even higher.

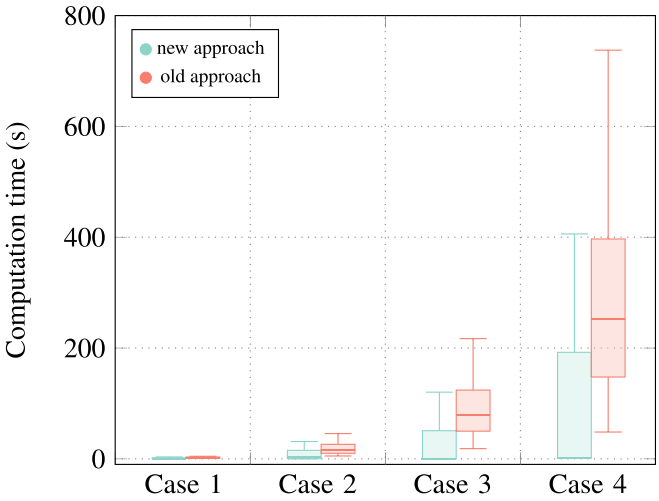
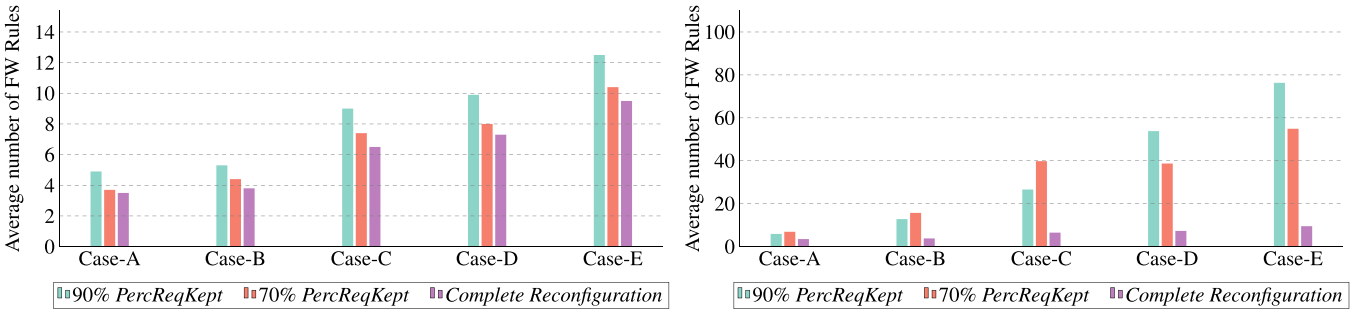


FIGURE 8 | Scalability tests.



(a) Results with ratio 2

(b) Results with ratio 10

FIGURE 7 | Optimality comparison.

6 | Conclusion and Future Work

This paper presented an autonomous methodology for the optimized reconfiguration of distributed firewall systems. This methodology aims to avoid human interventions when an attack must be mitigated in order to avoid mistakes in firewall configurations being introduced when humans try to step on them. This methodology is composed of multiple steps, such as the one for converting firewall configurations from medium-level policy languages to concrete low-level settings and the one for extracting security requirements automatically from IDS log files. However, a core component of this approach is React-VEREFOO, which models firewall reconfiguration as a MaxSMT problem. Thanks to this formalization, to the best of our knowledge, React-VEREFOO is the first one in the literature to address that problem while combining three main features: full automation in computing the firewall reconfiguration, formal correctness assurance of the computed configuration, and optimizations in terms of resource consumption. The proposal was designed considering use cases of network attacks, requiring the computation of a new formally correct and secure solution within a short computation time, so as to limit the exposure of the systems to the attack. The proposed strategy has been implemented as a framework, whose validation showed benefits in terms of performance with respect to a state-of-the-art technique that automatically computes the firewall configuration from scratch.

As future work, further evaluations with real and more extensive networks are currently ongoing to improve our claims, considering real networks and business use cases and also assessing the different contributions of each operation to the total computation time. We will then evaluate to extend the current methodology to the reconfiguration of other network security functions, such as antispam filters and web application firewalls, and to other possible firewall countermeasures to fight against intrusions, such as forwarding malicious traffic to a honeynet. Finally, we will investigate the possible impacts of false positives on the operations of this methodology and new mechanisms to unblock ports once a related attack is terminated, so as to further improve the effectiveness of the proposed approach.

Acknowledgements

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union—NextGenerationEU.

Conflicts of Interest

The authors declare no conflicts of interest.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

1. Verizon, "2024 Data Breach Investigations Report," (2024), https://www.verizon.com/business/resources/Tef6/reports/2024-dbir-data-breach-investigations-report.pdf, Visited: 2024-09-03.

2. D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "Automation for Network Security Configuration: State of the Art and Research Trends," *ACM Computing Surveys* 56, no. 3 (2024): 57:1–57:37, https://doi.org/10.1145/3616401.

3. S. Singh, Y.-S. Jeong, and J. H. Park, "A Survey on Cloud Computing Security: Issues, Threats, and Solutions," *Journal of Network and Computer Applications* 75 (2016): 200–222, https://doi.org/10.1016/j.jnca.2016.09.002.

4. H. Tabrizchi and M. K. Rafsanjani, "A Survey on Security Challenges in Cloud Computing: Issues, Threats, and Solutions," *Journal of Supercomputing* 76, no. 12 (2020): 9493–9532, https://doi.org/10.1007/s11227-020-03213-1.

5. A. Paradowski, L. Liu, and B. Yuan, "Benchmarking the Performance of Openstack and Cloudstack," in *Proceedings of the 17th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, ISORC 2014* (Reno, NV: IEEE, 2014), 405–412, https://doi.org/10.1109/ISORC.2014.12.

6. G. M. Yilma, F. Z. Yousaf, V. Sciancalepore, and X. P. Costa, "Benchmarking Open Source NFV MANO Systems: OSM and ONAP," *Computer Communications* 161 (2020): 86–98, https://doi.org/10.1016/j.comcom.2020.07.013.

7. Cloudflare, "DDoS Attack Trends for 2022 Q2," (2022), https://blog.cloudflare.com/ddos-attack-trends-for-2022-q2/ (Visited: 2024-07-25).

8. Proton, "A Brief Update Regarding Ongoing DDoS Incidents," (2022), https://proton.me/blog/a-brief-update-regarding-ongoing-ddos-incidents, (Visited: 2024-07-25).

9. F. Pizzato, D. Bringhenti, R. Sisto, and F. Valenza, "Automatic and Optimized Firewall Reconfiguration," in *Proceedings of NOMS 2024 IEEE Network Operations and Management Symposium, May 6-10, 2024* (Seoul, Republic of Korea: IEEE, 2024), 1–9, https://doi.org/10.1109/NOMS59830.2024.10575212.

10. A. Gember-Jacobson, A. Akella, R. Mahajan, and H. H. Liu, "Automatically Repairing Network Control Planes Using an Abstract Representation," in *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China: Association for Computing Machinery, 2017), 359–373, https://doi.org/10.1145/3132747.3132753.

11. A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "AED: Incrementally Synthesizing Policy-Compliant and Manageable Configurations," in *Proceedings of CoNEXT '20: The 16th International Conference on Emerging Networking EXperiments and Technologies* (Barcelona, Spain: Association for Computing Machinery, 2020), 482–495, https://doi.org/10.1145/3386367.3431304.

12. B. Tian, X. Zhang, E. Zhai, et al., "Safely and Automatically Updating In-Network ACL Configurations With Intent Language," in *Proceedings of the ACM Special Interest Group on Data Communication, SIGCOMM 2019* (Beijing, China: Association for Computing Machinery, 2019), 214–226, https://doi.org/10.1145/3341302.3342088.

13. F. Chen, A. X. Liu, J. Hwang, and T. Xie, "First Step Towards Automatic Correction of Firewall Policy Faults," *ACM Transactions on Autonomous and Adaptive Systems* 7, no. 2 (2012): 1–24, https://doi.org/10.1145/2240166.2240177.

14. N. B. Youssef and A. Bouhoula, "A Fully Automatic Approach for Fixing Firewall Misconfigurations," in *Proceedings of the 11th IEEE International Conference on Computer and Information Technology, CIT 2011* (Pafos, Cyprus: IEEE, 2011), 461–466, https://doi.org/10.1109/CIT.2011.84.

15. K. Adi, L. Hamza, and L. Pene, "Automatic Security Policy Enforcement in Computer Systems," *Computers & Security* 73 (2018): 156–171, https://doi.org/10.1016/j.cose.2017.10.012.

16. W. T. Hallahan, E. Zhai, and R. Piskac, "Automated Repair by Example for Firewalls," *Formal Methods in System Design* 56, no. 1 (2020): 127–153, https://doi.org/10.1007/s10703-020-00346-0.

17. M. Cheminod, L. Durante, L. Seno, F. Valenza, and A. Valenzano, "A Comprehensive Approach to the Automatic Refinement and Verification of Access Control Policies," *Computers & Security* 80 (2019): 186–199, https://doi.org/10.1016/j.cose.2018.09.013.

18. M. A. Rahman and E. Al-Shaer, "Automated Synthesis of Distributed Network Access Controls: A Formal Framework With Refinement," *IEEE Transactions on Parallel and Distributed Systems* 28, no. 2 (2017): 416–430, https://doi.org/10.1109/TPDS.2016.2585108.

19. D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Automated Optimal Firewall Orchestration and Configuration in Virtualized Networks," in *Proceedings of NOMS 2020—IEEE/IFIP Network Operations and Management Symposium* (Budapest, Hungary: IEEE, 2020), 1–7, https://doi.org/10.1109/NOMS47738.2020.9110402.

20. D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Automated Firewall Configuration in Virtual Networks," *IEEE Transactions on Dependable and Secure Computing* 20, no. 2 (2023): 1559–1576, https://doi.org/10.1109/TDSC.2022.3160293.

21. D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Towards a Fully Automated and Optimized Network Security Functions Orchestration," in *2019 4th International Conference on Computing, Communications and Security (ICCCS)* (Rome, Italy: IEEE, 2019), 1–7, https://doi.org/10.1109/CCCS.2019.8888130.

22. 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* 21, no. 4 (2024): 3740–3758, https://doi.org/10.1109/TNSM.2024.3407159.

23. H. Yang and S. S. Lam, "Real-Time Verification of Network Properties Using Atomic Predicates," *IEEE/ACM Transactions on Networking* 24, no. 2 (2016): 887–900, https://doi.org/10.1109/TNET.2015.2398197.

24. E. Al-Shaer, H. H. Hamed, R. Boutaba, and M. Hasan, "Conflict Classification and Analysis of Distributed Firewall Policies," *IEEE Journal on Selected Areas in Communications* 23, no. 10 (2005): 2069–2084, https://doi.org/10.1109/JSAC.2005.854119.

25. F. Valenza, S. Spinoso, and R. Sisto, "Formally Specifying and Checking Policies and Anomalies in Service Function Chaining," *Journal of Network and Computer Applications* 146 (2019): 102419, https://doi.org/10.1016/j.jnca.2019.102419.

26. L. de Moura and N. S. Bjørner, "Z3: An Efficient SMT Solver," in *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, Vol. 4963 (Budapest, Hungary: Springer, 2008), 337–340, https://doi.org/10.1007/978-3-540-78800-3_24.

27. VEREFOO, "VEREFOO GitHub Page," (2024), https://github.com/netgroup-polito/verefoo/ (Visited: 2024-07-25).

28. Y.-W. E. Sung, X. Tie, S. H. Y. Wong, and H. Zeng, "Robotron: Top-Down Network Management at Facebook Scale," in *Proceedings of the ACM SIGCOMM 2016 Conference* (Florianopolis, Brazil: Association for Computing Machinery, 2016), 426–439, https://doi.org/10.1145/2934872.2934874.

29. A. Gember-Jacobson, W. Wu, X. Li, A. Akella, and R. Mahajan, "Management Plane Analytics," in *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015* (Tokyo, Japan: Association for Computing Machinery, 2015), 395–408, https://doi.org/10.1145/2815675.2815684.