A novel approach for security function graph configuration and deployment

Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza
Dip. Automatica e Informatica, Politecnico di Torino, Torino, Italy, Emails: {first.last}@polito.it

***Abstract*—Network virtualization increased the versatility in enforcing security protection, by easing the development of new security function implementations. However, the drawback of this opportunity is that a security provider, in charge of configuring and deploying a security function graph, has to choose the best virtual security functions among a pool so large that makes manual decisions unfeasible. In light of this problem, the paper proposes a novel approach for synthesizing virtual security services by introducing the functionality abstraction. This new level of abstraction allows to work in the virtual level without considering the different function implementations, with the objective to postpone the function selection jointly with the deployment, after the configuration of the virtual graph. This novelty enables to optimize the function selection when the pool of available functions is very large. A framework supporting this approach has been implemented and it showed adequate scalability for the requirements of modern virtual networks.**

***Index Terms*—virtualized networks, network security, automated orchestration, function selection**

I. Introduction

Network Functions Virtualization (NFV) [1] and *Software-Defined Networking* (SDN) [2] have introduced high flexibility in enforcing security protection in modern computer networks. When a security provider must enrich a virtual service graph with the security functions that are needed to fulfill some *Network Security Policies* (NSPs, i.e., the security requirements that must be satisfied in the virtual service [3]) with the aim to build a security function graph, she has a huge number of alternative softwarized functions at her disposal. These security functions are commonly known as *virtual Network Security Functions* (vNSFs), and are programs which can run on general Virtual Machines or Dockers without requiring special-purpose embedding hardware.

Nevertheless, this high freedom of choice has a drawback. In the approach that is commonly pursued for enforcing security in a virtual network service [4], security providers select the vNSFs required to enforce the NSPs before the two next stages of security orchestration, i.e., i) the security enforcement in the virtual service, through the definition of allocation scheme and configuration of the selected vNSFs in the topology, and ii) their deployment in the physical infrastructure of the network. Due to this ordering of the operations, network information (e.g., the topology of the virtual service, or the behavior of service functions like load balancers and network address translators) is overlooked in the selection of the vNSFs. Ignoring this information may result into sub-optimizations impacting the stages following the vNSF selection, because it may have been useful to identify the minimum and the least resource consuming subset of vNSFs that are really needed among potentially thousands of them available.

An alternative approach would be to postpone the selection of the vNSFs after the enforcement of the security protection in the virtual service, just before or jointly with their deployment. Hence, the challenge that arises would be to understand how the virtual security function graph could be synthesized and configured, if the vNSFs are selected afterwards. It is clear that a link, previously represented by the vNSF selection, would be missing between the NSP specification and the security enforcement in the virtual function graph.

In light of all these considerations, this paper proposes a novel approach for security function graph configuration and deployment. In this approach, as hinted before, the vNSF selection is performed after the synthesis of the virtual graph. This is enabled by the introduction of a new level of abstraction: in each vNSF a set of functionalities (i.e., function features which can enforce corresponding security properties) is identified, and the virtual graph is created by allocating and configuring functionalities instead of vNSFs. Only later, depending on the allocated functionalities and their computed configuration, the effective selection and deployment jointly occur for the vNSFs, in a final single and optimized step. Last but not least, this approach is fully automated, so that all the stages of the process (i.e., functionality allocation and configuration, vNSF selection and placement) work on inputs provided by service providers but without requiring additional human intervention.

In summary, the contributions brought over by this proposal are the following.

A new level of abstraction. In NFV/SDN-based networks, traditionally the virtual security graph is already composed of vNSFs. By introducing the functionality abstraction, the virtual graph is built upon functionalities, enabling higher versatility for the next stages.

A novel approach for security graph configuration and deployment. The vNSF selection is postponed after the synthesis and configuration of a security graph composed by functionalities. This decision required to remodel the process, and to redefine the working mode of each stage.

An automated framework for security orchestration. This novel approach has been implemented within a prototype framework which can work with minimum external interactions, reducing human factor in producing errors and improving the performance efficiency.

The rest of this paper is structured as follows. Section II presents the functionality abstraction required for postponing the vNSF selection stage. Section III illustrates the novel approach that has been defined for security function graph configuration and deployment, thanks to the newly introduced functionality abstraction. Section IV describes the implementation of a framework based on this approach and the results of an experimental validation. Section V dissects related work, highlighting the differences with respect to the proposed approach. Finally, Section VI concludes the paper and prospects future work.

II. The Functionality Abstraction

The functionality abstraction represents a novelty which enables to split the vNSF selection from the synthesis of virtual security function graphs. This section illustrates how the vNSFs should be represented to enable this abstraction, and then it shows how the functionalities are derived from that representation.

A. vNSF manifest

Each vNSF is characterized by parameters that define the security properties it can enforce (e.g., the layer of the ISO/OSI stack where the vNSF 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 vNSF, they can be grouped in a single representation, called **vNSF manifest**.

For a vNSF 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 fields and features for which the vNSF can take a decision and/or which can be configured on it;
- A_v is the set of all the actions that the vNSF can enforce.

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

- F_v^+ is the set of all the fields for which the vNSF 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 fields for which a vNSF 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).

In the following, three examples are presented to clarify the concept of vNSF 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 sake of conciseness.

$$\boxed{\begin{array}{l}\text{vNSF } v_1\text{: iptables}\\ F_{v_1}^+ = \{\text{IPSrc}, \text{IPDst}, \text{pSrc}, \text{pDst}, \text{tProto}\}\\ F_{v_1}^* = \{\text{domain}, \text{url}, \text{mailAddress}, \text{payload}, ...\}\\ A_{v_1} = \{\text{allow}, \text{deny}\}\end{array}} \quad (1)$$

$$\boxed{\begin{array}{l}\text{vNSF } v_2\text{: Squid}\\ F_{v_2}^+ = \{\text{IPSrc}, \text{IPDst}, \text{pSrc}, \text{pDst}, \text{tProto}, \text{domain}, \text{url}, ...\}\\ F_{v_2}^* = \{\text{mailAddress}, \text{payload}, ...\}\\ A_{v_2} = \{\text{allow}, \text{deny}, \text{log}\}\end{array}} \quad (2)$$

$$\boxed{\begin{array}{l}\text{vNSF } v_3\text{: MyLogger}\\ F_{v_3}^+ = \{\text{domain}, \text{url}\}\\ F_{v_3}^* = \{\text{mailAddress}, \text{payload}, ...\}\\ A_{v_3} = \{\text{allow}, \text{log}, \text{alert}\}\end{array}} \quad (3)$$

The manifest of a packet filtering vNSF 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 vNSFs 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, PUT, 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 vNSF that has been developed and is available for the security provider is named "MyLogger". It cannot block packets, but it can only log the receiving of specific kinds of traffic and notifying the security provider 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 as well, because domain and url are information of higher level than layers 3 and 4 of the ISO/OSI stack.

B. Security functionalities definition

A security functionality (for brevity simply called functionality in the following) expresses how a subset of fields and actions (i.e., the manifest) of a vNSF is used to enforce the security properties expressed by an NSP.

As the person in charge of the NSP definition cannot know beforehand which vNSFs will be effectively deployed to enforce them, an implementation-independent language is used for their formulation. In literature, a similar level of abstraction is also called *Medium Level Policy Language* (MLPL) [5], because it provides all the information for the NSP enforcement, but in a way that is agnostic to the employed vNSFs.

An NSP p can be represented in MLPL as $p = (C_p, A_p)$:

- C_p is the set of policy conditions that must be fulfilled (e.g., they might specify the value of the IP 5-tuple fields representing a type of traffic to be blocked by a firewall, or the type of encryption algorithm to be employed by a VPN gateway);
- A_p is an ordered list of actions that must be applied on the packets identified by the conditions expressed in C_p.

An example of policy p is shown in (4).

$$\boxed{\begin{aligned}&\text{NSP } p\\ &C_p = \{\text{IPSrc} = 125.10.2.0/24, \text{IPSrc} = 20.20.20.1, \text{pSrc} = *,\\ &\qquad \text{pDst} = 80, \text{tProto} = \text{TCP}, \text{domain} = \text{dangerousSite.com}\}\\ &A_p = [\text{log}, \text{deny}]\end{aligned}} \tag{4}$$

In this formulation, whenever a field does not have to be characterized by a specific value, the $*$ value is used for the condition on that field. For example, pSrc $= *$ means that there is not a strict condition on the value the transport-level source port must have for the identification of the traffic on which the security actions must be enforced.

The functionalities derive from mapping the manifest of each vNSF to an NSP, and they represent how the vNSFs *might* be configured to enforce the security properties requested by the NSP. Therefore, each functionality f is represented in a similar way as the NSP itself, i.e., $f = (C_f, A_f)$. Specifically, C_f is the set of conditions applied on the fields of F_v^+ of the vNSF v from which f derives, whereas A_f is an ordered list of actions supported by the A_v set of the corresponding vNSF.

For instance, considering the three vNSFs v_1, v_2 and v_3 whose manifests have been presented in (1), (2) and (3), the functionalities deriving from those manifests and necessary to enforce the policy p presented in (4) are the following:

$$\boxed{\begin{aligned}&\text{Functionality } f_1\text{, derived by mapping the vNSF } v_1 \text{ to the NSP } p\\ &C_{f_1} = \{\text{IPSrc} = 125.10.2.0/24, \text{IPSrc} = 20.20.20.1, \text{pSrc} = *,\\ &\qquad \text{pDst} = 80, \text{tProto} = \text{TCP}\}\\ &A_{f_1} = [\text{deny}]\end{aligned}} \tag{5}$$

$$\boxed{\begin{aligned}&\text{Functionality } f_2\text{, derived by mapping the vNSF } v_2 \text{ to NSP } p\\ &C_{f_2} = \{\text{IPSrc} = 125.10.2.0/24, \text{IPSrc} = 20.20.20.1, \text{pSrc} = *,\\ &\qquad \text{pDst} = 80, \text{tProto} = \text{TCP}, \text{domain} = \text{dangerousSite.com}\}\\ &A_{f_2} = [\text{log}, \text{deny}]\end{aligned}} \tag{6}$$

$$\boxed{\begin{aligned}&\text{Functionality } f_3\text{, derived by mapping the vNSF } v_3 \text{ to NSP } p\\ &C_{f_3} = \{\text{dangerousSite.com}\}\\ &A_{f_3} = [\text{log}]\end{aligned}} \tag{7}$$

It is necessary that a single functionality or a combination of functionalities supports all the actions required by the NSP so that it is enough to enforce it. In this example, the functionality f_2 supports both the actions of logging and packet blocking, therefore it might be sufficient to enforce the NSP. Instead, f_1 and f_3 are not enough by themselves, because the former can only block packets, while the latter can only perform logging operations. As such, a combination of f_3 followed by f_1 would be needed.

In light of these definitions and examples, a consequence is that identifying all the possible combinations of functionalities that might enforce the NSPs allows to postpone the effective vNSF selection and leads to two important advantages, which have not been introduced in literature as for now. The first benefit is that, when building the virtual security graph, all the possible functionalities are evaluated, while considering the behavior of the virtual network itself. Only later, after having decided the functionalities that are really needed, the vNSFs are selected for the deployment in the physical network. The second advantage is that, if there are multiple vNSF implementations with the same manifest (e.g., iptables, ipfirewall and other packet filtering firewalls would have almost identical manifests), then the functionalities deriving from mapping their manifest to an NSP would be identical as well. At that point, considering a single functionality for the next stage of the orchestration process is enough, in place of considering multiple vNSFs which may differ only for resource demands (e.g., CPU or RAM requirements) but are the same in terms of offered security properties.

III. The Novel Workflow

This section describes the full novel workflow designed for security function graph configuration and deployment, after the introduction of the functionality abstraction. Specifically, this workflow, as depicted in Fig. 1, is composed of three stages:

1) the ***FU**nctionality **ID**entification* (**FUID**) stage (Subsection III-A) identifies which functionalities, supported by the available vNSFs, are required to enforce a set of user-specified NSPs;
2) the ***A**llocation and **C**onfiguration **G**eneration* (**ACG**) stage (Subsection III-B) automatically establishes the allocation scheme and configuration of the identified security functionalities in the virtual graph;
3) the ***SE**lection and **P**lacement* (**SEP**) stage (Subsection III-B) performes the selection of the vNSFs, postponed with respect to a traditional workflow, and their deployment in the physical network.

A. FUnctionality IDentification

The FUID stage works on a set of NSPs specified by the security provider. After receiving a set of NSPs expressed within MLPL, the FUID stage identifies the functionalities that might be needed to enforce them. To perform this task, it has access to a repository containing all the vNSFs that would be available for the effective deployment in the network.

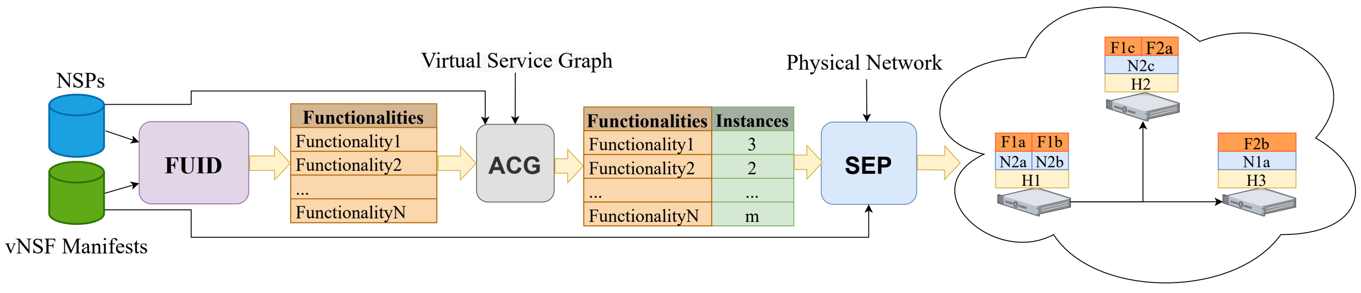


Fig. 1: Workflow of the security function graph configuration and deployment

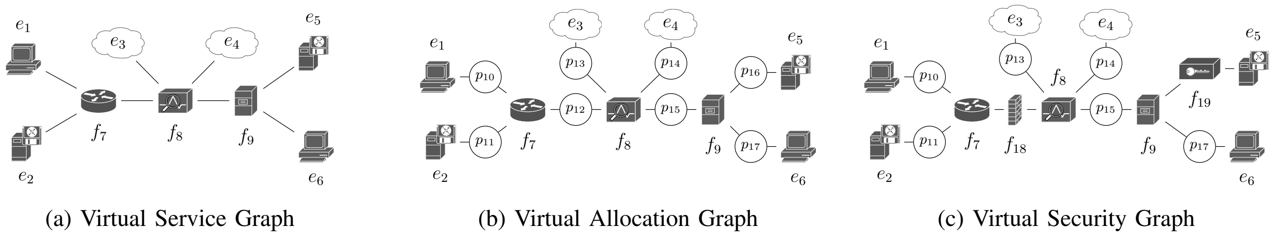


Fig. 2: Function graphs employed in the ACG stage

Each vNSF is accompanied with a manifest that follows the definition presented in Section II.

The manifest of each vNSF is mapped to each NSP, and the result of this operation is a security functionality. It might happen that, given more vNSF manifests, their resulting functionalities for a given NSP are identical. In that case, a single instance of that functionality is kept. Similarly, it might happen that a vNSF does not support any field among the ones specified in the conditions of an NSP. No functionality will derive from mapping its manifest to the NSP, and that vNSF is thus excluded a-priori for its enforcement.

Then, the computed functionalities are combined in such a way that only valid combinations that can fully enforce the actions requested by an NSP are considered. In case no valid combination is identified for an NSP, the global process immediately halts, and an early non-enforceability report is produced, notifying the security provider why the functionality identification failed. Otherwise, the valid combinations are passed on to the next stage of the process, for their allocation in the virtual graph. In this way, the security configuration will be performed in a way that is agnostic to the vNSF implementation.

In this stage, no information related to the virtual network is employed. This allows to execute the functionality identification for each NSP independently from the others.

B. Allocation and Configuration Generation

The ACG stage works on three inputs:

1) the NSPs specified by the service provider;
2) the set of functionalities identified in the FUID stage (i.e., the ACG stage is vNSF-agnostic);
3) the description of a *Virtual Service Graph* (VSerG).

The VSerG description provides information about the topology of the service where the security properties must be enforced, and the configuration of the network function (e.g., NATs, load balancers) composing it. As such, it represents the current status of an existing network graph, but completely devoid of any security function. In the ACG stage, a VSerG is internally represented as a *Virtual Allocation Graph* (VAllG). A VAllG contains a virtual placeholder position, called *Allocation Place* (AP), for each link among a pair of services of the corresponding VSerG. These APs represent the possible places where the functionalities identified in the FUID stage can be allocated and configured. For example, the VSerG depicted in Fig. 2a is represented as the VAllG depicted in Fig. 2b.

The purposes of the ACG stage are to establish the optimal and formally correct allocation scheme in the VSerG and configuration of the previously selected functionalities, in such a way to enforce all the NSPs. These goals are achieved by formulating and solving a *Maximum Satisfiability Modulo Theories* (MaxSMT) problem. This type of optimization problem derives its principles from the traditional SAT problem, but extends them in two ways. On one side, it introduces additional theories than the only Boolean algebra. On the other side, it is characterized not only by *hard* constraints that must be always satisfied (e.g., the ones expressing the enforcement of the NSPs), but also *soft* constraints that should be simply satisfied as far as possible and allow the formalization of optimization objectives. Such an approach is inspired by works proposed in literature about packet filtering firewalls [6] and VPN gateways [7], which have nonetheless a limited scope and proved its validity for a single type of function. Besides, this formulation enables two features: optimization and a-priori formal verification. About the former, an optimization objective is to establish the minimum number of required functionality instances. About the latter, following the trend of works such as [8], formal models are used to capture the behavior of service functions characterizing the VSerG, so that the solution for the MaxSMT problem has a formal assurance of being correct thanks to the intrinsic correctness-by-construction paradigm of a MaxSMT formulation.

The solution to the MaxSMT problem is a *Virtual Security Graph* (VSecG), where some APs are filled with the selected functionalities, which are also configured in such a way to enforce the user-specified NSPs. An example of VSecG is depicted in Fig. 2c, where f_{18} and f_{19} are security functionalities introduced in the VAllG. This graph represents a higher abstraction than the traditional virtual graph, where vNSFs are typically present.

C. SElection and Placement of the vNSFs

The SEP stage works on the following inputs:

1) the VSecG computed in the ACG stage;
2) the repository of vNSFs available for deployment;
3) the description of the physical network (i.e., how it is structured, and the resource availabilities of each host composing it);
4) an optimization policy, describing how each optimization criterion (i.e., minimization of CPU, RAM, disk, bandwidth, number of selected vNSFs and number of used physical hosts) has higher or lower priority than the others.

A first aim of this stage is to select the best subset of vNSFs, among the available ones, that supports the functionalities allocated and configured in the ACG stage. Jointly, the deployment scheme of the selected vNSFs is established, i.e., the decision on which host composing the physical network each vNSF should be embedded is taken. Having postponed the vNSF selection so that it is performed with their deployment has many advantages. At this stage, it is clear which functionalities are effectively needed, and how they have been allocated in the virtual graph. Besides, these two operations (i.e., selection and embedding) are typically subject to the same optimization criteria and constraints, even though they are traditionally in different steps of the security orchestration workflow. For example, in this novel approach, a certain vNSF is directly avoided to be selected if there is no a physical host available with adequate RAM capacity.

The selection and deployment problem has been formulated as a multi-objective *Integer Linear Programming* (ILP) problem. If there are hundreds of available vNSFs 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.). Therefore, an ILP formulation enables an automated resolution of the joint selection and deployment problem. More specifically, considering a set of n vNSFs v_1, v_2, ..., v_n, in this formulation an integer variable x_i is defined for each vNSF, representing the outcome of the selection (i.e., $x_i = 1$ if v_i has been selected to be deployed, $x_i = 0$ otherwise). These variables recur in constraints related to the resource capacity of the hosts (i.e., the selected vNSFs must be deployed in hosts having enough resources for their demands), and also in the optimization objectives.

Concerning optimization, the proposed approach is multi-objective. As such, it seeks the best solution that can be achieved by optimizing the resource consumption of the physical hosts according to the requested optimization policy, where the criteria have different priorities. In such an approach, the optimization is performed hierarchically, i.e., firstly the criterion with maximum priority is optimized, then the criterion with the second higher one, and so on. Formula (8) is an example of optimization objective defined in the ILP problem.

$$\text{Obj}\Big(\min\Big(\sum_{i=1}^{n} n_i * d_i * x_i\Big), p_d\Big) \tag{8}$$

In this formula, for a vNSF v_i, n_i is an integer variable representing the number of selected instances, d_i is an integer variable representing the disk required (in MB), while p_d represents the priority of the criterion related to the disk consumption with respect to the others. In words, (8) states that the sum of the disk space consumed by selected and deployed vNSFs should be minimized as far as possible, in compliance with other constraints of the ILP problem.

At the end, the deployment scheme is generated for the selected vNSFs, and it is passed on to the user so that she can deploy the functions by interacting with an orchestrator such as Open Baton or Kubernetes.

IV. Implementation and Validation

This section describes how the proposed novel workflow has been implemented, together with an experimental validation that has been performed to examine its scalability.

A. Implementation

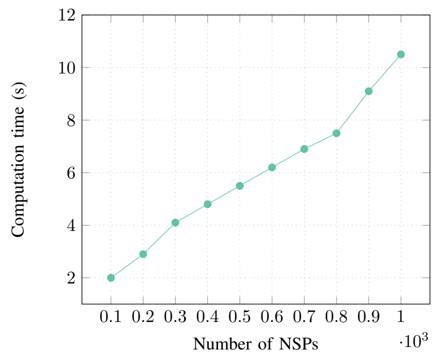
A prototype Java-based framework has been developed to implement the approach described in Section III. Specifically:

- a Java algorithm is employed for the FUID stage;
- a state-of-the-art MaxSMT solver, called Z3 (version 4.8.5) [9], is used to formulate and solve the optimization problem represented by the ACG stage;
- a state-of-the-art ILP solver, called Gurobi (version 8.1.1) [10], is used to formulate and solve the ILP problem for the SEP stage.

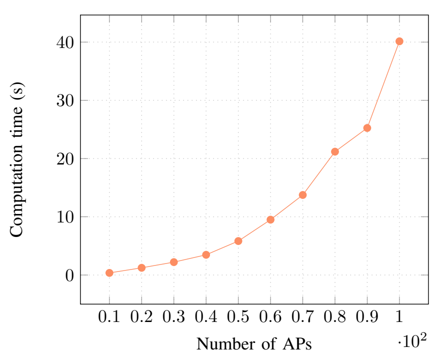
The framework exposes REST APIs for the interaction with human beings (e.g., service providers) and with other applications (e.g., automated service orchestrators such as Open Baton). The data exchanged through HTTP Requests and HTTP Replies can have an XML or JSON embedding.

B. Validation

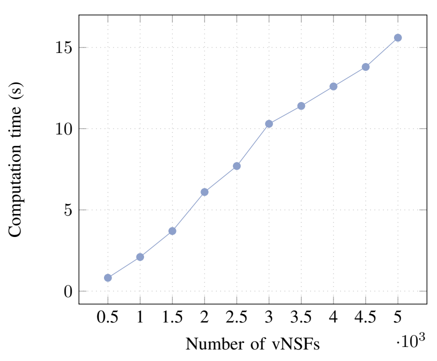
An experimental validation of the framework has been performed with the main aim of analyzing its scalability. This validation has been performed on a 4-core Intel i7-6700 3.40 GHz workstation with 32 GB RAM. Even though this prototype framework is based on a preliminary model



(a) FUID stage



(b) ACG stage



(c) SEP stage

Fig. 3: Experimental results of scalability tests

for the functionality abstraction, a scalability validation is interesting to understand the potential of this novel approach. In particular, considering the three stages of the proposed workflow, the metric of interest for each one varies. Therefore, the analyzed metrics have been the following:

- the number of NSPs for the FUID stage, because each NSP must be analyzed to identify the functionalities required to enforce it;
- the number of APs for the ACG stage, because each AP represents a candidate position in the virtual service for allocating a functionality;
- the number of available vNSFs for the SEP stage, because each one represents a possible choice for enforcing the allocated functionalities in the network.

The performance of each stage has been investigated independently of the others, by varying its own metric of interest.

Fig. 3 shows the results achieved for this scalability evaluation. Each value plotted in the charts represents the average value, computed over the results of 100 independent iterations.

These results point out that the framework has a good scalability, especially for the FUID and SEP stages. In fact, the former requires just a bit more than 10 seconds for identifying the required functionalities to enforce 1000 NSPs, whereas the latter performs the vNSF selection and deployment, accessing a repository of 1000 vNSFs, in around 15 seconds. The two corresponding plots in Fig. 3 show that for these stages the computation time does not increase exponentially, but it has a linear growth. The only stage that requires more time is ACG: around 40 seconds are needed to compute the functionality allocation scheme and configuration in a VAllG having 100 APs. The growth itself of the computation time for the ACG stage is not linear, but quadratic. There are two main reasons which explain this different behavior. The first is that the problem solved in the ACG stage is the most complex one, since configuring distributed functionalities across a virtual service, in an optimal and provably correct way, would be unfeasible for a human being. The second is that solving a MaxSMT problem requires more time than solving an ILP problem. Nonetheless, the MaxSMT formulation is needed for the support of additional theories and for enabling the correctness-by-construction paradigm of the approach.

Nevertheless, the total computation time required for performing all the steps of the proposed workflow is in line with the time required for network security management operations in virtual networks. As reference, [11] reports that the Deployment Process Delay (i.e., the time required for deploying and instantiating a virtual function and setting up an operational network service) is 134s for Open Source MANO, a well-known NFV-based orchestrator. Therefore, in less than this DPD time, the framework implemented to support the approach proposed in this paper can establish how a full virtual security service should be configured and deployed.

In conclusion, the experimental results confirm that not only the approach is suitable for replacing human beings in the security function graph configuration and deployment, but it can do so in much less time than what would require a person, and in a way that is compatible with the agility requirements of modern computer networks.

V. Related Work

Concepts akin to the functionality abstraction illustrated in this paper have barely been investigated in literature. A series of IETF RFC drafts, of which [12] is the most recent one, proposes the Capability Information Model to describe the security properties that a vNSF can enforce in a vendor-neutral manner. With such a description, it is not required to refer to a specific technology or vendor-dependent function when defining a security service. However, these ideas have not been completely formalized and exploited for researching novel ways to perform security configuration and deployment. In fact, [13] uses a capability model for abstracting vNSFs, but that work is restricted to access control and forwarding virtual functions. The work presented in [5] broadens the research to other types of security functions. However, as in [13], the capability model is not used to innovate the security configuration workflow, and the vNSF selection for the security enforcement is performed before the allocation and configuration stage as usual, losing all the benefits deriving from a possible postponement.

More research has been carried out about the automatic configuration of security functions on one side, about their deployment on the other side. Automatic ways for configuring vNSFs have been investigated singularly for packet filtering firewalls ([14], [15], [6]) and VPN gateways ([16], [7]), or for multiple types of functions at the same time ([5]). However, all these approaches are limited to establish the configuration of vNSFs instead of functionalities, and do not address the need of synthesizing their allocation scheme. Moreover, optimization criteria are not enforced with some rare exceptions (e.g., [6]), and only heuristic algorithms are employed without guarantees that the computed solutions are effectively optimized.

Finally, more effort has been spent in researching strategies for the deployment of virtual functions. Some relevant works in this research area are [17], [18] and [19]. A more exhaustive survey about deployment and embedding of virtual functions is presented in [20]. With respect to all these traditional approaches, in the workflow illustrated in this paper the deployment scheme is generated alongside the vNSF selection, as enabled by the postponement of that operation. This difference not only allows to perform the two operations jointly, but also optimizes them since they are constrained to the same optimization criteria.

VI. Conclusion and Future Work

The paper presented a novel approach for automatically configuring and deploying security functions in a virtual security function graph. This approach lays its foundations on the definition of the functionality abstraction, which allows to select the functions required to enforce security protection just before the deployment, after synthesizing the virtual graph. The overall process thus benefits from the optimizations deriving from this abstraction. A prototype framework has also been implemented to show the validity and scalability of this novel approach.

Future works are planned to be carried out regarding both the definition of the functionality abstraction and the formulation of the configuration and deployment workflow. About the former, different granularities for the functionality features may be acceptable. As such, an extensive evaluation of them will be conducted, with the purpose to understand the most suitable granularity and consequently model all the functionalities by using it. About the latter, the formulation of some stages may be altered (e.g., the MaxSMT problem defined for ACG process may be replaced or supported by heuristics), so that the overall performance of the framework can further improve.

Acknowledgment

This work has been partially supported by the EU H2020 Projects ASTRID (Grant Agreement no. 786922) and CyberSec4Europe (Grant Agreement no. 830929).

References

[1] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[2] D. Kreutz, F. M. V. Ramos, P. J. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[3] R. Boutaba and I. Aib, "Policy-based management: A historical perspective," *J. Netw. Syst. Manag.*, vol. 15, no. 4, pp. 447–480, 2007.

[4] C. Islam, M. A. Babar, and S. Nepal, "A multi-vocal review of security orchestration," *ACM Comput. Surv.*, vol. 52, no. 2, pp. 37:1–37:45, 2019.

[5] C. Basile, F. Valenza, A. Lioy, D. R. López, and A. P. Perales, "Adding support for automatic enforcement of security policies in NFV networks," *IEEE/ACM Trans. Netw.*, vol. 27, no. 2, pp. 707–720, 2019.

[6] D. Bringhenti, G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "Automated optimal firewall orchestration and configuration in virtualized networks," in *NOMS 2020 - IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, April 20-24, 2020*. IEEE, 2020, pp. 1–7.

[7] D. Bringhenti, G. Marchetto, R. Sisto, and F. Valenza, "Short paper: Automatic configuration for an optimal channel protection in virtualized networks," in *Proceedings of the 2nd Workshop on Cyber-Security Arms Race, colocated with ACM CCS 2020*. ACM, 2020, p. 25–30.

[8] D. Bringhenti, G. Marchetto, R. Sisto, S. Spinoso, F. Valenza, and J. Yusupov, "Improving the formal verification of reachability policies in virtualized networks," *IEEE Transactions on Network and Service Management*, 2020, in press.

[9] L. M. de Moura and N. Bjørner, "Z3: an efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Hungary, March 29-April 6, 2008*, vol. 4963. Springer, 2008, pp. 337–340.

[10] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2021. [Online]. Available: http://www.gurobi.com

[11] G. M. Yilma, F. Z. Yousaf, V. Sciancalepore, and X. P. Costa, "Benchmarking open source NFV MANO systems: OSM and ONAP," *Comput. Commun.*, vol. 161, pp. 86–98, 2020.

[12] L. Xia, J. Strassner, C. Basile, and D. R. Lopez, "Information model of nsfs capabilities," Internet Draft, RFC Editor, RFC, 09 2019. [Online]. Available: https://tools.ietf.org/id/draft-ietf-i2nsf-capability-05.txt

[13] K. Giotis, Y. Kryftis, and 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.

[14] Y. Bartal, A. J. Mayer, K. Nissim, and A. Wool, "*Firmato*: A novel firewall management toolkit," *ACM Trans. Comput. Syst.*, vol. 22, no. 4, pp. 381–420, 2004.

[15] J. García-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and S. Preda, "MIRAGE: A management tool for the analysis and deployment of network security policies," in *Proc. of the 5th Intern. Workshop, Data Privacy Management and Autonomous Spontaneous Security DPM10*, 2010, pp. 203–215.

[16] M. Rossberg, G. Schaefer, and T. Strufe, "Distributed automatic configuration of complex ipsec-infrastructures," *J. Network Syst. Manage.*, vol. 18, no. 3, pp. 300–326, 2010.

[17] L. E. Li, V. Liaghat, H. Zhao, M. Hajiaghayi, D. Li, G. T. Wilfong, Y. R. Yang, and C. Guo, "PACE: policy-aware application cloud embedding," in *Proceedings of the IEEE INFOCOM 2013, Turin, Italy, April 14-19, 2013*. IEEE, 2013, pp. 638–646.

[18] X. Li and 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.

[19] G. Marchetto, R. Sisto, F. Valenza, J. Yusupov, and A. Ksentini, "A formal approach to verify connectivity and optimize VNF placement in industrial networks," *IEEE Trans. Ind. Informatics*, vol. 17, no. 2, pp. 1515–1525, 2021.

[20] J. Gil-Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 518–532, 2016.