Towards a fully automated and optimized network security functions orchestration

Daniele Bringhenti, Guido Marchetto, Riccardo Sisto, Fulvio Valenza, Jalolliddin Yusupov
Dip. Automatica e Informatica, Politecnico di Torino, Torino, Italy
Emails: {daniele.bringhenti, guido.marchetto, riccardo.sisto, fulvio.valenza, jalolliddin.yusupov}@polito.it

***Abstract*—Automated policy-based network security management tools represent a new research frontier to be fully explored, so as to reduce the number of human errors due to a manual and suboptimal configuration of security services. Moreover, the agility that an automated tool would require can be provided by the most recent networking technologies, Network Functions Virtualization and Software-Defined Networking, which move the network management from the hardware level to the software. However, even though a Security Automation approach is nowadays feasible and would bring several benefits in facing cybersecurity attacks, pending problems are that currently only a limited number of automatic management tools have been developed and that they do not have a direct integration with cloud orchestrators, consequently requiring human interaction. Given these considerations, in this paper we propose a novel framework, whose goal is to automatically and optimally allocate and configure security functions in a virtualized network service in a formal and verified way, directly integrated in cloud orchestrators. We validated this contribution through an implementation that is able to cooperate with two well-known orchestrators, that are Open Baton and Kubernetes.**

***Index Terms*—network security optimization, network security orchestration, cloud security, network functions virtualization**

I. Introduction

Misconfiguration of *Network Security Functions (NSFs)* such as firewalls and VPN terminators has recently become the third most critical exploit for cybersecurity attacks, as Verizon underlined in its most recent Data Breach Investigations Report [1]. This problem is intrinsic of the manual approach by means of which network administrators work, since typically filtering or protection rules are distributed on the NSFs with heuristic and suboptimal criteria based on human common sense [2].

This critical risk motivates the introduction of *automated policy-based network security management tools*: they can assist human beings in the creation and configuration of a security service by means of an automatic process in charge of creating the policy of each NSF so as to respect some security requirements, also called intents, expressing the goals to be compliant with. The advantages of pursuing *Security Automation* are evident: some examples are avoidance of human errors, automatic conflict analysis of the policies and formal verification of their effective correctness. A fundamental benefit is, nonetheless, the possibility to pursue *optimality*: automated algorithms can, in fact, reach the optimal outcomes more easily than manually. This has undoubtedly a critical impact on the resources that are needed to create the effective security service. However, despite all these positive prospects, the research is still moving its first steps towards a fully automated and optimized approach; altogether, the number of developed tools is still limited in this context [3].

Moreover, as demanded by any automatic process, agility is a fundamental requirement to support a similar approach. From this point of view, novel networking technologies such as *Network Functions Virtualization (NFV)* [4] and *Software-Defined Networking (SDN)* [5] can bring a heavy contribution. In fact, the former introduces a decoupling between the computing functions, that are now virtualized as Virtual Machines or Dockers, and the physical general-purpose servers on which they are installed. Then, the latter allows fast creation of the forwarding paths, that different kinds of traffic must follow in the network, by means of a software process. Consequently, in a cloud environment a service can be set up remotely with a very limited latency from the request.

A problem which nevertheless arises in this context is the huge variety of tools that can be used to orchestrate the virtual functions. Although the *European Telecommunications Standards Institute (ETSI)* defined a standard architecture [6], each NFV and cloud orchestrator has different peculiarities and characteristics [7], which reflect the purposes they have been built for and the targets of their developers or vendors. This has a severe impact on the portability of any automatic process which would be in charge of the allocation and configuration of virtual security functions, because it should adapt itself to work with a huge variety of different APIs, input and output data formats.

Given all these considerations, in this paper we propose a fully automated framework called VEREFOO, that stands for *Verified Refinement and Optimized Orchestration*. Its purposes are to refine high-level security requirements, which are expressed with a human-friendly language, into the optimal allocation scheme and configuration of the NSFs on a *Service Graph (SG)* representing the network service [8]. This step is performed in a *correctness-by-construction* fashion, so that these is formal assurance of its correctness; moreover, the computed results are optimal with regard to a set of cost functions, such as minimization of the number of installed functions or number of rules in their policies.

Then, this framework is able to deploy the virtual functions and configure them through a direct interaction with some

orchestrators, without any manual operation. In particular, the two well-known orchestrators that have been integrated with VEREFOO at the moment are Open Baton, that traditionally manages Virtual Machines in an NFV environment, and Kubernetes, which can be also in charge of the orchestration of Dockers in a cloud scenario.

The remainder of this paper is structured as follows. Section II describes the most important related works in these research fields. Section III describes the approach which we followed in the design of the framework and provides an overview on the VEREFOO architecture. In Section IV the implementation is described alongside with its validation and finally Section V provides a brief conclusion to the paper.

II. Related Works

A. Network Security Automation

Automatic security mechanisms have been proposed for firewalls in literature. [9]–[11] define policy-based automatic methodologies to configure firewalls with respect of some security requirements; however, they lack formal assurance of the correctness and they are designed to work in traditional networks with hardware appliances, instead of an NFV or cloud environment. Instead, [12]–[14] enables automation when fixing firewall misconfigurations, exploiting formal verification at the same time; nevertheless, these approaches do not allow the creation of the firewall policies from scratch and, in the case of [12] and [13], they are not still thought for virtualized networks.

On the contrary, automatic configuration of other NSFs has been investigated less extensively. [3] represents the most important work for a policy refinement activity that is not limited to a single kind of function; however, it does not contribute to the creation of a complex security Service Graph, since it is limited to the configuration of function chains.

Then, about the automatic service composition, [15]–[18] contribute to create network services in cloud environments by exploiting novel networking technologies, but without the support of formal verification. Other works [19] [20] establish the optimal firewall allocation in a virtualized service according to some cost functions; even though their purpose is similar to the allocation feature of VEREFOO, however, they do not focus on a larger pool of network functions among which to choose for enforcing the security policies, neither automatically define their configuration.

B. NFV and cloud orchestration

Management & Orchestration (MANO) of the NFV and cloud infrastructure plays a central role in the modern computer networks. Centralizing the deployment, management and monitoring of the *Virtual Network Functions (VNFs)*, which give the possibility to have a complete end-to-end *Network Service (NS)*, simplifies the way the service providers reach their users. Our focus in this paper is on the configuration of the VNFs and the support for Service Function Chaining which a MANO should provide. In the following we analyse the main open source NFV and cloud MANO platforms on the market:

- *Open Source MANO (OSM)* [1] is the prominent framework, proposed by ETSI, whose design currently inspires all the alternative tools for orchestration of VNFs in a virtualized environment. Since it is an open source project, it must be able to interface with a number of functions or platforms that belong to different vendors. For this reason, a research trend is to define novel architectural models which can abstract from the vendor-specific peculiarities of each VNF implementation: examples of this work are modelling languages such as TOSCA and YANG. Nevertheless, the models which support orchestration of security functions in the specificity are few in numbers [21]. OSM offers a multiple *Virtual Infrastructure Manager (VIM)* support, which means it could be used with multiple Infrastructure-as-a-Service technologies (e.g. OpenStack, AWS, OpenVIM, VMware) for the resource orchestration. It allows also to combine them with different SDN Controller technologies (e.g. ONOS, floodlight, ODL) to manage the underlying connectivity. A monitoring system and an experimental support for VNFFG are provided as well. In order to manage the VNFs instance OSM adopts Juju of Canonical as VNF Manager.
- *Open Baton* [2] is an NFV MANO solution compliant with the ETSI NFV MANO specifications. It has a modular architecture, in which a message broker (RabbitMQ) grants the communication between a set of different orchestration and supplementary services. The main services offered are the orchestration of resources, the monitoring system and a set of drivers which allow to use this platform over multiple VIM technologies. In order to extend Open Baton for supporting other VIMs, it is required to create a new driver plug-in and a specific VNFM for this technology. This approach gives an interesting flexibility to the VIM support, indeed, Open Baton has been the first NFV platform to give support to Docker Engine as VIM. So far this platform though does not support SFC or VNFFGs mechanisms.
- *OpenStack Tacker* [3] is an additional service for the OpenStack framework [4]. This service provides NFV Orchestration functionalities (e.g. VNFs/NSs management), leveraging the services included in the OpenStack IaaS platform (e.g. Neutron, Nova, Heat). The main advantage of using this platform is the full support for SFC with the service named networking-sfc and VNFFGs; in the *Network Service Descriptor (NSD)* we could provide an high-level description of FSPs and Classifiers as well.

Other works proposed in the literature on NFV and cloud orchestration, that are worth mentioning, are [22] and [23].

[1] Open Source MANO. [Online]. Available: https://osm.etsi.org
[2] Open Baton. [Online]. Available: https://openbaton.github.io
[3] Tacker. [Online]. Available: https://docs.openstack.org/tacker/latest/
[4] OpenStack. [Online]. Available: https://www.openstack.org

On one side, vConductor [22] is a framework that can construct and monitor virtual enterprise networks with a multi-objective resource scheduling of the VNFs. On the other side, [23] describes a model-based approach that exploits *network function-agnostic* software components such as translators and gateways to install functional configurations into each network function of a complete service that is managed with a framework integrated in a cloud management platform. These frameworks are richer in terms of security functions that can be scheduled, but an automatic policy-based configuration is not integrated, thus requiring a human contribution in the creation of the virtual service; in fact, also [23] only performs a translation of vendor-independent rules instead of a policy refinement. Finally, [24] proposes a modular NFV architecture that permits policy-based management of VNFs, handling they whole life-cycle and exploiting an Information Model to provide an abstraction of network resources, network control functions and VNFs capabilities; however, their limitation is that the only network security capability that is modelled is access control.

III. The VEREFOO framework

In this section, we will illustrate the main principles and purposes of the approach we followed in designing the architecture of VEREFOO; in particular, we will provide a complete overview on the framework, describing the tasks that each module is in charge of and explaining how it has been integrated with the most well-known NFV and cloud orchestrators.

A. VEREFOO approach

VEREFOO manages the creation, configuration and orchestration of a complete end-to-end network security service following a modular approach, that is reflected by the design of the framework itself.

First of all, VEREFOO automatically performs, on a provided *Service Graph*, an optimized allocation and configuration of the *Network Security Functions (NSFs)* that are necessary to fulfill an input set of *Network Security Requirements (NSRs)*, which can be expressed by the service designer – i.e. the person in charge of creating a network service – by exploiting a high-level language. High flexibility is granted by allowing the users to define the NSRs repository and to create the catalog of functions available in the system.

The input Service Graph is made by network functions without any security capability; so, it must be firstly automatically transformed into a logical topology, called *Allocation Graph*, where between any pair of virtual functions an Allocation Place is created. Each Allocation Place is a placeholder position where a NSF could be allocated if it is needed to satisfy the input security constraints. An example is showed in Figure 1, where it is worth underlining that each element, from the NAT to the load balancer, are actually VNFs instead of physical appliance. Then, to establish the optimal allocation scheme of the NSFs and their configuration, a *correctness-by-construction* approach is followed, by the definition of a

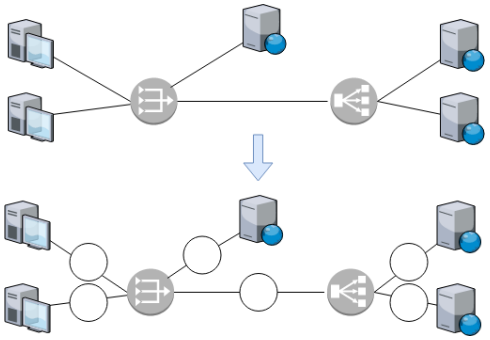

Fig. 1. Example of Allocation Graph generation.

Maximum Satisfiability Modulo Theories (MaxSMT) problem that is in charge of automatically choosing which network functions are needed and where to allocate them; thus, also formal assurance of the correctness is provided. Optimality is undoubtedly the central factor that has been considered to pursue these objectives: actually, the best solution is the scenario where the minimum number of VNFs for security functions are introduced and the minimum number of rules are configured in their policies.

This first step is completely performed on a logical level. Then, after the creation of this virtual security service, a second objective is to establish the optimal placement of each function on the physical servers that compose the substrate network. Other cost functions are considered in this phase, such as minimization of the latency between VNFs or resource consumption. Besides, since each NSF is characterized by policy rules that are expressed with a medium-level language that provides abstraction from the different implementation, a translation is needed to get the vendor-specific configuration of each virtual function.

Finally, the service is set up by means of an integration with cloud orchestrators in order to provide security properties for the communication between end points or networks. It is important to remark that this result is achieved just starting from a Service Graph and a set of security requirements as inputs, thanks to the fully automatic algorithms that are exploited in the overall approach, as it will be more extensively explained in the next subsection.

B. Framework overview

Figure 2 presents a complete overview of the framework, so that we can provide a brief description of each component to give a general idea of the workflow.

According to our vision, the user of the NFV orchestrator - i.e. the service designer - is able to introduce as input:

- a set of Network Security Requirements (NSRs) to express the security constraints which must be fulfilled, by exploiting a high-level or a medium-level language [3] depending on the experience level of the user, through a Policy GUI which makes the creation of the requirements easier;

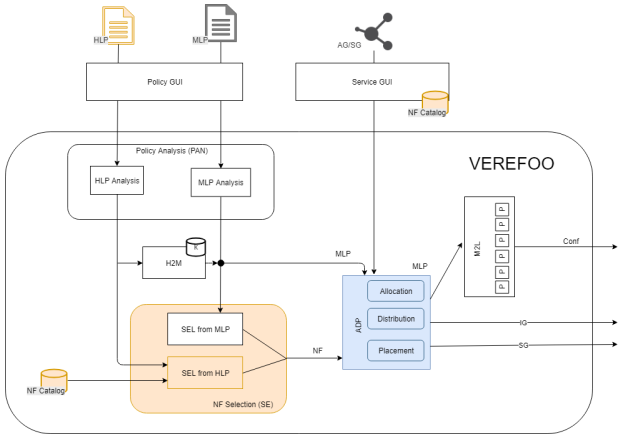

Fig. 2. VEREFOO general architecture.

- a Service Graph (SG) or, in alternative, directly an Allocation Graph (AG) through a Service GUI, which provides access to a Network Functions Catalog (NF Catalog) from which the user can decide which functions – simple network functions or also NSFs – immediately allocate on the graph.

A preliminary phase is represented by the *Policy ANalysis (PAN)*; the goal of this module, which receives the NSRs as input, is to perform a conflict analysis exploiting well-known techniques [25]–[28], to establish if some of the requirements are in conflict, and to create the minimal set of constraints which must be respected in the network. It can provide an early non-enforceability report to the service designer in case the input security requirements are characterized by mistakes which cannot be solved by means of this automatic process but require a reformulation by the user.

If the specified security requirements are expressed in a high-level language, the *High-to-Medium (H2M)* module performs a refinement to get a corresponding set of medium-level NSRs, which contain all the useful information for the future creation of the policies of the NSFs automatically allocated on the virtual graph and the low-level configuration of the VNFs placed on the substrate network.

Then, a key role is covered by the *NF Selection (SE)* module; based on the input high-level and medium-level NSRs, it decides which NSFs are required to satisfy them, choosing them from a pre-built catalog, that is the same list the service designer has access through the Service GUI. This step requires an optimization process by means of which the optimal set of NSFs is selected, even though this operation does not exploit any knowledge about the topology of the Allocation Graph. This result is achieved in the following way: firstly SE receives, from a module outside the framework, the list of the instances of the required capabilities and then it searches, between the functions present in the NF Catalog, which ones support the requested capabilities and selects the optimal functions, taking into account available physical resources, so as to be able to allocate them in the physical servers. The selection of functions is subject to the conditions imposed: in the first place the functions must be able to support the capabilities, but it is also necessary that certain physical resources are available in order to be able to place the functions on the servers. In addition, the choice is subject to optimizations: it is possible to reduce the cost of the functions as well as reduce the amount of RAM needed.

The *Allocation, Distribution and Placement (ADP)* module is one of the main elements of the architecture, whose purpose is to compute a Service Graph with the added NSFs and to decide the VNFs placement on the substrate network receiving as input the medium-level NSRs, the list of selected NSFs and the original Service Graph or directly the Allocation Graph. The ADP module uses *z3Opt* [29] as a MaxSMT solver and Verigraph [30] [31] as a tool for NSRs verification to provide three main features:

- given a list of NSFs selected by the NF Selection module, it orchestrates their allocation on the Allocation Graph — received in input or obtained from the processed Service Graph -– in order to satisfy the input NSRs expressed by means of the medium-level language;
- in contemporary with the allocation phase, a second task is the distribution of the policy rules on the allocated NSFs, always expressed in medium-level language but not necessarily identical to the input NSRs formulation, because the policy rules can be minimized according to optimization goals;
- in a secondary step, after the creation of the Service Graph enriched with the NSFs, the VNFs implementing the network functions of the original Service Graph and the added NSFs are placed in the physical infrastructure following the principle of minimizing the resource consumption and at the same time the medium-level policy rules of the NSFs are translated into the low-level configuration of the VNFs themselves.

An additional output of the ADP element is the list of medium-level policy rules by means of which each network function instance must be configured; then, the corresponding low-level configuration that depends on the specific implementation of the deployed function is generated by the *Medium-to-Low (M2L)* module, which performs a translation of the vendor-independent expressions into the rules which must be set on the proper function.

C. Integration with NFV and Cloud Orchestrators

The security service that has been created by the ADP module of VEREFOO must be then instantiated by creating the virtual processes on the physical servers. For this purpose, communication with orchestrators is needed.

To reach this objective, the complete architecture of the solution we are proposing in this paper does not include only VEREFOO, that represents the logical computing core, but also other tools that work as intermediate interfaces between the orchestrators and VEREFOO itself. Figure 3 shows four examples of all the possible interfaces that can be developed and included in our framework:

- *VeriBaton* can provide integration with Open Baton;

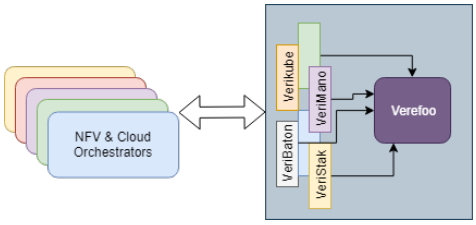

Fig. 3. Integration architecture.

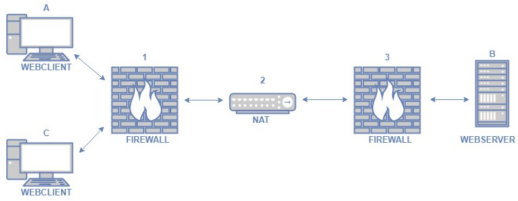

Fig. 4. NSD with two web client nodes, two firewalls, one NAT, and one web server.

- *VeriKube* can provide integration with Kubernetes;
- *VeriMano* can provide integration with OSM;
- *VeriStak* can provide integration with OpenStack Tacker.

Each interface offers RESTful APIs to interact with both the ADP module of VEREFOO, from which it receives all the needed information to create and configure the security service in the virtualized network, and the NFV or cloud orchestrator, that is in charge of the set-up and management of the life-cycle for each VNF. The presence of these interfaces is transparent to the user, who interacts exclusively with VEREFOO as beforehand described, but their role is fundamental. In fact, if all the current orchestrators such as those that have been named are not able to automatically create and configure a network service, this becomes possible by means of the integration with our framework, that is thus achieved without requiring to exploit VEREFOO and the orchestrator separately.

To make clearer how this integration is achieved, in the following section we provide the details of the integration with two interfaces, VeriBaton and VeriKube, alongside with the validation of the proposed approach through various use cases in Open Baton and Kubernetes.

IV. Implementation and Validation

In order to meet functional requirements, different design approaches of the VEREFOO framework have been considered, taking into account various use case scenarios and used service orchestrator capabilities and characteristics. Service orchestrators are a relatively new technology, yet different products are already available from open-source communities (e.g., Juju, Open Baton, Kubernetes, MAESTRO). The implementation of the VEREFOO use cases will integrate the framework in existing orchestration frameworks, such as Open Baton and Kubernetes.

A. Open Baton VEREFOO integration

A contribution to Open Baton project has been the most flexible solution considered, as deep integration with the MANO architecture. The main goal of this integration is to extend the capabilities of the Open Baton orchestrator in terms of Service Graph verification and validation, by integrating the tool VEREFOO. This solution allows to interact with the orchestrator introducing graph validation and optimization in the phase of Network Service catalog upload, thus allowing onboarding of a Service Graph only after a validation check, rejecting formally invalid descriptors, and possibly updating the graph to optimize configurations and service design. To achieve this, we meet the following principles:

- *Interface compatibility*: the interface used to interact with VEREFOO service should match completely the interface exposed by Open Baton for NSD onboarding. In this way, the end user can be unaware of VEREFOO presence if not interested, and behave as it was interacting with Open Baton; network services can be developed following ETSI data model, making VEREFOO validation capabilities pluggable at the user discretion. VEREFOO becomes this way a sort of ”proxy” which could be used depending on the needs, with NSD instances built directly for Open Baton. As communication with Open Baton happens through a REST interface over HTTP, VEREFOO will be itself a RESTful API server.
- *Input validation*: VEREFOO acts as a validator component for the input provided to Open Baton, implying that an invalid Service Graph will be blocked before reaching Open Baton with suitable feedback for the user. It is in charge of verifying that nodes are correctly organized in a chain, and policies specified as input such as reachability and isolation between VNFs are satisfiable, assuring that a service present in the catalog once deployed does behave as expected.
- *Graph optimization*: once received the optimal service configuration from VEREFOO, the original input should be modified according to it. Possible scenarios include removal of nodes from the graph and automatic configuration of elements such as firewalls, which should be reflected on the NSD to be uploaded to Open Baton.

To verify the capabilities of the framework, we designed an NSD instance representing a common use case of the tool and feed them as input to VEREFOO. Figure 4 describes visually the configuration of the test instance including two web client nodes, two firewalls and one NAT VNFs connecting to a web server. The requirements of the service request are:

- reachability is required between node A and node B;
- isolation is required between node C and node B;
- node 1, 2 and 3 are optional and are not configured.

Once Open Baton (ETSI) compliant NSD description of the service is provided as an input to the orchestrator, we expect to minimize the number of NSFs, while satisfying these user requirements. As a result, we obtain a report that the service validation is successful. Moreover, the Service Graph has been updated as expected, where node 3 has been removed, while

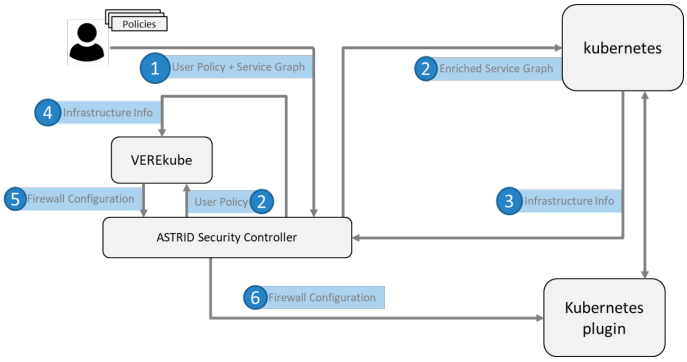

Fig. 5. Overall workflow of the Kubernetes integration.

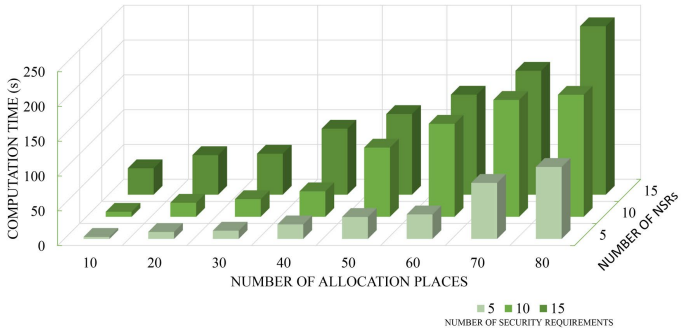

Fig. 6. Results of scalability tests.

node 1 has been configured to allow traffic directed from node A to node B, denying everything else.

This validates the proposed approach, successfully presenting a solution capable of translating the information model based on the ETSI specifications to the application-specific format defined by the verification engine albeit the substantial differences in data representation and structure between VEREFOO and Open Baton orchestrator.

B. Kubernetes VEREFOO integration

Kubernetes is an open source system for automating deployment, scaling, and management of containerized applications. Kubernetes enables to quickly deploy containerized applications, scaling it according to the user needs, without having to stop anything in the process. It is made to be portable, extensive and self-healing, granting an easier management from people who have to administrate the system. The Kubernetes orchestrator is the second orchestrator used for demonstration and validation. We integrate the VEREFOO framework with Kubernetes to provide lightweight monitoring and enforcement hooks in each virtual function, which can be dynamically programmed, to react to management events and security alerts, by invoking specific security services. In this integration VEREFOO takes as input the service topology, the current network configurations, and the security policies, and returns as output the configuration of the security hooks. In the current implementation, the scope is limited to automatic firewall configuration. Figure 5 presents the general workflow of the VEREFOO integration with Kubernetes orchestrator. Our implementation involves a number of components, and it starts with user delivering policies and Service Graph to the controller, at this time the enriched Service Graph with all the information model required by Kubernetes is defined and delivered. At this point security controller sends the user policy to VEREFOO. Kubernetes provides the infrastructure information based on the deployed graph to the controller, which is then delivered to VEREFOO. In the next step, VEREFOO computes formally verified configuration parameters of the firewalls, in order to satisfy the user policies and delivers them to the controller, which then sends it back to context broker. Context broker is in charge of enforcing the firewall rules in the NSFs.

In this context, the results of some performance tests carried out on the introduced integration are illustrated, in order to show which goals have been achieved and to understand which limitations should be refined in the future. We focus on two metrics – numbers of Allocation Places and of Network Security Requirements –, to perform the scalability tests by increasing one metric to an higher value, while keeping the others fixed, to understand to which extend the first metric is scalable. Given this assumption, the results of the performance tests which have been carried out to understand the scalability of the developed framework are showed in Figure 6 for the Allocation Places and for the Network Security Requirements. They have been achieved with a machine characterized by a 3.40 GHz Intel i7-6700 CPU and 32GB of RAM.

This chart shows that the computation time does not increase exponentially either with the number of Allocation Places or with the number of NSRs. This result is particularly positive, given the intrinsic worst-case computational cost of a MaxSMT problem, that belongs to the NP-complete class. Consequently, the framework is able to manage Service Graphs of medium-big dimensions, with a high number of links and end points, providing the optimal solution to the presented problem, if the number of Network Security Requirements is not extremely high. These results clearly show that the network structure and the number of security requirements strongly influence time performance, but provides us a strong hint about the fact the the approach we are following is feasible and worth to be further explored.

V. Conclusion and Future Works

This paper presents a novel framework to provide an automated and optimized orchestration of NSFs in an NFV and cloud environment, by exploiting the benefits of the recent virtualization techniques in the networking field. The optimal allocation and configuration of the virtual functions is performed through the integration with some well-known orchestrators – e.g. Open Baton and Kubernetes – so that no further manual operation is needed.

As possible future works, we are currently working to enrich the methodology on which the framework design is

based, by introducing the formal model of a larger set of NSFs (e.g. anti-spam filters and intrusion detection systems) to be automatically configured and other kinds of security requirements that can be expressed as input by the user.

Moreover, we are planning to extend the integration of VEREFOO with other NFV and cloud orchestrators, in order to achieve our ultimate goal to have a fully integrated platform that could work with most of the current orchestrators, since the validation we achieved with our current implementation provided us satisfactory results about the path we are pursuing.

ACKNOWLEDGMENT

This work was supported in part by the European Commission, under Grant Agreement no. 786922.

REFERENCES

[1] Verizon, "Data Breach Investigations Report," 2019.

[2] K. Popovic and Z. Hocenski, "Cloud computing security issues and challenges," 06 2010, pp. 344 – 349.

[3] C. Basile, F. Valenza, A. Lioy, D. R. Lopez, 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.

[4] 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.

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

[6] "Network function virtualization - White Paper 2," The European Telecommunications Standards Institute, Tech. Rep., October 2013.

[7] R. Mijumbi, J. Serrat, J. Gorricho, S. Latré, M. Charalambides, and D. López, "Management and orchestration challenges in network functions virtualization," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 98–105, 2016.

[8] J. Zhang, Z. Wang, N. Ma, T. Huang, and Y. Liu, "Enabling efficient service function chaining by integrating NFV and SDN: architecture, challenges and opportunities," *IEEE Network*, vol. 32, no. 6, pp. 152–159, 2018.

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

[10] P. Verma and A. Prakash, "FACE: A firewall analysis and configuration engine," in *2005 IEEE/IPSJ International Symposium on Applications and the Internet (SAINT 2005), 31 January - 4 February 2005, Trento, Italy*, 2005, pp. 74–81.

[11] J. D. Guttman, "Filtering postures: Local enforcement for global policies," in *1997 IEEE Symposium on Security and Privacy, May 4-7, 1997, Oakland, CA, USA*, 1997, pp. 120–129.

[12] N. B. Youssef and A. Bouhoula, "A fully automatic approach for fixing firewall misconfigurations," in *11th IEEE International Conference on Computer and Information Technology, CIT 2011, Pafos, Cyprus, 31 August-2 September 2011*, 2011, pp. 461–466.

[13] K. Adi, L. Hamza, and L. Pene, "Automatic security policy enforcement in computer systems," *Computers & Security*, vol. 73, pp. 156–171, 2018.

[14] A. Gember-Jacobson, A. Akella, R. Mahajan, and H. H. Liu, "Automatically repairing network control planes using an abstract representation," in *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, 2017, pp. 359–373.

[15] E. J. Scheid, C. C. Machado, M. F. Franco, R. L. dos Santos, R. J. Pfitscher, A. E. S. Filho, and L. Z. Granville, "Inspire: Integrated nfv-based intent refinement environment," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, Portugal, May 8-12, 2017*, 2017, pp. 186–194.

[16] Y. Han, J. Li, D. Hoang, J. Yoo, and J. W. Hong, "An intent-based network virtualization platform for SDN," in *12th International Conference on Network and Service Management, CNSM 2016, Montreal, QC, Canada, October 31 - Nov. 4, 2016*, 2016, pp. 353–358.

[17] A. S. Jacobs, R. J. Pfitscher, R. A. Ferreira, and L. Z. Granville, "Refining network intents for self-driving networks," *Computer Communication Review*, vol. 48, no. 5, pp. 55–63, 2018.

[18] Z. Hao, Z. Lin, and R. Li, "A sdn/nfv security protection architecture with a function composition algorithm based on trie," in *Proc. of the 2Nd International Conference on Computer Science and Application Engineering*, ser. CSAE '18, 2018, pp. 176:1–176:8.

[19] M. Yoon, S. Chen, and Z. Zhang, "Minimizing the maximum firewall rule set in a network with multiple firewalls," *IEEE Trans. Computers*, vol. 59, no. 2, pp. 218–230, 2010.

[20] M. A. Rahman and E. Al-Shaer, "Automated synthesis of distributed network access controls: A formal framework with refinement," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 2, pp. 416–430, 2017.

[21] "Network Functions Virtualisation (NFV): Management and Orchestration (GS/NFV-MAN-001)," The European Telecommunications Standards Institute, Tech. Rep., 2014.

[22] W. Shen, M. Yoshida, K. Minato, and W. Imajuku, "vconductor: An enabler for achieving virtual network integration as a service," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 116–124, 2015.

[23] S. Spinoso, M. Leogrande, F. Risso, S. Singh, and R. Sisto, "Seamless configuration of virtual network functions in data center provider networks," *J. Network Syst. Manage.*, vol. 26, no. 1, pp. 222–249, 2018.

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

[25] E. Al-Shaer, H. H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 10, pp. 2069–2084, 2005.

[26] F. Valenza, S. Spinoso, C. Basile, R. Sisto, and A. Lioy, "A formal model of network policy analysis," in *2015 IEEE 1st International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, Sep. 2015, pp. 516–522.

[27] F. Valenza, C. Basile, D. Canavese, and A. Lioy, "Classification and analysis of communication protection policy anomalies," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2601–2614, Oct 2017.

[28] C. Basile, D. Canavese, A. Lioy, and F. Valenza, "Inter-technology conflict analysis for communication protection policies," in *Risks and Security of Internet and Systems*, J. Lopez, I. Ray, and B. Crispo, Eds. Cham: Springer International Publishing, 2015, pp. 148–163.

[29] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *Proc. of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 337–340.

[30] G. Marchetto, R. Sisto, J. Yusupov, and A. Ksentini, "Virtual network embedding with formal reachability assurance," in *14th International Conference on Network and Service Management, CNSM 2018, Rome, Italy, November 5-9, 2018*, 2018, pp. 368–372.

[31] G. Marchetto, R. Sisto, F. Valenza, and J. Yusupov, "A framework for verification-oriented user-friendly network function modeling," *IEEE Access*, vol. 7, pp. 99 349–99 359, 2019.