A novel abstraction for security configuration in virtual networks

A novel abstraction for security configuration in virtual networks

Daniele Bringhenti[a,*], Riccardo Sisto[a], Fulvio Valenza[a]

[a] *Dipartimento di Automatica e Informatica, Politecnico di Torino, Torino, Italy*

Abstract

The incessant growth of network virtualization determined the proliferation of Virtual Network Functions (VNFs), software programs that can run on general-purpose servers and that can also integrate security controls for protection from cyber-attacks. However, a high availability of VNFs may be counterproductive for the network administrators who have to select the most suitable ones to establish the security configuration of their network. On the one hand, the vendor-dependent technicalities of each VNF may cloud the security controls it can actually perform. On the other hand, VNF selection traditionally occurs before the synthesis of the virtual network graph, so it does not employ any network information and it may outcome unoptimized results. In light of these shortcomings, this paper proposes a novel security configuration workflow, based on new abstractions that we call projections. They represent the security-related operations that VNFs should perform to enforce a security policy. Thanks to these abstractions, the actual selection of the VNFs can be postponed to the moment their deployment in the physical network is actually required. In fact, projections are enough for the synthesis of the virtual security graph. This paper also proposes a two-step algorithm for computing projection chains as candidate solutions for graph synthesis. The proposed approach has been implemented as a Java framework and a set of tests have validated its applicability to real-world VNFs, correctness, scalability and optimization. These tests showed that the new security configuration workflow can achieve a significant reduction for the number of selected VNFs and their deployment cost. Specifically, in the analyzed scenario, the improvement percentages for these two parameters are 79% and 90% with respect to the worst-case strategy, while 68% and 77% with respect to a traditional more optimized configuration strategy.

Keywords: virtual computer networks, automatic network security, security function selection

1. Introduction

Virtualization has profoundly changed the traditional vision of networking. Agility and dynamism have become decisive factors in enforcing security in modern computer networks by leveraging ideas deriving from *Network Functions Virtualization* (NFV) and *Software-Defined Networking* (SDN). On the one hand, new security functions can be easily deployed on the fly to face an attack as soon as it is detected. On the other hand, state-of-the-art orchestrators can update a function configuration in real time, without stopping it and temporarily pausing the protection it provides. The benefits of these innovations have led to continuous development of virtual functions, which network administrators have at their disposal for enforcing the required security.

Network administrators are commonly in charge of enforcing *Network Security Policies* (NSPs) that describe how the communications should be protected in their networks. For example, NSPs describe which traffic flows

*Corresponding author

Email addresses: daniele.bringhenti@polito.it (Daniele Bringhenti), riccardo.sisto@polito.it (Riccardo Sisto), fulvio.valenza@polito.it (Fulvio Valenza)

must be encrypted, which endpoints can communicate with each other, or which packets must be logged [1]. For each NSP, network administrators must select one or more concrete virtual function implementations, commonly known as *Virtual Network Functions* (VNFs). For example, if there is the requirement to block some traffic flows, they may decide to use iptables, ipfirewall or an alternative VNF. However, the higher the freedom of choice is, the more complex making decisions is. A large number of VNFs is currently available for each security function type, as it is easy for each vendor to release their branded version as software product. Therefore, multiple ones may be suitable to enforce the same security property [2].

1.1. Motivation

The just mentioned high availability of VNFs turns out to be counter-productive in the usual workflow followed by administrators to apply security. Such workflow is composed of three operations [3]: i) selection of a suitable set of VNFs for enforcing the desired NSPs; ii) synthesis of the virtual network graph, by deciding where VNF instances should be placed in the logical network, and by defining their configuration rules; iii) embedding of all the virtual nodes of the logical network, inclusive of the security VNFs, in the physical infrastructure (e.g., a network composed of general-purpose servers).

In this sequence of operations, the first stage, i.e., VNF selection, lays the foundations to satisfy both security and network demands when enforcing user-requested policies. Nevertheless, in the current form, it may satisfy these different requirements only partially, and often in an unsatisfactory way. On the one hand, as the VNFs that are selected are full-fledged software implementations, they contain many technicalities that often cloud the fact that they share the same security operations. On the other hand, as the selection occurs so early in the configuration workflow, it does not consider network information, such as the topology of the network, or the configuration of other network service functions like load balancers, or the characteristics of the servers composing the physical infrastructure.

Without that information, a premature selection of the VNFs, among the several ones that share the same security functionalities but with different implementation technicalities, may produce non optimized results. Two main sub-optimizations that may derive from early VNF selection concern deployment cost and energy efficiency. For example, if there is the requirement to block two different types of traffic (e.g., the web traffic to some domain and the mail traffic to some other domain), an early selection may end up with choosing a distinct VNF to satisfy each requirement (e.g., a web application firewall for the first type and a packet filter for the second one), without realizing that a single VNF instance could provide the security logic to implement both functionalities. Deploying more VNFs than required entails consuming more resources than necessary in the servers where the VNFs are deployed, and more energy for keeping them active. Even though virtualized networks are characterized by high flexibility and operational efficiency, deployment and energy optimizations are still open problems [4].

These issues are even more relevant in virtualized networks based on *Internet-of-Things* (IoT). The multitude and heterogeneity of IoT devices are constantly pushing the size of modern computer networks, characterized by more and more interconnections among those devices [5, 6]. This context magnifies the importance of optimization, whose application has been already discussed in literature to address problems such as service scheduling [7], task planning [8] and attack identification [9]. In contrast, the problem of optimizing the selection of security VNFs and avoiding redundant decisions in the security configuration process has been scarcely investigated at the moment, even though the problem is as relevant as the ones optimization has already been applied to.

1.2. Contributions of the paper

The paper aims to allow optimum VNF selection by proposing a novel security configuration workflow. The objective is to split the two purposes of VNF selection occurring in the traditional configuration workflow, i.e., enforcement of security requirements and physical resource consumption minimization. To achieve it, the novel workflow is based on abstract representations of the NSPs, called projections. The projection of an NSP against a concrete function expresses the security-related operation that the VNF should perform to enforce the NSP, independently of all the specific details of its implementation, such as how the operation is configured in the function and the resources the function requires for its operation. Projections are computed before synthesizing the virtual security graph, and they are used to generate it instead of their concrete VNF counterparts. Then, the actual VNF selection is postponed to be performed jointly with their embedding into the physical infrastructure in a last stage. Thus, only the VNFs that are strictly required are selected and deployed, depending on how the virtual service has been generated to fulfill the NSP projections.

The idea of this novel security configuration workflow, where the VNF selection is postponed with respect to traditional approaches, has been presented preliminarily in [10]. This work improves and completes that preliminary idea with the following main contributions:

(i) This paper enriches the preliminary description of the security configuration workflow, presented in [10], by explaining with greater detail how each step works, and how the different workflow steps must be linked to each other so as to provide a full automated security configuration to a computer network. This description is also paired with a clarifying use case, which guides the readers in understanding the novelty provided by postponing VNF selection to the end of the workflow.

(ii) A complete formalization of the projection abstraction is presented. This formalization captures all the essential information required for the synthesis of the virtual security graph in a compact form, and abstracts from vendor-dependent VNF technicalities.

(iii) The two-step algorithm, in charge of computing all the possible projection chains that should be used to decide how to enforce a user-specified NSP in the virtual network, was only briefly mentioned in [10]. Here, the complete algorithm is formalized. This paper focuses on the initial stage of the proposed workflow, which represents a major novelty in literature. In fact, for the last two stages, our proposal can be easily integrated with already existing approaches, which can be re-used with minor modifications.

(iv) Differently from the first prototype discussed in [10], which only implemented the first stages of the workflow, a new, complete framework has been developed to demonstrate the feasibility of the proposed approach. This new framework includes the interaction with other tools that carry out the remaining operations of the security workflow (i.e., synthesis of the virtual security graph, VNF selection and embedding) in a fully automated way.

(v) An extensive validation of this framework has been carried out on a state-of-the-art computing machine to assess model generality, correctness, scalability and optimization. This validation showed that the choice of postponing VNF selection led to an improvement in terms of number of selected VNFs and their deployment cost. For example, in the analyzed scenario, the improvement percentages for these two parameters are 79% and 90% with respect to the worst-case strategy, while 68% and 77% with respect to a traditional more optimized configuration strategy.

1.3. Organization

The remainder of this paper is structured as follows. Section 2 discusses related work. Section 3 illustrates the novel proposed security configuration workflow, it formalizes the projection abstraction, and it presents the two-step algorithm for projection chain computation. Then, Section 4 describes the PoC implementation of the proposed approach, and it discusses the results of the validation carried out on it. Finally, Section 5 concludes the paper and discusses future work.

2. Related Work

This section discusses the state of the art about security configuration (Subsection 2.1) and VNF abstraction models (Subsection 2.2), underlining their main limitations with respect to our proposal.

2.1. Security configuration

Security configuration has been widely researched in literature, also in relation to the new threats that have been emerging in virtualized networks. This topic is quite broad, as security configuration encompasses multiple operations: VNF selection, synthesis of the virtual security graph (i.e., deciding where the functions should be allocated in the virtual service) and definition of their operational rules. However, for all these operations, almost all the studies focused on concrete VNFs as the starting point for the design of new configuration approaches.

About the two operations of VNF selection and synthesis of the virtual security graph, which are usually challenged jointly in literature, relevant studies are [11], [12], [13], [14] and [15]. Specifically, Scheid et al. [11] propose the k-means clustering algorithm for VNF selection, according to which groups of VNFs are created according to the level of security they can provide. Then, service graphs are generated by selecting VNFs from those groups (e.g., if a high security level is requested in attack detection, an implementation of IDS will be chosen from the VNF cluster labeled with “high security"). Hao et al. [12] employ a composition algorithm based on Trie trees for the synthesis of a security service, in a way that can automatically manage changes of the IP addresses of VNF instances. The objectives of these studies were then extended by the other ones, by pairing them with optimization purposes. On the one hand, Liu et al. [13] employ a greedy iterative algorithm to compose chains of VNFs, with the aim to find a function combination with the highest total throughput and the minimum consumption of physical resources. On the other hand, both Liu et al. [14] and Sendi et al. [15] propose an ILP formulation of the selection problem and a heuristic, so that users can choose either one depending on whether they prioritize optimization or performance. In particular, the heuristic described by Liu et al. [14] is a breadth-first search algorithm, through which function chains are composed to minimize CPU, storage, bandwidth and latency. Instead, the heuristic algorithm proposed by Sendi et al. [15] partitions the network into areas, and for each one of them it solves the selection problem with the aim to provide high scalability. Despite the relevance of these studies, all of them work on concrete implementations of VNFs, and they still adhere to the traditional security configuration workflow. Differently from these studies, the projection abstraction proposed in this paper separates security-related operations from networking parameters and implementation details in the definition of VNF models. As such, the main advantages of the proposed approach are the benefits already discussed in Section 3.1, i.e., better optimization of VNF selection and resource consumption.

About the computation of operational rules, almost all the automatic approaches that have been presented in literature address the configuration of a single type of virtual functions, except the study by Basile et al. [16], which we will analyze later. Specifically, these approaches simply focus on either packet filtering firewalls [17, 18, 19, 20], or

channel protection systems such as VPN gateways [21, 22], or IoT devices [23, 24], or SDN switches [25, 26]. While each of these techniques focuses on a single function type, our vision is to provide a global security configuration, considering several function types.

Instead, as previously mentioned the approach proposed by Basile et al. [16] can automatically configure multiple function categories (packet filter, URL filter, HTTP filter, VPN gateway, anti-virus, anti-malware, content filter, monitoring, anonymity proxy). However, even this technique is limited with respect to our proposed approach. On the one hand, this approach can be applied only to function chains, while our approach can be applied to graph topologies that are more common in modern networks. On the other hand, in that study, VNF selection is carried out according to the usual workflow, while our approach benefits from the postponement of VNF selection.

2.2. VNF abstraction models

The findings derived from this analysis of the state of the art about security configuration are not surprising. After all, in literature, concepts similar to the projection abstraction proposed in our paper have barely been investigated. The only exception is represented by a series of IETF RFC drafts, of which [27] is the most recent one. These RFC drafts propose the *Capability Information Model* (CapIM) as a central component of a more complete architecture, designed to provide standard interfaces for managing VNFs in an efficient manner [28]. This model associates each VNF with a set of capabilities, representing the security controls they can enforce (e.g., packet filtering, detection), in a way that is vendor-neutral and implementation-independent. Such a representation avoids referring to a specific technology or vendor-dependent function when defining a security service, as for the projection abstraction. CapIM represents the foundation for a limited number of studies. Giotis et al. [29] employ the capability model as a means to abstract VNFs, but it just applies it to access control and forwarding virtual functions. Hyun et al. [30] enhance the CapIM-based architecture to make it compliant with the SDN technology. Zarca et al. [31] use a tailored version of the model for the dynamic management of authentication, authorization, and accounting.

At the moment, CapIM (alongside with its aforementioned extensions and customizations) represents the first and last effort in the literature to provide a higher abstraction of security-related operations that can be performed by VNFs. It also has several limitations. Differently from our proposal, the capabilities that can be associated with VNFs are fixed and represent all the possible operations that the VNF might do. Instead, in our vision, a projection is a flexible representation, because it expresses the security-related operation that a VNF should perform to enforce a specific NSP. With this definition, it gets rid of redundant descriptions, and it provides high adaptability to the user-specified policies. Besides, CapIM is almost never used to innovate the security configuration workflow, with the exception of the intent-based technique presented by Basile et al. [16]. However, their approach has limitations that have been already discussed in Subsection 2.1.

In conclusion, the contributions of this paper represent an important step in improving conventional ways for security configuration, opening the path for an additional level of abstraction that is becoming essential to answer the high productivity of software development in network security.

3. The Proposed Approach

This section describes the proposed workflow for improving security configuration (Subsection 3.1), which represents the starting point for designing the projection abstraction concept (Subsection 3.2). It also formalizes the two-step algorithm designed to identify and compute the projection chains that are required to enforce the user-specified NSPs (Subsection 3.3 and Subsection 3.4).

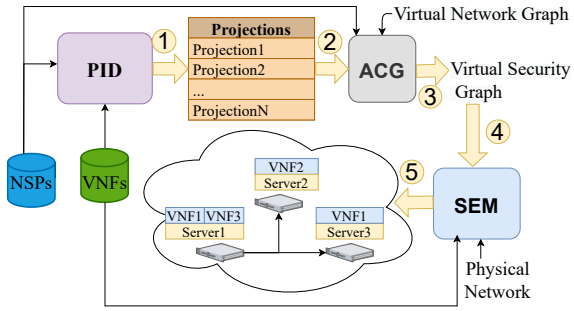


Figure 1: The proposed workflow for security configuration

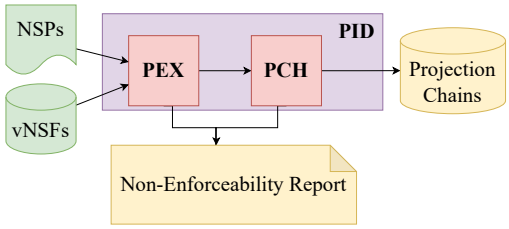


Figure 2: Projection IDentification: the two tasks

3.1. The New Workflow

As reported in [3], where a Security Orchestrator module is introduced in the ETSI NFV Reference Architecture, the traditional workflow for security configuration is composed of three sequential stages. First, the most suitable VNFs are selected for the enforcement of the NSPs in the network. For example, in order to block all the packets having 102.10.3.88 as source IP address and 8080 as destination TCP port, firewalling VNFs as iptables or ModSecurity may be selected. This decision is exclusively dependent on conditions related to security, as it does not consider other factors as network topology. Second, the allocation scheme and configuration for the VNFs in the logical network topology is computed. The logical topology, also called *Virtual Network Graph* (VNG), is an abstract representation of how traffic flows cross the network. This second stage must also compute the configuration rules for each VNF positioned in the VNG, so that they fulfill the original NSPs. The result of this operation is a VNG enriched with security functions and is commonly known as *Virtual Security Graph* (VSG). Third, the VNF embedding on the general-purpose servers composing the physical network infrastructure is established. This final operation is constrained by physical limitations, such as the CPU and RAM availability of the servers.

This traditional workflow can lead to sub-optimizations related to deployment cost and energy efficiency, as already explained in Section 1. Instead, the workflow that we are proposing postpones VNF selection to be done jointly with VNF embedding, with the aim to produce a more optimized result. To achieve this purpose, the workflow has been structured with three stages, as shown in Figure 1: the ***P**rojection **ID**entification* (PID) stage, the ***A**llocation and **C**onfiguration **G**eneration* (ACG) stage and the ***S**election and **EM**bedding* (SEM) stage.

PID – The objective of the PID stage is to identify all the possible ways a set of NSPs may be enforced by using the available VNFs. However, the PID stage does not consist in the common VNF selection. Instead, it employs abstract representations of the NSPs, called projections. A projection consists in mapping the elements composing an NSP (i.e., the actions that must be performed and under which conditions) onto what a VNF can offer to enforce the NSP (i.e., the VNF configuration settings). In this mapping operation of an NSP, all the implementation-dependent technicalities of each VNF against which the NSP is projected are omitted. For example, let us suppose that an NSP specifying that all the traffic going from node with IP address 117.0.3.2 to node with IP address 33.21.1.1 must be blocked is mapped onto iptables. The result projection would be the following one: "({IPSrc = 117.0.3.2, IPDst = 33.21.1.1}, {deny})" This projection is expressed with a vendor-independent formulation, as it does not involve the specific commands to set up the actual iptables configuration.The formalization of the projection abstraction will be detailed in Subsection 3.2.

Starting from the VNF manifests and the NSPs, the projections can be computed with a two-step algorithm, called ***P**rojection **ID**entification* (PID). The two steps are executed sequentially as also shown in Figure 2. The first

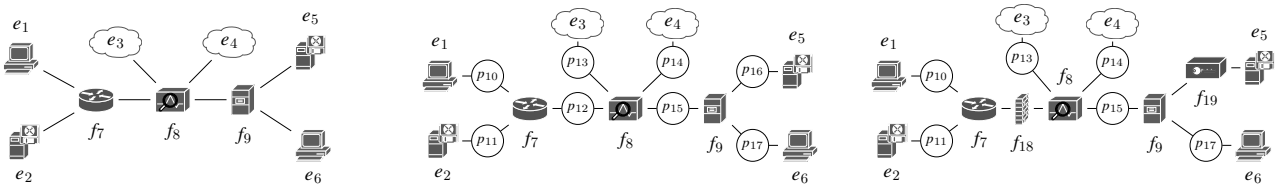
step directly works on the input NSPs and VNFs to compute the corresponding projections, whereas the second one combines the projections into chains that can fully enforce all the NSPs. As shown in Figure 2, if the algorithm fails, it produces a non-enforceability report with the reason of the failure.

In greater detail, the first step of the algorithm is to compute the corresponding projection for each pair composed of a VNF and an NSP. In doing so, it employs some optimizations. On the one hand, it may happen that the projections of the same NSP against different VNFs are identical. For example, the result of projecting an NSP against two packet filter implementations as iptables and ipfirewall would be the same, as they would share the same manifest. In that case, a single instance of that projection is simply used, thus avoiding useless redundancy. On the other hand, it may occur that a projection cannot be defined for a pair composed of a VNF and an NSP, e.g., when a VNF does not have any configuration field corresponding to the ones specified in the conditions of an NSP. The consequence is that the VNF is excluded for the NSP enforcement, and will not be used by the next stages of the workflow. This step, called ***P**rojection **EX**traction* (PEX), will be detailed in Subsection 3.3.

The second step of the algorithm is to combine the computed projections into chains that can fully enforce all the actions requested by the NSP they derive from. In fact, mapping an NSP onto the manifest of a VNF may result into a projection that contains only partial information of the original NSP. In case no valid chain is identified for an NSP, the global process immediately halts. An early non-enforceability report is also produced, notifying the network administrator why the projection identification failed. Otherwise, the valid combinations are passed on to the next stage of the process, for the synthesis of the virtual security graph. In this way, the security configuration will be performed in a way that is agnostic to the VNF implementation. This step, called ***P**rojection **CH**aining* (PCH), will be detailed in Subsection 3.4.

The need of this second step can also be explained with an example. Let us suppose that a network administrator can use two VPN gateway VNFs (Strongswan and OpenConnect) and a intrusion detection VNF (Suricata) to enforce an NSP requesting that, when a packet from 127.0.3.4 to 45.66.10.2 crosses the network of the administrator, it must be encrypted and a notification must be produced: "({IPSrc = 127.0.3.4, IPDst = 45.66.10.2}, {encrypt, alert})". If the NSP is projected against each of the two VPN gateways, the resulting projection maintains only the information of the encryption operation of the original policy, as the used VNFs cannot work as intrusion detection systems: "({IPSrc = 127.0.3.4, IPDst = 45.66.10.2}, {encrypt})". Nonetheless, the projection is the same for both VNFs, as they can offer the same security-related operations. Consequently, a single instance of that projection can be used in the security workflow, thus avoiding useless redundancy. Similarly, if the NSP is projected against the intrusion detection system, the resulting projection maintains only the information of the alerting operation: "({IPSrc = 127.0.3.4, IPDst = 45.66.10.2}, {alert})". In this example, at the end of the identification of the NSP projections, none of them is a full representation of the original NSP. Therefore, in this case, the PID stage must combine them into a projection chain where all the actions requested by the original NSP are supported. However, there may be cases where a single projection still contains all the original information, depending on the features of the VNF against which the NSP is projected.

ACG - The objective of the ACG stage is to synthesize the VSG so that for each NSP a projection combination that represents it is fully enforced. It works on the logical level of the virtual network, which provides information about the network topology and the configuration of the functions composing the service (e.g., NATs, load balancers). The VNG is internally represented in the ACG stage as a *Virtual Allocation Graph* (VAG). A VAG is a



(a) Virtual Network Graph (b) Virtual Allocation Graph (c) Virtual Security Graph

Figure 3: Function graphs employed in the ACG stage

graph derived from the VNG so that, for each link connecting a pair of nodes of the original VNG, a node named *Allocation Place* (AP) is included in the corresponding position of the VAG. An AP is a placeholder position where the ACG stage may decide to allocate an implementation-independent representation of the function configurations, called *functionality*, which may be needed to enforce at least a projection. For example, the VNG depicted in Figure 3a is represented as the VAG depicted in Figure 3b. The solution that the ACG stage computes is a VSG, where some APs are filled with the computed functionalities. An example of VSG is depicted in Figure 3c, where f_{18} and f_{19} are functionalities introduced in the VAG.

The creation of the functionalities from a corresponding projection requires a refinement operation, because the projections were computed in the PID stage overlooking information related to the VNG. The need of refinement is motivated by two reasons. First, not always a one-to-one relationship exists between input projections and output functionalities. For example, let us suppose that a projection requires to block all traffic flows coming from 154.66.0.2 to any IP address: "({IPSrc = 154.66.0.2}, {deny})". If these traffic flows do not have any common AP in their paths, multiple APs must be used by the ACG stage. In each one of them, a functionality that is equal to the projection is allocated. This shows how it is possible that multiple functionalities, derived from the same projection, are needed to enforce an NSP. Second, even though the functionality has an abstract representation that resembles the projection, the content is different because it must takes into account how traffic may be altered by the VNG configuration. For example, let us suppose that all the packets going from the source port 88 of 123.4.5.6 to 44.5.6.7 must be logged: "({IPSrc = 123.4.5.6, IPDst = 44.5.6.7, pSrc = 1337}, {log})". It may happen that the only APs where this projection may be fulfilled are after a NAT, with a configuration which establishes that the source IP address each packet having 123.4.5.6 as original value for that field must be modified into 44.5.9.2. Consequently, a functionality that is generated and allocated in that AP would have the following configuration: "({IPSrc = 44.5.9.2, IPDst = 44.5.6.7, pSrc = 1337}, {log})". Therefore, this example shows that the fields of a functionality may have different values with respect to the ones of the corresponding projection.

In summary, the main difference with respect to the VSG synthesis of the traditional workflows is that the VSG is not composed of VNFs, but of an abstract representation of their configuration (i.e., the functionalities derived from the projections). However, refinement methods that have been already proposed in literature for specific types of VNFs may be reused in the ACG stage of this novel workflow with minor adjustments. Studies were proposed in literature about packet filtering firewalls (e.g., [32], [33], [19], [20]) and VPN gateways (e.g., [34], [21]). Inspired from them, in [10] we specifically proposed to formulate the functionality allocation and configuration problem as a Maximum Satisfiability Modulo Theories problem in order to achieve both a-priori formal correctness and optimization. This problem formulation is specifically thought to work with our proposed PID stages, but the other existing studies may be extended to be compliant with them.

Table 1: VNFs available in the use case

VNF	VNF Type	Deployment Cost
v_1	packet filter firewall	3
v_2	packet filter + web-application firewall	5
v_3	packet filter + web-application firewall + logger	6
v_4	intrusion detection system	5
v_5	logger	3
v_6	anti-spam filter	4

Table 2: NSPs defined for the use case

NSP	NSP Requirement	Projection Set
n_1	block "youtube.com" from 124.10.0.0/16 (e_4)	$\{p_{1,1}, p_{2,1}, p_{3,1}\}$
n_2	block traffic from 124.10.0.2 (in e_4) to 87.4.2.3 (e_2)	$\{p_{1,2}, p_{2,2}, p_{3,2}\}$
n_3	log traffic from 87.4.2.3 (e_2)	$\{p_{3,3}, p_{5,3}\}$
n_4	detect traffic anomalies from 98.8.8.0/24 (e_3)	$\{p_{4,4}\}$
n_5	block "linkedin.com" from 42.0.10.4/24 (e_5)	$\{p_{1,5}, p_{2,5}, p_{3,5}\}$
n_6	block traffic from 42.0.10.4/24 (e_5) to 42.0.10.8/24 (e_6)	$\{p_{1,6}, p_{2,6}, p_{3,6}\}$

SEM - The ACG does not provide a clear indication on which VNFs must be used for the NSP enforcement or on which servers they should be deployed. Therefore, the objective of the final stage, the SEM stage, is dual. The first one is to select the best subset of VNFs, among the available ones, that supports the functionalities allocated and configured in the ACG stage. The second objective is to compute the deployment scheme of the selected VNFs, i.e., to decide on which node of the physical network each VNF should be installed. Traditionally, these two objectives are achieved separately. In literature, some studies (e.g., [35], [36], [37]) propose smart algorithms for identifying the optimal set of VNFs to enforce security intents, while other ones (e.g., [38], [39], [40], [41]) exhaustively researched the problem of virtual network embedding. On the one hand, methods that have been proposed for different objectives can be paired and sequentially executed. On the other hand, a single method can be formulated. For example, in [10] we propose the idea to formulate the selection and embedding problem as a multi-objective *Integer Linear Programming* (ILP) problem. If there are hundreds of available VNFs that implement the same functionalities, then a manual decision would have been troublesome and error-prone to be taken, even overlooking all the other degrees of freedom (e.g., number of physical hosts, etc.).

As anticipated before, the stage that will be formalized in the reminder of the paper is the PID stage, as for that one no existing approaches can be leveraged at the moment.

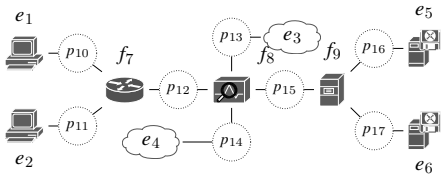


Figure 4: VAG employed in the use case

3.1.1. *Clarifying use case*

The definition of the projection concept for abstracting the security-related operation that a VNF should perform to enforce an NSP and the reorganization of the full configuration workflow bring manifold advantages. These benefits are illustrated by means of a clarifying use case, where the novel proposed approach is applied and compared to the traditional security configuration workflow.

In this example, Table 1 lists the VNFs that are available to deploy in the physical infrastructure, it briefly describes their function type, and it shows their deployment cost, expressed in numeric format. This cost representation is a simplification of the real deployment costs, that are commonly known by the network administrators and that derive from multiple parameters (characteristics of the servers, energy efficiency, etc.). Table 2 presents the NSPs that must be enforced in the network, providing a short description of the traffic that is object of the corresponding policy action. This table also compactly shows against which VNFs each NSP can be successfully projected ($p_{i,j}$ is the projection of n_i against the manifest of v_j). Figure 4 represents the VAG derived from a network topology that is a simplified version of a real one, i.e., the network of our University Department.

A first advantage of the novel security configuration

workflow is that it prevents redundant solutions. Thanks to the projection abstraction, the VNF selection occurs after knowing which projections of the original policies are needed to enforce them and how the functionalities associated with such policies have been allocated in the virtual graph. With the traditional workflow, instead, the administrators are commonly prone to making redundant and unoptimized decisions. For example, they may select three VNFs – the packet filter v_1, the web-application firewall v_2 and the logger v_5 – to enforce n_1, n_2 and n_3 respectively. However, in this case, a single VNF is enough, because the traffic flows related to these three NSPs converge to the same AP of the VAG, i.e., p_{12}. The novel workflow can produce the minimum allocation scheme of functionalities, so avoiding redundancies, which in this case means selecting just v_3 instead of the three VNFs v_1, v_2 and v_3. In terms of deployment cost, this solution costs only 6, instead of 11.

A second advantage is that this workflow more easily leads to the selection of VNFs with a lower cost, when more than one offer the functionalities necessary to enforce the NSPs. For example, the NSPs n_5 and n_6 can be enforced by both a packet filter and a web-application firewall. Intuitively, the deployment cost of a packet filter is less than the one for a web-application firewall. However, a packet filter may be selected only under specific circumstances, e.g., when an AP where the firewall may be allocated is only crossed by flows related to the NSPs n_5 and n_6. These cases are difficulty identified with the traditional workflow. There, the administrator usually decides to select the web-application firewall, as it is function that works at the highest level of the ISO/OSI stack and thus guarantees the satisfaction of both the NSPs. Instead, if VNF selection occurs after the VSG synthesis, it is already clear where the functionalities derived from the projections are allocated and which traffic flows they must block or allow. This allows choosing a packet filter whenever it is possible.

Finally, security administrators usually struggle in selecting the most appropriate VNF for the enforcement of security policies, even among the VNFs that perform similar operations. Each VNF is a different implementation, developed by a different developer team. Therefore, it has different configuration languages, networking parameters, or performance. A VNF that can decide if a packet must be dropped on the basis of its 5-tuple may require a more complex technical jargon for writing the filtering rules, another one may require more CPU and RAM for the set-up in the physical network, another one may take more time for being installed and be ready to filter network packets. All these networking and performance factors are important, but they do not directly influence the outcome of decisions concerning security. If abstract representations of NSPs and VNF configurations (i.e., respectively, the projection and the functionality) may temporarily discard that information and allow using it in the next stages of the orchestration (e.g., during the VNF deployment), security decisions can be taken more easily, quickly and with better results.

3.2. The Projection Abstraction

The definition of the projection abstraction requires modeling two basic elements: the VNFs that can be instantiated in the network and the NSPs that must be enforced by them.

3.2.1. VNF Model

Each VNF model is characterized by configuration fields that define the security properties it can enforce (e.g., the conditions expressing the layer of the ISO/OSI stack where the VNF can work, the algorithms it can execute, or the actions it can perform on the traffic). With the objective to provide a comprehensive view on all the parameters characterizing a VNF, they are grouped into a single representation, called *VNF manifest*.

For a VNF v, the corresponding manifest M_v is composed of two sets, i.e., $M_v = (F_v, A_v)$:

- F_v is the set of all the features for which the VNF can take a decision and/or which can be configured on it. This set includes packet fields (e.g., source and destination IP addresses, web-application fields as domain or url) and other configuration elements that determine the working modes of the VNF (e.g., the encryption algorithm and the key length if a VNF is a VPN gateway);
- A_v is the set of all the actions that the VNF can enforce.

In turn, the field set F_v is organized into two subsets, i.e., $F_v = (F_v^+, F_v^*)$, because it is important to discriminate the fields that a VNF can configure on itself from those it can only use to take decisions:

- F_v^+ is the set of all the features for which the VNF can take a decision and which it can configure (e.g., for a packet filtering firewall such as iptables, all the fields of the IP 5-tuple belong to this set, because the configuration rules are composed of conditions based on IP addresses, ports and transport-level protocol);
- F_v^* is the set of all the features for which a VNF can take a decision, but without configuring them, i.e., by configuring other fields which may allow to reach the same security property (e.g., if a specific web domain must be blocked, iptables might be used, however it cannot configure a "domain" field, but only a corresponding IP address).

Below, three examples are presented to clarify the concept of VNF manifest. In these manifests, only a subset of all the fields that are present in the F_v^+ and F_v^* sets are reported for the sake of conciseness.

$$\begin{array}{|l|}\hline \text{VNF } v_1\text{: iptables} \\ F_{v_1}^+ = \{\text{IPSrc}, \text{IPDst}, \text{pSrc}, \text{pDst}, \text{tProto}\} \\ F_{v_1}^* = \{\text{domain}, \text{url}, \text{mailAddress}, \text{payload}, \ldots\} \\ A_{v_1} = \{\text{allow}, \text{deny}\} \\ \hline \end{array} \tag{1}$$

$$\begin{array}{|l|}\hline \text{VNF } v_2\text{: Squid} \\ F_{v_2}^+ = \{\text{IPSrc}, \text{IPDst}, \text{pSrc}, \text{pDst}, \text{tProto}, \text{domain}, \text{url}, \ldots\} \\ F_{v_2}^* = \{\text{mailAddress}, \text{payload}, \ldots\} \\ A_{v_2} = \{\text{allow}, \text{deny}, \text{log}\} \\ \hline \end{array} \tag{2}$$

$$\begin{array}{|l|}\hline \text{VNF } v_3\text{: MyLogger} \\ F_{v_3}^+ = \{\text{domain}, \text{url}\} \\ F_{v_3}^* = \{\text{mailAddress}, \text{payload}, \ldots\} \\ A_{v_3} = \{\text{allow}, \text{log}, \text{alert}\} \\ \hline \end{array} \tag{3}$$

The manifest of a packet filtering VNF such as iptables, ipfirewall or equivalent firewall implementations, which can only work at layers 3 and 4 of the ISO/OSI stack, is shown in (1). These VNFs can decide if a received packet should be allowed to be forwarded to the next hop or denied depending on the values of the IP 5-tuple. However, this does not mean that a packet filtering firewall cannot take decisions for packets having fields such as web domain and url.

Instead, web application firewalls such as Squid have a manifest similar to the one presented in (2). With respect to a packet filter, this type of firewall can also configure rules based on web domains, urls, HTTP methods (e.g., POST, GET), Content-Type, etc. All the other fields which were present in $F_{v_1}^*$ are in $F_{v_2}^*$ as well, since Squid is a firewall as iptables, but it simply works on a different level. Nonetheless, both of them do not have in their $F_{v_1}^*$ and in $F_{v_2}^*$ sets any parameter related to encryption (e.g., algorithm, encryption key length).

Finally, the virtual functions that can be used to enforce some security properties do not have to be well-known implementations such as iptables or Squid, but they can be software programs developed by any developer, running on a Virtual Machine or Docker. As it is possible to see from (3), the manifest description is flexible enough to support also this type of functions. In this example, the VNF that has been developed and is available for the network administrator is named "MyLogger". It cannot block

packets, but it can only log the receiving of specific kinds of traffic and notifying the network administrator about that event. Additionally, it has been developed in such a way that the only fields which are present in the configuration rules are web domain and url. Therefore, the fields of the IP 5-tuple itself are absent from the F^{+}_{v3} set. They are not in the F^{*}_{v3} set either, because domain and url are higher level information.

3.2.2. NSP Model

An NSP n is modeled as $n = (C_n, S_n)$:

- C_n expresses the conditions that determine the traffic on which the policy actions must be applied;
- S_n expresses the actions that must be applied and the enforcement modes (e.g., the packet fields on which an action must be applied, or the algorithm to be used).

C_n is an (unordered) set, and it can be represented as $C_n = \{c_1, c_2, ..., c_m\}$. Each $c \in C_n$ is defined over a field f, that can be accessed with the $c.f$ notation. The condition can specify a single value for the field (e.g., IPSrc = 10.0.0.1), a range of values (e.g., pSrc = [80-100]) or the special symbol $*$, meaning that each possible value that can be assigned to that field is valid (e.g., domain = $*$).

S_n can be a set (i.e., an unordered collection) $\{s_1, s_2, ..., s_l\}$ or a list without repetitions (i.e., an ordered collection) $[s_1, s_2, ..., s_l]$. Each $s \in S_n$ is modeled as (a_s, B_s), where:

- a_s is the action that must be enforced (e.g., block, encrypt);
- B_s is a set of bindings "field – (optional) value", specifying additional information about how the action must be enforced (e.g., the binding "IPSrc = 20.1.2.4" might specify how the source IP address must be changed by a network address translator, whereas "algorithm = AES-128-CBC" might specify the encryption algorithm a VPN gateway must use to provide confidentiality). If no binding is specified (e.g., when the action is applied on the whole packet satisfying the conditions), $B_s = \emptyset$.

The actions in S_n can be optionally grouped into multiple subsets K_1, K_2, ..., K_r, ..., K_p, where each subset must contain at least two actions. If two or more actions belong to K_r, that means they must be enforced by the same VNF. This formalization is introduced to support the cases where the actions cannot be managed by different VNFs. For example, a network administrator may require that all the packets satisfying certain conditions are logged and blocked by the same VNF, because they are dangerous and must be discarded as soon as possible avoiding any further hop.

An example of NSP n is shown in (4). This policy requires that, for each packet satisfying all the conditions, firstly its source and destination IP addresses are logged, and then the whole packet is blocked so that it cannot reach the destination.

$$\boxed{\begin{array}{l} \text{NSP } n \\ C_n = \{\text{IPSrc} = 125.10.2.0/24, \text{IPDst} = 20.20.20.1, \text{pSrc} = *, \\ \qquad \text{pDst} = 80, \text{tProto} = \text{TCP}, \text{domain} = \text{dangerousSite.com}\} \\ S_n = [(\text{log}, \{\text{IPSrc}, \text{IPDst}\}), (\text{deny}, \emptyset)] \end{array}} \tag{4}$$

The proposed VNF and NSP models are general enough to support both real-world concrete implementations of security functions and the policies that a network administrator may really request. In fact, their actual generality has been validated with some tests, which will be presented in Section 4.

3.2.3. Projection Model

A projection p represents the security-related operation that a VNF v should perform to enforce an NSP n. A projection p consists in mapping the elements composing an NSP n (i.e., the actions that must be performed and the conditions under which the NSP must be fulfilled)

onto what a VNF v can offer to enforce the NSP (i.e., the VNF configuration settings). In this mapping operation of an NSP, all the implementation-dependent technicalities of each VNF against which the NSP is projected are omitted. Consequently, if the same NSP is projected against different VNFs that fulfill the same security objectives, the resulting projections are presumably the same.

As a projection directly derives from an NSP, it is modeled similarly. Specifically, p is modeled as $p = (C_p, S_p)$ where:

- C_p expresses the conditions that determine the traffic on which the corresponding actions must be applied;
- S_p expresses the actions that the VNF against which the NSP is projected can enforce to fulfill it.

Here we intuitively show how a projection is expressed with an example. Considering the three VNFs v_1, v_2 and v_3 whose manifests have been presented in (1), (2) and (3), the projection deriving from mapping the policy n presented in (4) onto those manifests are:

$$\boxed{\begin{array}{l}\text{Projection } p_1\text{, derived by mapping the NSP } n \text{ onto the VNF } v_1\\ C_{p_1} = \{\text{IPSrc} = 125.10.2.0/24, \text{IPDst} = 20.20.20.1, \text{pSrc} = *,\\ \qquad \text{pDst} = 80, \text{tProto} = \text{TCP}\}\\ S_{p_1} = [(\text{deny}, \emptyset)]\end{array}} \tag{5}$$

$$\boxed{\begin{array}{l}\text{Projection } p_2\text{, derived by mapping the NSP } n \text{ onto the VNF } v_2\\ C_{p_2} = \{\text{IPSrc} = 125.10.2.0/24, \text{IPDst} = 20.20.20.1, \text{pSrc} = *,\\ \qquad \text{pDst} = 80, \text{tProto} = \text{TCP}, \text{domain} = \text{dangerousSite.com}\}\\ S_{p_2} = [(\text{log}, \{\text{IPSrc}, \text{IPDst}\}), (\text{deny}, \emptyset)]\end{array}} \tag{6}$$

$$\boxed{\begin{array}{l}\text{Projection } p_3\text{, derived by mapping the NSP } n \text{ onto the VNF } v_3\\ C_{p_3} = \{\text{domain} = \text{dangerousSite.com}\}\\ S_{p_3} = [(\text{log}, \{\text{IPSrc}, \text{IPDst}\})]\end{array}} \tag{7}$$

3.3. PEX: Projection EXtraction

The Projection EXtraction (PEX) operation aims to compute the projection of an NSP against a VNF, if it exists. Algorithm 1 has been designed for accomplishing this goal.

First, given a policy n and the manifest m_v of a VNF v, the condition set C_p of the corresponding functionality $p_{v,n}$ is computed (lines 1-9). For each policy condition $c \in C_n$ based on a field f, the manifest of the VNF should include that field in the F_V^+ set or in the F_V^*. In the former case, the condition c simply becomes a condition of the projection as well (line 5), as the VNF can configure that field with specific values. In the latter, a new condition $f = *$ is created and included in the condition set of the projection, because the VNF can only take decisions regarding that field, but it cannot configure it (line 7). If the NSF manifest does not support any condition field of the policy, the resulting condition set of the projection remains empty. The algorithm immediately stops, and an early non-enforceability report is produced to inform the user about this event (line 9).

Second, the action set of the projection is computed (lines 10-25). Differently from the condition case, it is not enough that a policy action $s \in S_n$ is included in the action set A_v of the VNF manifest (line 12). It is possible that the action s must be enforced on some fields and parameters (e.g., the packet source address must be modified). All the fields on which the action s is applied must belong to the F_V^+ set of the VNF, because the function must be able to directly operate on those fields (line 15). If one of them is not supported, then the action cannot be part of the output projection.

The creation of the action set requires an additional check with respect to the condition set. In the policy specification, the user may have requested that two or more actions must be necessarily applied by the same function (e.g., a packet satisfying certain conditions must be logged and then discarded avoiding any other hop in the

Algorithm 1 computation of p_{vn}

Input: a policy n, a VNF manifest m_v
Output: $p_{v,n}$

1: $C_p \leftarrow \emptyset$ ▹ Creation of the condition set
2: **for each** $c \in C_n$ **do**
3: **if** $\exists f \in F_v \mid f = c.f$ **then**
4: **if** $f \in F_v^+$ **then**
5: $C_p \leftarrow C_p + \{c\}$
6: **else**
7: $C_p \leftarrow C_p + \{f = *\}$
8: **if** $C_p = \emptyset$ **then**
9: **exit**(no field is supported)
10: $S_p \leftarrow \emptyset$ ▹ Creation of the action set
11: **for each** $s \in S_n$ **do**
12: **if** $s.a_s \in A_v$ **then**
13: supported$(s.a_s) \leftarrow$ true
14: **for each** $b \in s.B_s$ **do**
15: **if** $b.f \notin F_v^+$ **then**
16: supported$(s.a_s) \leftarrow$ false
17: **if** supported$(s.a_s) =$ true **then**
18: $S_p \leftarrow S_p + \{s\}$
19: **for each** $s \in S_p$ **do**
20: **for each** $s' \in S_n$ **do**
21: **if** $s \neq s' \wedge s' \notin S_p \wedge (\exists K_l | s, s' \in K_l)$ **then**
22: $S_p \leftarrow S_p \setminus \{s\}$
23: **break**
24: **if** $S_p = \emptyset$ **then**
25: **exit**(no action is supported)
26: **return** $p_{v,n} = (C_p, S_p)$

network). Therefore, either all those actions are included in the action set of the projection, or none of them. After generating the A_v set as previously described, if the algorithm notices that only a subset of actions that should be applied by the same function is present in it, they are removed (lines 19-23). At that point, if the produced action set is empty, that would again trigger the generation of an early non-enforceability report to the user (lines 23-24). Otherwise, the projection is finally produced with the computed condition and action sets (line 26).

The worst-case time complexity of Algorithm 1, used for the computation of $p_{v,n} = (C_p, S_p)$, can be estimated as the sum of the time complexities of three sequential code blocks. Lines 1-9 have $O(|C_p|)$ complexity, because $O(1)$ operations are performed on each element of C_p. Lines 10-18 have $O(|S_n| \cdot \max_{s \in S_n}(s.B_s))$ complexity because that code block consists of two nested loops. The external one iterates on each element of S_n, whereas the internal one requires a number of iterations that in the worst case is

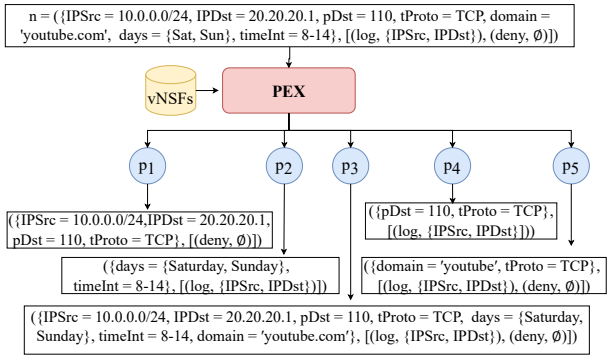


Figure 5: Projection EXtraction: a visual example

equal to the cardinality of the largest $s.B_s$ set, with $s \in S_n$. Lines 19-26 have $O(|S_n|^2)$ complexity, because also that code block is made of two nested loops, both iterating on the S_n set. Summing up, the overall worst-case time complexity for the computation of $p_{v,n}$ is $O(|C_p| + |S_n| \cdot \max_{s \in S_n}(s.B_s) + |S_n|^2)$. However, by considering the NSPs that are commonly defined in studies related to security policy refinement [1][16], the condition set commonly is much larger than the action set (e.g., just by imposing conditions on the IP 5-tuple, five conditions are included in the C_p set). Therefore, the dominant term among those appearing in the notation of asymptotic complexity is $|C_p|$.

A visual example is shown in Figure 5. Each projection denoted by the p symbol derives from mapping policy n against a different VNF manifest. For instance, p_1 derives from a simple packet filter that cannot manage fields related to domain or time interval, p_3 derives from a VNF that may fully enforce the NSP, and p_4 from a VNF that can only log the IP 5-tuple of the received traffic.

Given an NSP, the algorithm is repeated for each VNF that is available. The resulting projections may not contain all the information of the original NSP, e.g., they may support a partial set of all the actions requested by the NSP. Therefore, the PEX task is not sufficient, but a projection chaining operation is still required.

3.4. PCH: Projection Chaining

The Projection CHaining (PCH) operation aims to compute all the possible chains of the projections output by the

Table 3: Symbol table

Symbol	Explanation
n	network security policy that must be enforced
S_n	action set of n
s_i	i-th action in S_n, with $i = 1, ..., \|S_n\|$
K_l	h-th subset of actions of S_n, including the same function should be in charge of, with $l = 1, ..., m$
a_{s_i}	operation of i-th action
B_{s_i}	enforcement mode set of i-th action
b_{ih}	h-th enforcement mode of i-th action, with $h = 1, ..., \|B_{s_h}\|$
P_v	projection set
p_j	j-th projection of P_v, with $j = 1, 2, ..., \|P_v\|$
x_{ij}	binary variable, whose value is set to 1 by the solver if the j-th projection is chosen as responsible for the i-th action of p, otherwise it is set to 0
y_{ij}	binary variable, whose value is set to 1 before launching the solver if the j-th projection supports the i-th action, otherwise it is set to 0
z_{ijh}	binary variable, whose value is set to 1 before launching the solver if the j-th projection supports the i-th action with h-th enforcement mode of p, otherwise it is set to 0

PEX operation. As the PID stage is topology-independent and it works on each policy independently from the other ones, it cannot decide if a chain is more suitable than the others.

The problem of finding the projection chains has been formulated as an *Enumeration Problem* (EP) over a set of *Constraint Satisfaction Problem* (CSP)-like formulas. A chain is a solution for the EP if it contains a projection responsible for each action of the original NSP. Among all the projections that support a certain NSP action, the solver chooses a single one as responsible with the aim to avoid redundancy. Equations (8)-(11) represent the problem constraints, and Table 3 describes the symbols used for their formulation. Among them, the output binary variable appearing in the formulas, x_{ij}, expresses if a certain projection p_j is chosen as responsible for action s_i (when $x_{ij} = 1$) or this task is assigned to another projection (when $x_{ij} = 0$). If S_n is a set, the assignment of index i to each policy action is random. Instead, if S_n is a list, the assignment naturally follows the ordering of the actions in it, so that index 1 is assigned to the first action, and index $|S_n|$ is assigned to the last one.

$$\sum_{j=1}^{|P_v|} x_{ij} = 1, \forall i = 1, ..., |S_n| \tag{8}$$

$$x_{ij} \leq y_{ij}, \forall i = 1, ..., |S_n|, \forall j = 1, ..., |P_v| \tag{9}$$

$$|B_{s_i}| \cdot x_{ij} \leq \sum_{h=1}^{|B_{s_i}|} z_{ijh}, \forall i = 1, ..., |S_n|, \forall j = 1, ..., |P_v| \tag{10}$$

$$|K_l| \cdot x_{ij} \geq \sum_{i' | s_{i'j} \in K_l} x_{i'j}, \forall i = 1, .., |S_n|, \forall j = 1, .., |P_v| \tag{11}$$

The four quantified CSP-like formulas can be explained as follows:

1. According to formula (8), one and only one projection p_j is responsible for action s_i. Even though multiple projections may fulfill this task, in each enumerated solution only one is chosen.

2. According to formula (9), a projection p_j can be chosen as responsible for action s_i only if it supports the operation a_{s_i}, i.e., if $\exists s_k \in S_p$ such that $a_{s_k} = a_{s_i}$. This constraint is required as the PEX operation may have extracted projection that only partially support the actions required by the corresponding NSP.

3. According to formula (10), a projection p_j can be chosen as responsible for action s_i only if it supports all the enforcement modes defined in B_{s_i}. This constraint is included in the formulation of the EP problem only if $B_{s_i} \neq \emptyset$, otherwise constraint (9) is enough as condition of choice.

4. According to formula (11), if a projection p_j is chosen as responsible for action s_i and if s_i belongs to a set K_l of actions that must be applied by the same projection, then p_j must be responsible also for all the other actions in K_l as well. This constraint is included in the formulation of the EP problem only if there is the specification of at least a K_l set.

Each assignment for the x_{ij} variables represents a possible projection chain, as $x_{ij} = 1$ implies that the i-th action requires an instance of the j-th projection for its enforcement. However, the solution set computed by solving this

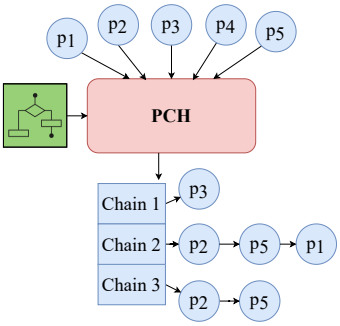


Figure 6: Projection CHaining: a visual example

EP problem may not be complete, and two post-processing operations may be required under specific circumstances.

First, if S_n is an (unordered) set, the assignment of index i to each policy action does not follow any strict ordering guideline. Therefore, only a possible permutation of the actions out of the $|S_n|$ possible ones is established. This deficiency is easily overcome, by computing all the other permutations.

Second, it may happen that two variables having consecutive values for the i index have the same j index, i.e., the actions require the same projections to be enforced. On the one hand, if the two actions are not part of a K_l set, then either a single instance or a pair of instances of the j-th projection may be used to enforce those actions. Therefore, both solutions must be derived from the single assignment computed by solving the EP problem. On the other hand, if the two actions are part of a K_l set, then the only possible solution is that one where a single projection is used.

A visual example of the PCH operation is shown in Figure 6. The projections that are input to the EP characterizing this operation are the same ones that were presented in Figure 5. They are combined in three different chains, which can enforce all the actions of the requested NSP n. As it can be seen, Chain 1 is composed of a single projection, p_3, because it can perform both the requested operations, i.e., logging the source and destination IP addresses of the packets identified by the policy conditions, and then block those packets from reaching their destination. Instead, the other chains require more projections, because each one of them is not enough to enforce all the actions.

After the completion of the post-processing operations, the computation of the VSG starting from the projection chains is left to the ACG stage. As explained in Section 3.1, this stage works on information coming from the network, and having complete visibility on all the requested policies, so it is a complex operation by itself. Nonetheless, the way it can take decisions is simplified by the fact that the possible projection chains have been already computed, and they can be represented as constants in the definition of the problem instead of open variables.

4. Implementation and validation

As this paper focuses on the first stage of the proposed security configuration workflow, this section describes how the models defined for the projection abstraction and the algorithms designed for their computation have been implemented and validated.

4.1. Implementation

The PID stage of the security configuration workflow has been implemented as a Java framework. The code is publicly available in the GitHub repository at the following link: https://github.com/netgroup-polito/verefuse.

The input VNF manifests and the NSPs 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 automatically computed projection chains. The framework exposes a set of REST APIs, so that it can interact with the user or with other applications. As both the PEX and PCH tasks can work on each NSP independently from the others, the code of both has been parallelized. The implementation allows the user to specify the number of threads which must be used in the execution of the program. If the user does not specify the thread number, then eight threads are used by default.

Besides, the formulation and resolution of the enumeration problem of the PCH task are internally managed by employing the mathematical programming solver Gurobi[1] (version 8.1.1). This solver can work on different types of problems, e.g., linear programming, mixed-integer linear programming, quadratic programming. Gurobi offers simple APIs in multiple high-level programming languages, including Java, which have been used in the PID's implementation for the definition of the problem constraints and the resolution.

4.2. Validation

4.2.1. Experimental setup and validation objectives

The experimental setup used for the framework validation consists in a machine with an Intel i7-6700 CPU running at 3.40 GHz and 32GB of RAM.

This setup has been used to fulfill four validation objectives, whose aims are to check:

- the generality of the models defined for the VNF manifests (section 4.2.2);
- the correctness of the algorithms employed for the projection extraction and chaining (section 4.2.3);
- the scalability of the PID stage and its superiority with respect to the state of the art solutions (section 4.2.4);
- the optimization provided by the proposed security configuration workflow (section 4.2.5).

4.2.2. Validation of model generality

We have analyzed 30 VNFs that are currently available for enforcing security requirements, and we have tried to model their manifests. Among the considered VNFs, there are packet filtering firewalls (iptables, ipfirewalls, nftables, PfSense), web-application firewalls (ModSecurity, IronBee, NAXSI, WebKnight), anti-spam filters (SpamAssassin, MailCleaner, Rspamd), VPN gateways (Strongswan, Openswan, SoftEther, OpenConnect), intrusion detection systems (Suricata, Snort, Zeek). For each VNF, we have identified the actions that can be performed (i.e., the A set) and the fields that can be configured (i.e., the F^+ set) by carefully analyzing their configuration guides and examples. Then, we have identified the fields for which the VNFs may take decision but they cannot be configured (i.e., the F^* set) by reasoning about what security properties each VNF can enforce, also by referring to the classical ISO/OSI protocol stack. For example, all the fields related to web application (e.g., URL, domain) belong to the F^* set for each analyzed packet filtering VNF. The analysis of such a high number of open-source functions shows how our model is general enough to support their representation as manifest. Therefore, it is also suitable for similar functions, and it can be easily extended for future VNFs by introducing new actions and fields in the A, F^+ and F^* sets.

4.2.3. Validation of correctness

The correctness of the framework has been validated by applying it on several use cases. We have written multiple NSPs, and we have run the framework to create the projection chains. For each NSP, we have run it multiple times, changing the VNF manifests that may be used, with the aim to create particular cases that could test specific characteristics of the algorithms used in the PID stage. For example, the following scenarios have been considered:

- VNFs with the same manifest are used to enforce an NSP, to check that the framework creates the same projection for them and uses it once in creating the projection chains;
- an inadequate number of VNFs is used to enforce an NSP, to check that the framework produces a non-enforceability report stating that a feasible solution for the security configuration problem does not exist;

[1] Link: https://www.gurobi.com/. Last accessed: February 20th, 2023.

- NSPs with unordered actions have been written, to check that the framework can consider also the more complex case where all the chains derived from the solution of the enumeration problem are subject to a post-processing step to consider all the possible permutations of the projections;
- NSPs with actions that must be applied on packet fields have been written, to check that the framework uses only the VNF manifests having all those fields in the F^+ for creating a projection.

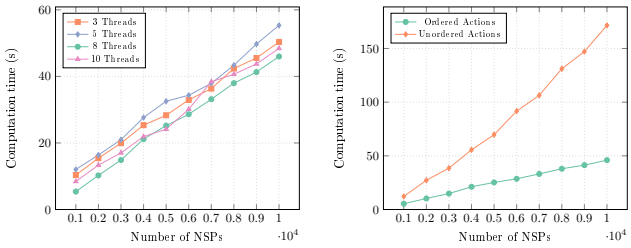
After the computation of the projections, tools for automatic security configuration (e.g., [20] for firewalls, [21] for VPN gateways) have been fed with the result of the PID stage. With the aim to verify that the NSPs are correctly enforced by the configuration computed by the tools when using the projections generated by our framework, we have used the Mininet emulator to instantiate the related network topologies in a controlled environment. The tests made on the Mininet emulation confirmed that all NSPs are satisfied, as expected.

Both the VNF manifests and the NSPs used for the generality and correctness tests are available in the GitHub repository, so that they can be used to reproduce the tests and can be extended to consider other scenarios.

4.2.4. Evaluation of scalability

A series of scalability tests have been done on the framework. The two parameters against which scalability has been evaluated are the numbers of NSPs and of VNFs from which the projections must be derived. The scenarios employed for these scalability tests are extended versions of the problem inputs used for the description of the clarifying use case in Section 3.1, i.e., the network topology shown in Figure 4, the VNF database of Table 1 and the NSPs of Table 2.

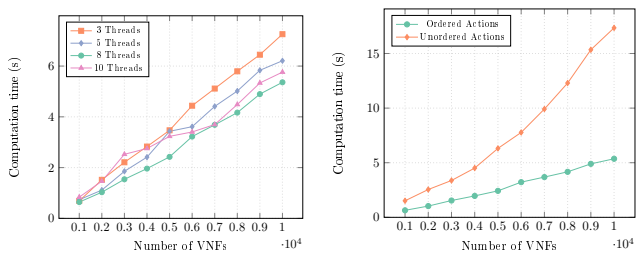
Figure 7 shows the results for the scalability tests related to the number of NSPs. For these tests, the number of VNFs is fixed to 100, whereas the number of NSPs



(a) Thread number (b) Action ordering

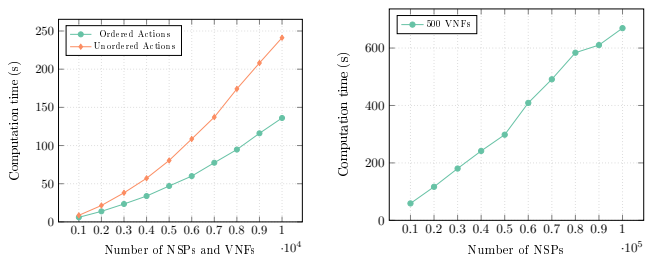
Figure 7: Scalability versus number of Network Security Policies

is progressively increased from 1000 to 10000. Each dotted plot in the charts composing Figure 7 represents the average value computed over 100 repetitions of the test. Initially, the behavior of the framework has been analyzed when also varying the number of threads used for the execution of the internal algorithms, and supposing that the NSP actions are ordered. The results depicted in Figure 7a show that, for the machine we have used, the least computation time is almost always achieved when eight threads are used. If a higher number of threads is employed, the performance gets worse, because thread creation overhead is not adequately compensated by our machine's CPU capacity of running so many threads simultaneously. Then, the difference in computation time that occurs if the NSPs do not impose an ordering to the actions has been evaluated. It is presented in Figure 7b. As it was expected, the behavior of the tool gets worse if the actions requested by each NSP are unordered. In the worst case that has been analyzed, less than one minute is enough to compute the solutions when the actions are ordered, whereas almost three minutes are required for unordered actions. This is easily explained by the fact that, in the PCH algorithm, all the possible permutations of the actions must be considered. Two remarks are worth mentioning, though. First, the case where all 10000 NSPs require unordered actions has been artificially created to put our system under great stress. In reality, the NSPs that must be enforced in a network have a mixed nature, i.e., some require ordered actions while others do not. Second, the result achieved for the worst case is significant by itself, if compared with



(a) Thread number (b) Action ordering

Figure 8: Scalability versus number of Virtual Network Functions



(a) Equal increasing number of NSPs and VNFs (b) Very high scalability versus number of NSPs

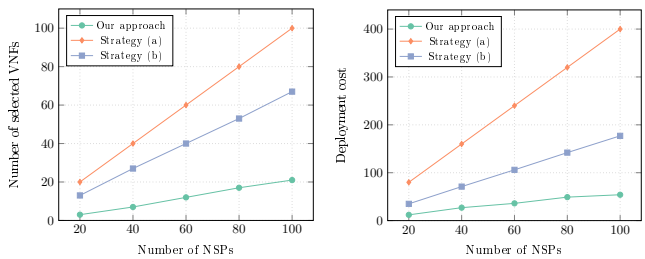
Figure 9: Validation under two peculiar scenarios

what human users may do manually. Manual approaches would struggle in dealing with such a huge number of NSPs without the aid of an automated tool, and they would take a much higher time than only three minutes to identify all the possible projections.

Similar considerations apply to the tests carried out for checking the framework scalability for increasing numbers of VNFs. The results of these tests are depicted in Figure 8. Differently from the previous tests, the number of NSPs is fixed to 100, whereas the number of VNFs is progressively increased from 1000 to 10000. Again, each dotted plot in the charts composing Figure 8 represents the average value computed over 100 repetitions of the test. On the one hand, from the analysis of those results, the creation of eight threads is confirmed to be the best choice for our machine. On the other hand, again the cases where policy actions are unordered require more time than those where actions are ordered. An interesting consideration is that the scalability for increasing number of VNFs is even better than the one for NSPs. This result can be explained by the fact that for each NSP the whole PID process must be executed, whereas each additional VNF simply represents an additional decision variable, but the number of times the PID process is executed stays the same.

Two additional scenarios where the framework scalability has been validated are shown in Figure 9. First, we have tested the implementation by equally increasing the numbers of VNFs and NSPs, until each of them is 10000. Even though the computation time is higher than the ones shown in Figure 7b and Figure 8b, Figure 9a shows that the trend is not exponential, but it follows the same growth of the previously analyzed scenarios, both for NSPs having ordered or unordered actions. Second, we have considered a stressing scenario, where the administrator of a big-sized network has 500 VNFs available for enforcing a huge number of NSPs, i.e., from 10000 to 100000 NSPs. Even though such high numbers may be rare, the fast growth of modern virtual networks may soon create circumstances where they are not so uncommon. Anyway the framework is able to manage this case as well, as shown in Figure 9b. The time required for computing the projection chains is surely higher, as for the worst case (i.e., the one characterized by 100000 NSPs and 500 VNFs) more than 10 minutes are required. But again the trend is not exponential, and the algorithm surely performs better than what a human user may achieve.

The results of these tests show that our framework has better scalability than state-of-the-art tools that are used for VNF selection. On the one hand, the algorithm proposed in [12] is able to select the most suitable VNFs among at most 50 ones in 1.5 s. On the other hand, the approach pursued in [13] computes the solution to the VNF selection problem for 1500 NSPs in 20 s. In both cases, our solution can reach the solution for projection identification in faster times (e.g., it takes 12 s to solve the problem for 1500 NSPs), but it can manage a much larger set of VNFs – Figure 8b shows its scalability till 10000 VNFs, Figure 9b shows its scalability till 100000 NSPs. Moreover, the computation time required by this implementation of the



(a) Optimization: VNFs (b) Optimization: deployment cost

Figure 10: Evaluation of optimization

PID stage is much lower than the time required by algorithms that may be employed in the next stages of the security configuration workflow. For example, the heuristic method employed in [16] to manage the ACG stage for 20 VNFs takes 4 s, the optimization strategy proposed in [20] requires around 3 minutes to configure 100 packet filtering firewalls, and the embedding algorithm illustrated in [40] takes up to 4 minutes to define the embedding scheme of 30 VNFs in a physical network. With respect to them, our implementation of the PID stage is in line with the strict timing requirements of virtual networks, and can manage scenarios with higher numbers of VNFs and NSPs.

4.2.5. Evaluation of optimization

The optimization that the novel security configuration workflow can achieve in terms of two main metrics (i.e., selected VNFs and deployment cost) has been evaluated varying the size of the configuration problem. This validation is based on similar scenarios as those used for the scalability evaluation. Indeed, also these scenarios are extended versions of the problem inputs used for the description of the clarifying use case in Section 3.1.

In order to show how our approach behaves with respect to the state of the art, we compare it with two strategies based on the traditional configuration workflow. We consider: (a) the worst-cost strategy that selects a different VNF instance for enforcing each NSP, and that always chooses VNFs whose manifest includes the NSP condition fields of the highest level in the ISO/OSI stack (e.g., to block packets going to a web server, a web-application firewall is always chosen instead of a packet filter); (b) a more optimized strategy that performs a pre-analysis of the NSRs to understand if for some of them a single VNF can be selected (e.g., when the policy conditions are the same for two NSPs). For these workflows, the algorithms that are used for the configuration and embedding stages are those presented respectively in [20, 40].

Figure 10a and Figure 10b show the optimization that our approach can provide with respect to the two traditional strategies. Strategy (a) clearly fails in optimizing both the number of selected VNFs and the deployment cost, as its only aim is to provide a solution to the configuration problem without looking for optimization. Strategy (b) improves the result with respect to the worst case, but it suffers from the way the traditional workflow is organized. In fact, as VNF selection occurs earlier than virtual network synthesis, strategy (b) cannot know where the security functionalities must perform their operations in the network topology. Therefore, it redounds them so as to consider all the possible ways in which the network may be synthesized. Instead, our approach achieves a considerable gain in both number of selected VNFs and deployment cost, as it benefits from all the advantages illustrated in Section 3.1. In fact, selecting the VNFs after their functionalites avoids redundant allocation schemes and chooses VNFs with a lower cost, as the VNF selection stage has all the required information about how security must be enforced in the network. The impact of these benefits can by quantified by estimating the improvement percentages of our approach with respect to the the worst-cost strategy (a) and the more optimez strategy (b), when the three techniques are applied to the validation scenario where 100 NSPs must be enforced. On the one hand, postponing VNF selection in this scenario allows the selection of 79% fewer VNFs than thw worst-case strategy (a), and 68% fewer VNFs than strategy (b). On the other hand, the savings in terms of deployment cost provided by our novel configuration workflow in this scenario is 90% with

respect the worst-case strategy (a), and 77% with respect strategy (b).

5. Conclusions and Future Work

This paper presented the projection abstraction, aiming to abstract the security-related operations that VNFs can perform to enforce security policies, in a way that is independent from the specific characteristics of their implementations. This abstraction allowed us to reorganize the traditional configuration workflow, in such a way that projections are used for the synthesis of the virtual security graph instead of their VNF counterparts, whose selection is postponed to the moment their deployment is required in the physical network. A formalization for this abstraction has been proposed, alongside with an algorithm to compute projection chains as candidate solutions that fully support the user-specified network security policies.

A Java-based framework has been developed to implement this algorithm, and validation tests have been carried out on a state-of-the-art computing machine to assess model generality, correctness, scalability and optimization. These tests showed that the proposed approach can optimize both VNF selection and embedding thanks to their different position in the new workflow. In fact, they showed that the developed framework can reduce both the number of selected VNFs and their deployment cost. For example, in the analyzed scenario, the improvement percentages for these two parameters are 79% and 90% with respect to the worst-case strategy, while 68% and 77% with respect to a traditional more optimized configuration strategy. These benefits can be particularly relevant for scenarios such as virtualized IoT networks, where energy efficiency and resource consumption are problems that must comply with the limited resources provided by constrained devices.

As future work, we are planning to integrate the security configuration workflow based on the projection abstraction with reaction and mitigation techniques. The objective of that work is to create a fully autonomous system, where network security is periodically updated in accordance with policies defined by human users, and also with the identification of on-going cyber attacks.

References

[1] R. Boutaba, I. Aib, Policy-based management: A historical perspective, J. Netw. Syst. Manag. 15 (4) (2007) 447–480.

[2] C. Islam, M. A. Babar, S. Nepal, A multi-vocal review of security orchestration, ACM Comp. Surv. 52 (2) (2019) 37:1–37:45.

[3] B. Jäger, Security orchestrator: Introducing a security orchestrator in the context of the ETSI NFV reference architecture, in: Proc. of the IEEE TrustCom/BigDataSE/ISPA, 2015.

[4] C. Pham, N. H. Tran, S. Ren, W. Saad, C. S. Hong, Traffic-aware and energy-efficient vnf placement for service chaining: Joint sampling and matching approach, IEEE Trans. Serv. Comp. 13 (1) (2020).

[5] I. Farris, T. Taleb, Y. Khettab, J. Song, A survey on emerging SDN and NFV security mechanisms for iot systems, IEEE Commun. Surv. Tutorials 21 (1) (2019) 812–837.

[6] I. Alam, K. Sharif, F. Li, Z. Latif, M. M. Karim, S. Biswas, B. Nour, Y. Wang, A survey of network virtualization techniques for internet of things using SDN and NFV, ACM Comput. Surv. 53 (2) (2021) 35:1–35:40.

[7] S. Javanmardi, M. Shojafar, R. Mohammadi, V. Persico, A. Pescapè, S-fos: A secure workflow scheduling approach for performance optimization in sdn-based iot-fog networks, J. Inf. Secur. Appl. 72 (2023) 103404.

[8] A. V. Ventrella, F. Esposito, A. Sacco, M. Flocco, G. Marchetto, S. Gururajan, APRON: an architecture for adaptive task planning of internet of things in challenged edge networks, in: 2019 IEEE 8th International Conference on Cloud Networking, CloudNet 2019, Coimbra, Portugal, November 4-6, 2019, IEEE, 2019, pp. 1–6.

[9] D. Mishra, B. Naik, J. Nayak, A. Souri, P. B. Dash, S. Vimal, Light gradient boosting machine with optimized hyperparameters for identification of malicious access in iot network, Digital Communications and Networks (2022).

[10] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, A novel approach for security function graph configuration and deployment, in: Proc. of the IEEE Inter. Conf. on Network Softwarization, 2021.

[11] E. J. Scheid, C. C. Machado, M. F. Franco, R. L. dos Santos, R. J. Pfitscher, A. E. S. Filho, L. Z. Granville, Inspire: Integrated nfv-based intent refinement environment, in: Proc.

of the IFIP/IEEE Symp. on Integrated Network and Service Management, 2017.

[12] Z. Hao, Z. Lin, R. Li, A SDN/NFV security protection architecture with a function composition algorithm based on trie, in: Proc. of the Inter. Conf. on Computer Science and Application Engineering, 2018.

[13] Y. Liu, H. Zhang, J. Liu, Y. Yang, A new approach for delivering customized security everywhere: Security service chain, Secur. Commun. Networks 2017 (2017).

[14] Y. Liu, Y. Lu, W. Qiao, X. Chen, A dynamic composition mechanism of security service chaining oriented to sdn/nfv-enabled networks, IEEE Access 6 (2018).

[15] A. S. Sendi, Y. Jarraya, M. Pourzandi, M. Cheriet, Efficient provisioning of security service function chaining using network security defense patterns, IEEE Trans. Serv. Comput. 12 (4) (2019).

[16] C. Basile, F. Valenza, A. Lioy, D. R. López, A. P. Perales, Adding support for automatic enforcement of security policies in NFV networks, IEEE/ACM Trans. Netw. 27 (2) (2019).

[17] A. El-Hassany, P. Tsankov, L. Vanbever, M. T. Vechev, Netcomplete: Practical network-wide configuration synthesis with autocompletion, in: S. Banerjee, S. Seshan (Eds.), Proc. of the USENIX Symp. on Networked Systems Design and Implementation, 2018.

[18] C. Bodei, P. Degano, L. Galletta, R. Focardi, M. Tempesta, L. Veronese, Language-independent synthesis of firewall policies, in: Proc. of the IEEE European Symp. on Security and Privacy, 2018.

[19] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, J. Yusupov, Automated optimal firewall orchestration and configuration in virtualized networks, in: Proc. of the IEEE/IFIP Network Operations and Management Symposium, 2020.

[20] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, J. Yusupov, Automated firewall configuration in virtual networks, IEEE Trans. Dependable Secur. Comput.In press (2023).

[21] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, Short paper: Automatic configuration for an optimal channel protection in virtualized networks, in: Proc. of the Work. on Cyber-Security Arms Race, 2020.

[22] L. Firdaouss, A. Bahnasse, B. Manal, Y. Ikrame, Automated VPN configuration using devops, in: N. Varandas, A. Yasar, H. Malik, S. Galland (Eds.), The 12th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2021), Leuven, Belgium, November 1-4, 2021, Vol. 198 of Procedia Computer Science, Elsevier, 2021, pp. 632–637.

[23] A. M. Zarca, J. B. Bernabé, R. Trapero, D. Rivera, J. Villalobos, A. F. Skarmeta, S. Bianchi, A. Zafeiropoulos, P. Gouvas, Security management architecture for nfv/sdn-aware iot systems, IEEE Internet Things J. 6 (5) (2019) 8005–8020.

[24] M. A. Rahman, A. Datta, E. Al-Shaer, Automated configuration synthesis for resilient smart metering infrastructure, EAI Endorsed Trans. Security Safety 8 (28) (2021) e4.

[25] D. Bringhenti, J. Yusupov, A. M. Zarca, F. Valenza, R. Sisto, J. B. Bernabé, A. F. Skarmeta, Automatic, verifiable and optimized policy-based security enforcement for sdn-aware iot networks, Comput. Networks 213 (2022) 109123.

[26] J. Kim, Y. Kim, V. Yegneswaran, P. A. Porras, S. Shin, T. Park, Extended data plane architecture for in-network security services in software-defined networks, Comput. Secur. 124 (2023) 102976.

[27] L. Xia, J. Strassner, C. Basile, D. R. Lopez, Information model of nsfs capabilities, Rfc, RFC Editor (2019).

[28] D. R. López, E. Lopez, L. Dunbar, J. Strassner, R. Kumar, Framework for interface to network security functions, RFC 8329.

[29] K. Giotis, Y. Kryftis, V. Maglaris, Policy-based orchestration of NFV services in software-defined networks, in: Proceedings of the 1st IEEE Conference on Network Softwarization, NetSoft 2015, London, United Kingdom, April 13-17, 2015, IEEE, 2015, pp. 1–5.

[30] S. Hyun, J. Kim, H. Kim, J. Jeong, S. Hares, L. Dunbar, A. Farrel, Interface to network security functions for cloud-based security services, IEEE Commun. Mag. 56 (1) (2018) 171–178.

[31] A. M. Zarca, D. G. Carrillo, J. B. Bernabé, J. O. Murillo, R. Marín-Pérez, A. F. Skarmeta, Enabling virtual AAA management in sdn-based iot networks, Sensors 19 (2) (2019) 295.

[32] Y. Bartal, A. Mayer, K. Nissim, A. Wool, *Firmato*: A novel firewall management toolkit, ACM Trans. Comp. Syst. 22 (4) (2004).

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

[34] M. Rossberg, G. Schaefer, T. Strufe, Distributed automatic configuration of complex ipsec-infrastructures, J. Network Syst. Manage. 18 (3) (2010).

[35] S. Jiao, X. Zhang, S. Yu, X. Song, Z. Xu, Joint virtual network function selection and traffic steering in telecom networks, in: Proc. of the IEEE Global Communications Conference, 2017.

[36] J. Pei, P. Hong, D. Li, Virtual network function selection and chaining based on deep learning in SDN and nfv-enabled networks, in: Proc. of the IEEE Inter. Conf. on Communications Workshops, 2018.

[37] J. Pei, P. Hong, K. Xue, D. Li, D. S. L. Wei, F. Wu, Two-phase virtual network function selection and chaining algorithm

based on deep learning in sdn/nfv-enabled networks, IEEE J. Sel. Areas Commun. 38 (6) (2020).

[38] L. E. Li, V. Liaghat, H. Zhao, M. Hajiaghayi, D. Li, G. T. Wilfong, Y. R. Yang, C. Guo, PACE: policy-aware application cloud embedding, in: Proc. of the IEEE INFOCOM, 2013.

[39] X. Li, C. Qian, An NFV orchestration framework for interference-free policy enforcement, in: 36th IEEE International Conference on Distributed Computing Systems, ICDCS 2016, Nara, Japan, June 27-30, 2016, IEEE Computer Society, 2016, pp. 649–658.

[40] G. Marchetto, R. Sisto, F. Valenza, J. Yusupov, A. Ksentini, A formal approach to verify connectivity and optimize VNF placement in industrial networks, IEEE Trans. Ind. Informatics 17 (2) (2021).

[41] C. Basile, C. Pitscheider, F. Risso, F. Valenza, M. Vallini, Towards the dynamic provision of virtualized security services, in: F. Cleary, M. Felici (Eds.), Cyber Security and Privacy - 4th Cyber Security and Privacy Innovation Forum, CSP Innovation Forum 2015, Brussels, Belgium, April 28-29, 2015, Revised Selected Papers, Vol. 530 of Communications in Computer and Information Science, Springer, 2015, pp. 65–76.