Adding Support for Automatic Enforcement of Security Policies in NFV Networks

Cataldo Basile, Fulvio Valenza, Antonio Lioy, Diego R. Lopez, and Antonio Pastor Perales

***Abstract*— This paper introduces an approach toward the automatic enforcement of security policies in network functions virtualization (NFV) networks and dynamic adaptation to network changes. The approach relies on a refinement model that allows the dynamic transformation of high-level security requirements into configuration settings for the network security functions (NSFs), and optimization models that allow the optimal selection of the NSFs to use. These models are built on a formalization of the NSF capabilities, which serves to unequivocally describe what NSFs are able to do for security policy enforcement purposes. The approach proposed is the first step toward a security policy aware NFV management, orchestration, and resource allocation system—a paradigm shift for the management of virtualized networks—and it requires minor changes to the current NFV architecture. We prove that our approach is feasible, as it has been implemented by extending the OpenMANO framework and validated on several network scenarios. Furthermore, we prove with performance tests that policy refinement scales well enough to support current and future virtualized networks.**

***Index Terms*— NFV, policy refinement, NFV management and orchestration, network security, security policy.**

I. Introduction

CURRENTLY two technologies seem to have the power to significantly change computer networks: Network Functions Virtualization (NFV) [1] and Software-Defined Networking (SDN) [2]. NFV proposes a virtualized infrastructure where network functions are implemented by software appliances, named Virtual Network Functions (VNF), that are decoupled from physical network devices. SDN introduces the possibility to dynamically reconfigure the network, by selecting arbitrary paths and redirecting traffic through specific middleboxes. These technologies are in synergy. NFV fosters flexible and programmable network function deployment and ease of scaling, and its adoption is expected to reduce administration tasks, response times, and TCO (Total Cost of Ownership). SDN enables a more efficient use of resources, as selected traffic can be dynamically redirected to virtual and physical network functions, e.g., per-user and per-flow basis.

Flexible management is one of the main advantages of these new networking paradigms. However, in NFV flexibility cannot be extended to security controls, named (virtual) Network Security Functions (NSFs),[1] as the enforcement of security policies requires careful adaptation every time a change is made. In fact, while networking elements and protocols are designed to automatically adapt to changes, dynamic adaptation is not always a design principle for NSFs. Therefore, in this paper we propose to take advantage of the flexibility of networking management to also achieve flexible enforcement of security policies. The research in this paper addresses two important challenges, as highlighted by works literature [1], [3]. First, it supports the *automatic enforcement of security policies*. Given a target network and a set of security requirements expressed with a high-level language (i.e., the *security policy*), our model supports the configuration[2] of all the security functions in the target NFV network so that the security policy is enforced. Our model identifies and reports non-enforceability issues when the security policy requires security functions not available in the target network (e.g., to filter URLs without having a HTTP filter). Second, we support *dynamic adaptation to network changes*. We enable administrators to react and maintain the security policy enforced in case of any topological change of the network, like NSF failure or substitution of a NSF with another one that implements the same security functions (e.g., substitution of a firewall with a different one from another vendor with equivalent filtering capabilities), and any addition/removal of VNFs or NSFs. More precisely, given 1) a set of security requirements expressed with a high-level security policy language, 2) an initial network already configured to enforce this policy, and 3) a target network (different from the initial one), our model allows the configuration of all the NSFs so that the security policy is enforced in the target network. Non-enforceability issues are reported in this case too.

The long term objective of our research is to support a broadest scenario: *security policy aware network management and orchestration* (SPA-MANO), where only the functional and security requirements are specified, while the network topology and the security functions that optimally implement the security policy are automatically decided and allocated in a way that is transparent to the administrators.

Manuscript received September 23, 2016; revised June 4, 2017, January 18, 2018, June 15, 2018 and September 13, 2018; accepted January 18, 2019; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Guan. This work was supported by the SECURED and SHIELD Projects under Grant 611458 and Grant 700199. *(Corresponding author: Fulvio Valenza.)*

C. Basile, F. Valenza, and A. Lioy are with the Dipartimento di Automatica e Informatica, Politecnico di Torino, 10129 Turin, Italy (e-mail: cataldo.basile@polito.it; fulvio.valenza@polito.it; antonio.lioy@polito.it).

D. R. Lopez and A. Pastor Perales are with Telefónica I+D, 28050 Madrid, Spain (e-mail: diego.r.lopez@telefonica.com; antonio.pastorperales@telefonica.com.

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/TNET.2019.2895278

[1] https://datatracker.ietf.org/wg/i2nsf/charter/

[2] Configuring a NSF is the process that defines a set of low-level settings that, once deployed, determines the behavior of the security controls.

We address the two challenges above in NFV networks by adding a new component, the Security Awareness Manager (SAM), as a part of the Orchestrator. Indeed, the two scenarios are very similar and we addressed them by means of *policy refinement*.[3] The SAM is in charge of executing the policy refinement when the network is initialized and every time a change is detected into the network or into the security policy. Our policy refinement approach also includes a decision phase where an *optimization model* allows selecting the best way to refine the policy in the target network, according to (network and device) performance, and security. Although other works exist on policy refinement, to the best of our knowledge there are no initiatives that propose to support automatic security policy enforcement in NFV with policy refinement.

In addition to the refinement and optimization models, this paper presents other contributions. To make policy refinement feasible we have developed a solution to a known open issue[4]: to find an unequivocal method to represent the security features that an NSF could provide for policy enforcement purposes. We propose the *capability model*, which is presented in Section III, where capabilities are a compact yet precise way to determine which policies an NSF can implement, as they describe the actions that can be enforced and the conditions for their enforcement. Moreover, capabilities describe other policy-related aspects that a refinement system must know when generating configurations for target NSFs (like the supported resolution strategy or how to set the default action). Other works related to a capability model exist, however, they address a similar issue but only for traffic flow control [6], or model virtual resources for allocation purposes [7]. Instead, we specifically model NSFs capabilities.

On top of the capability model, we have built the Medium-Level Policy Language (MLP), an abstract language to represent NSF configurations with a vendor-independent but capability-aware format (see Section IV-C), another known open issue in literature [1].

Finally, we have developed a High-Level Policy language (HLP), which allows easy specification of security requirement at a high-level of abstraction. The HLP has been defined to specify the security policies useful in our use cases. Other high-level languages exist in literature, mainly for access control policy specification (see Section VIII-C), but, as in our case, they only solve problems related to specific use cases.

We tested the validity of our approach by specifying security requirements and automatically enforcing them in NFV networks deployed with OpenMANO. First, we proved that adding the SAM in a NFV architecture is straightforward and effective, as we easily included the SAM in OpenMANO. Then, we used the SAM to enforce security requirements into a set of target networks, by refining policies into configurations for all the NSFs. Both security requirement and target networks have been taken from the use cases of the EU project SECURED, thus we proved the applicability to real cases (see Section VII-B). Moreover, we have performed in-depth scalability testing on synthetic-yet-realistic networks and we have found promising results (see Section VII-C).

Note that, we don't think that our models can be directly adopted by working groups or used in practice as they are, as several other low-level and management aspects need to be considered at large. However, they were essential to prove that using policy refinement for easy management of VNF is feasible, and it can be done by using a capability model and an abstract representation of the policy.

The paper is structured as follows. Section II introduces our approach, the SAM, and shows how to implement it in the current NFV architecture. Section III presents the foundations of our capability model and the algebra for composing capabilities. Sections IV and V discuss the models to refine HLP policies and allow optimal selection of NSFs to use for policy enforcement. Section VI sketches how dynamic adaptation to network changes is managed with our approach and Section VII describes the implementation of our approach and the results of the performance analysis. Section VIII presents the most important related work in three areas, NFV management and orchestration, resource optimization, and policy refinement. Finally Section IX draws conclusions, discusses on the applicability of this approach and depicts the next steps toward a security policy aware MANO.

II. Approach

We now discuss that supporting automatic enforcement of security policies and dynamic adaptation to network changes is a feasible task that requires minor changes to the current NFV architecture.

A. Background

ETSI has defined the Management and Orchestration Framework (MANO) to provide the required functionality for provisioning and configuring VNFs as well as the configuration of the infrastructure where these functions must run. MANO also provides orchestration and life cycle management of VNF resources. We shortly sketch here the main features, while further details, which are not needed for this treatment, can be found in a survey from Mijumbi *et al.* [1].

The main NFV MANO components are the Virtual Infrastructure Manger (VIM), the VNF Manager (VNFM), and the NFV Orchestrator (NFVO). The VIM manages and controls a NFV Infrastructure (NFVI). NFVIs are combination of physical and virtual resources where VNFs are deployed. A NFV architecture may contain many VIMs, if there are more NFVIs provided by more infrastructure providers. The VNFM ensures the correct life cycle management of VNFs, which include instantiation, scaling (increase or reduce the resources associated to a VNF), and termination. The VNFM may resort to a (legacy) component, the Element Management system (EM), to manage selected VNFs. NFVO orchestrates resources and services. Moreover, it may allocate end-to-end services by composing VNFs [8].

The MANO may access information from ad hoc repositories that store data needed for its operations (e.g., the Network Service catalogue (NS) and the VNF catalogue). However, it may also collect additional data as Network

[3]Policy refinement is the process that takes as input a policy at high-level of abstraction and transforms it into a low-level concrete one [4].

[4]Other works [1], [5] report the same issue, moreover the I2NSF IETF Working Group aims at identifying a solution for the same problems.

Service Descriptors (NSDs). A NSD consists of static information that describes deployment flavors of Network Service and potential dependencies, like the VNF Forwarding Graphs, Virtual and Physical Network Functions, and Virtual Links.

Even if the current NFV specifications state that VNF Forwarding Graph information elements contain Network Forwarding Path elements that are used to route traffic "which in turn include policies (e.g. MAC forwarding rules, routing entries, etc.) and references to Connection Points (e.g. virtual ports, virtual NIC addresses, etc.)" [8], there are no examples of Network Forwarding Path in the standard appendices. Moreover, the tools that we have examined, namely OpenMANO and OPENBATON, do not support forwarding information. That is, the network services that can be specified are chains of VNFs. ETSI proof of concept 28 (POC#28) on NFV is looking for effort toward SDN Controlled VNF Forwarding Graph, but it is an ongoing activity. Since we internally use abstract service graphs to represent networks for policy refinement purposes, this lack of available standard representations does not affect our work.

For our experimentations, we have specified target networks as a set of NSDs, each one describing a chain, connected by means of external SDN routing information by OpenFlow rules. These rules have also been used to route the traffic of our sample client nodes toward the network services.

B. Reference Scenario

For the sake of clarity, we introduce a simplified scenario to be used whenever an example is needed to contextualize our approach.

A Network Service Provider (NSP) provides network and security services for its customers. The NSP uses five categories of NSFs: packet filters (PF), web application firewalls (WAF), parental controls (PC), virtual private networks (VPN), and network security monitors (MON). The NSP uses a single application/control for each category of NSFs, namely: PF is implemented with Iptables; WAF is enforced by Squid; PC is implemented by E2gardian; VPN uses IPsec+IKE features implemented by a Strongswan instance; MON is implemented by Bro. NSFs are provided as service function chains.

A company buys the network and services offered by the NSP and wants three security requirements enforced, expressed here as HLP-like statements: 1) "Enable malware detection in PDF attachments in corporate email"; 2) "Allow employees to connect to social networks at lunch"; 3) "Protect confidentiality of traffic between Turin and Madrid offices".

Additional information about this scenario will be discussed in Section VII-B, where we present the validation of our approach, and in the supplementary downloadable material, where we report all the input and output data necessary to validate our approach on this example in the NFV context.

C. The Security Awareness Manager Architecture

To have a security policy aware orchestration, we propose the use of a new component, which we name Security Awareness Manager (SAM). The SAM transparently enforces user security requirements by providing an additional layer between the administrator and the NFV orchestrator that performs the following key features: 1) interpretation of the security policy and identification of the NSFs that may enforce the policy, 2) generation of the configurations of the NSFs that will implement the security policy, and 3) invocation of the standard MANO management operations to deploy the generated configurations.

In our implementation, the SAM is a component that provides additional features to the NFVO but whose functionality logically pertains it. However, in the future, the SAM could be part of the NFVO. In practice, the SAM is a component that has to be connected to the NFVO to provide its policy-related services, but it is immaterial, with the current NFV standard, whether it is an internal module of a security-aware NFVO or a separate module that is part of an external framework.

The SAM requires a dedicated repository to perform its operations: the Policy Repository (PR). In principle, the PR will be in charge for storing all the policy abstractions needed for security policy awareness purposes, including the input security policies and the output NSF configurations. We will show in the next sections that in our policy refinement approach, the PR stores the input policies (expressed with HLP), the abstract NSFs configurations (in MLP), and the concrete configurations of all the needed NSFs (expressed in their own configuration languages). We also store in the PR all the intermediate data structures in the NSDs (used for policy enforceability purposes) and the reports produced by the refinement process (to inform the administrators in charge for configuring the NFV architecture and enforcing the policy). Furthermore, the SAM accesses an internal *Knowledge Base (KB)* to store the semantic of high-level concepts, that is, the mappings between high-level HLP concepts to the network-related low-level information needed to implement the policy (e.g., network topology info, addresses, black lists). For instance, in our example, we have introduced mappings to associate "social networks" with the URLs of social network web sites, "Turin offices" with the public IP addressed of the Turin offices, and "corporate email" is associated to MIME objects from/to the mail server defined with IP, ports and protocols. Identifying these mappings is a laborious yet easy task. To implement our prototype, we followed a bottom-up approach; we have written the configurations of several security controls able to enforce the statements allowed by the HLP, found the involved concepts, and added into the KB the proper mappings. Unfortunately, every time that a new HLP concept or new types of statement is introduced in the HLP, new relations could be needed by the KB. However, we expect open source initiatives or companies interested in commercializing products to perform these maintenance operations. Analogously, while we manually populated the KB for our experiment (i.e., by defining all the mappings), we expect that, in future, filling in the KB will be (almost completely) performed by including external sources of information, such as corporate directory services, DNS data, virus DB, and URL lists.[5] Finally, the SAM retrieves the network and VNF informations from the VNF repository.

We have integrated and tested the SAM in OpenMANO, an open source project led by Telefónica, which provides one

[5]For instance, SquidGuard refers to http://www.squidguard.org/blacklists.html

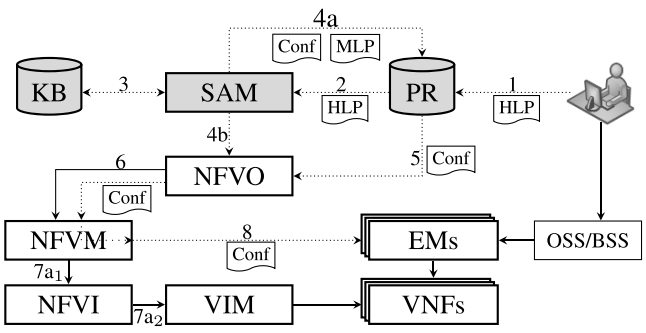

Fig. 1. Automatic policy enforcement in NFV: workflow.

of the most up to date open source implementations of the NFV MANO reference architecture. OpenMANO has been preferred due to its modular architecture, which has allowed us a smooth insertion of the new functionalities. Furthermore, OpenMANO is actually used by Telefónica to manage, deploy and test different types of VNFs in their labs, and currently evolving into a complete MANO solution supported by a wide community through the OSM project. Thus OpenMANO is a working software, not only a prototype, and it is currently maintained and extended by Telefónica. Moreover, recently OpenMANO has become the starting point of Open Source MANO, the official ETSI NFV reference architecture [9]. The integration of the SAM to OpenMANO has shown a promising impact on policy-based NFV management. The results in Sections VII show that adding support for policy awareness does not significantly delay the NFVO operations.

D. The Security Awareness Manager Workflow

Fig. 1 shows the SAM and the new storages (in gray) together with the ETSI NFV components (the other white boxes), and the interactions among them. The adoption of our approach imposes additional steps to the ETSI NFV workflow, which are represented as dotted arrows (the solid ones are already foreseen by NFV). Further details on the steps are presented in the next sections.

The workflow starts with a user writing his own security requirements as an HLP policy (step 1), which is stored in the PR. The creation of the user policy triggers the following steps. The SAM, after being notified about the availability of a new policy, retrieves the HLP policy (step 2). The SAM, by using the information in the KB (and information about target network, that is the NSDs and SDN information as OpenFlow rules), identifies the capabilities NSFs needed to enforce the HLP policy then it selects the best set of NSFs in the target network to enforce the identified capabilities. If the network does not contain NSFs able to enforce the policy, the user is informed about the non-enforceability issue and a log is generated. If the policy is enforceable, the SAM first refines the HLP policy into a set of abstract MLP policies for all the selected NSFs (with the mapping stored in the KB) then it translates them into the corresponding NSF configurations (step (3), Sections IV-D, IV-E, and V). The SAM stores the MLP and the configurations in the repository (step (4a)), and notifies the NFVO that new information is available into the PR ((step (4b))). The NFVO retrieves the NSF configurations from the PR and the NSD from the standard repositories (step (5)). The NFVO contacts the NVFM to instantiate the selected NSFs through the VIM and passes it the configurations (step (6)). The actual instantiation is performed by the NFVI and VIM (steps 7a and 7b). The actual configuration of the network functions is pushed by the NFVM through the Element Managers (EM) of each involved function (step (8)).

Note that, if our approach is not used, the configurations for all the NSFs need to be manually written and pushed through Operation and Business Support Systems (OSS/BSS) by the administrators.

E. Advantages of the SAM

Currently, only the resources to be assigned to the each NSF (e.g., memory size, processor utilization, storage) and available in the VNF descriptors are considered for the allocation of VNFs. However, a security policy aware MANO, given the information generated during the refinement process by the SAM, may use algorithms that allocate the NSFs of all the users (and customers) that work better than the existing ones, because more data are available to base their decisions. For instance, if users need to perform packet filter, a SPA-MANO may decide to add the rules that enforce the user policy into an existing Iptables NSF and forward traffic with SDN, or if users ask (and pay) for isolation, it may allocate a separate NSF. This decision may be led by the resources available at the nodes, performance considerations, on the network delays introduced to forward the traffic, the workload of the NSF (which will also include the number and type of rules to enforce). Moreover, a SPA-MANO may decide based on more precise performance models of security functions, which may also consider the number of the rules they have to implement (which are currently available for some packet filters [10]) and the condition types of each rule (e.g., if rules use conditions as regular expressions [11]). As an example, this information can be used to determine how many rules to insert before deciding to allocate an additional NSF, as for some functions, performance is stable until a threshold is reached, then it dramatically increases. Even if it is not in the scope of this paper, optimized allocation algorithms that also consider policy-related information is an interesting research area we are considering for future work.

Finally, another important aspect that is outside the scope of this research paper but needs to be addressed before using this refinement system in the current NFV implementations, is determining the proper order of configuration of the NSFs. Indeed, the glitches arising from transition from a policy configuration to a new one can lead the target network to inconsistent states that may have devastating effects on both functionality and security. We are not aware of existing systems able to perform this task, research should focus on extending the features of the VIM.

III. Capability Model

As human beings, administrators and network security experts understand each other when referring to security controls by just naming categories. For instance, experts unequivocally refer to “packet filters” as stateless devices able to drop packets based on conditions on source and destination

IP addresses, source and destination ports, and IP protocol type fields [12]. Moreover, it is known that packet filter rules are prioritized[6] and it is often possible to specify a default action.

However, more information is needed to better clarify the behavior of, for instance, stateful firewalls or application layer filters. Controls that fall in these categories may show substantial differences (e.g., depending on the vendor/product) on how they classify packets and communications to determine when to apply decisions, maintain the stateful information, and process protocols. That is, they are in the same category but they cannot be used interchangeably. Also communication protection protocols are usually considered as a single category. Indeed, all of them protect packets[7] with symmetric algorithms whose keys could have been negotiated with asymmetric cryptography, and have a clear behavior defined by standards. However, each protocol has its own peculiarity, e.g., it works at a different ISO/OSI layer and support different encryption algorithms, and authentication, key agreement and negotiation abilities. Moreover, implementations of these protocol show further minor differences. Therefore, even in this case, no interchangeability is ensured.

For this reason, a capability model is needed to precisely and unequivocally clarify what a security function can do in term of security policy enforcement. With such a model, automatic systems, like the refinement engine we are proposing, and human beings can understand 1) whether a function is suitable or not for implementing a high-level policy and 2) how it can be configured to enforce such high-level security policy.

To build our capability model,[8] we have analyzed several NSFs that fall in different categories, including packet filter, URL filter, HTTP filter, VPN gateway, anti-virus, anti-malware, content filter, monitoring, anonymity proxy. Moreover, we have also analyzed common extensions of the generic NSFs, like optionally loadable modules (e.g., iptables supports the addition of modules with the `-m` option, some module is standard "state" and included in the base distribution, some needs to be explicitly downloaded and installed, like "time"). Therefore, based the geometric model of policies [13], we have modeled the capabilities of a NSF in two parts: 1) rule-based capabilities, which describe how rules can be written in terms of actions and conditions (see Section III-A); 2) policy-based capabilities, which describe how to write a coherent policy given a set of rules specified according to the rule-based capabilities (see Section III-B). We have identified a list of rule-based and policy-based capabilities (presented in the the supplemental material provided with this paper). Nevertheless, our model is extensible as additional capabilities can be added if required to describe new security functions. Our capability model is provided with a set operations to manipulate capabilities by means of a capability algebra (see Section III-C).

[6]More precisely, packet filters implement the First Matching Rule (FMR) or Last Matching Rule (LMR) resolution strategies.

[7]To ensure protection, these protocols check header or payload integrity, optionally apply confidentiality, anti-reply protections, and authenticate peers.

[8]Preliminary material about capabilities has been published as an IETF I2NSF draft https://tools.ietf.org/html/draft-baspez-i2nsf-capabilities-00 and revived in https://datatracker.ietf.org/doc/draft-xibassnez-i2nsf-capability/.

A. Rule Based Capabilities

Security functions are able to enforce actions and own traffic classification features. *Actions* ($\mathcal{A}$) describe the operations that a security function can perform on packets/flows, like filtering or encrypting traffic, and operations that are not performed directly on the traffic, like logging matching rules or notifying events. We assume that all the actions available at a security function are well known and organized in the *NSF action set*. For instance, the action set of a generic packet filter will include the Allow and Deny actions. Communication protection functions will have action sets that depend on the technology. For instance, NSFs based on IPsec will have action sets that include actions expressing the protocol (AH vs. ESP), and the mode (tunnel vs. transport mode).

Classification features ($\mathcal{C}$) concern the possibility of the security function to identify target packets/flows on which the actions could be enforced. Classification features are specified by means of conditions clauses that are logical formulas of conditions [13]. Conditions are typed predicates defined over given selector. A selector is an abstract representation of the values that a protocol field may take, e.g., the IP source selector is the set of all possible IP addresses, and the part of the packet where the values come from, e.g., the IP source selector refers to the IP source field in the IP header. Moreover, selectors may imply a list of allowed values, such as the states associated to a traffic (like the NEW, ESTABLISHED, RELATED, INVALID states in iptables), or implicitly determine the type of data (like the strings of characters for HTTP protocol fields). In practice, a condition on a given selector matches a packet/flow if it is evaluated to true with the values extracted from the packet/flow. A condition clause matches a packet/flow if its formula is evaluated to true. Since in most cases formulas are represented in disjunctive normal form, a condition clause matches a packet/flow if all the conditions are true, otherwise a Boolean expression needs to be evaluated.

We have categorized the types of selectors in exact match, range-based, regex-based, and custom-match [11].

Exact match selectors are (unstructured) sets: elements can only be checked for equality, as no order is defined on them. As an example, the protocol type field of the IP header is a unordered set of integer values associated to protocols.

Range-based selectors are ordered sets, thus easily mapped to integers, where it is possible to naturally specify ranges. As an example, the ports in the TCP protocol are well represented using a range-based selector (e.g., 1024-65535). We include in the range-based selectors all the category of selectors that have been defined by Al-Shaer and Hamed [12] as *prefix match*. These selectors allow the specification of ranges of values by means of simple regular expressions. The typical case is the IP address selector (e.g., 10.10.1.*). There is no need to distinguish between prefix match and range-based selectors as 10.10.1.* easily maps to [10.10.1.0, 10.10.1.255].

Another category of selector types includes the *regex-based selectors*, where the matching is performed by using regular expressions. This selector type is frequent at the application layer, where data are often represented as strings of text. The regex-based selector type also includes as subcase the *string-based* selectors, where matching is evaluated using string matching algorithms (SMA). Indeed, for our purposes, string matching can be mapped to regular expressions, even

if in practice SMA are much faster. For instance, Squid, a popular Web caching proxy that offers various access control capabilities, allows the definition of conditions on URLs that can be evaluated with SMA (e.g., `dstdomain`) or regex matching (e.g., `dstdom_regex`).

Finally, we introduce the idea of *custom check selectors* to model conditions that do not provide details about logic of the matching process. For instance, custom check selectors describe features of malware analysis and IDS tools that look for specific patterns and return a Boolean value if the pattern is found. In order to be properly used by high-level policy based processes (like reasoning systems, refinement systems), custom check selector need (at least) to be described as black-boxes, that is, by means of the list of fields that they take in input in order to return the Boolean verdict.

B. Policy-Level Capabilities

Besides the rules, other "technicalities" need to be specified in order to properly manage and configure security functions, and a sound capability model must also be able to describe them. According to the geometric model, a policy is specified by means of: 1) resolution strategy ($\mathcal{R}$), which abstracts the decision criteria for the action to apply when a packet matches two or more rules; 2) external data, such as priority, identity of the rule creator, and creation time, which are associated to each rule (but not part of the rule itself) and used by the resolution strategy to decide; 3) default action ($\mathcal{D}$), which is the action applied if no rules match the packet/flow.

We have ignored in this work options to generate configurations that may have better performance (e.g., with chains or ad hoc structures [10]). Adding support to these forms of optimization is certainly feasible with a limited effort but it was outside the scope of this paper, that is, to show that adding security awareness to NFV management and orchestration features is possible. It is one of the task for future work.

C. Formal Model and Algebra of Capabilities

Formally, the capabilities associated to a NSF X are defined by a 4-tuple: $cap_X = (A; C; R; D) \subseteq (\mathcal{A}; \mathcal{C}; \mathcal{R}; \mathcal{A})$. Where $\mathcal{A}$ is the set of all the possible actions and $A \subseteq \mathcal{A}$ are the actions available from X, $\mathcal{C}$ is set of all the existing conditions types and $C \subseteq \mathcal{C}$ are the ones available from X, $\mathcal{R}$ is the set of all the existing resolutions strategies and $R \subseteq \mathcal{R}$ are the ones contained in X, and $D \subseteq A$ lists the actions supported by X that can be used as default action ($\emptyset$ if the default action is not configurable).

Moreover, capabilities can be added and subtracted, given $cap_1 = (A_1, C_1, R_1, d_1)$ and $cap_2 = (A_2, C_2, R_2, d_2)$:

$$cap_{1+2} = cap_1 + cap_2 = (A_1 \cup A_2, C_1 \cup C_2, R_1, d_1)$$
$$cap_{1-2} = cap_1 - cap_2 = (A_1 \setminus A_2, C_1 \setminus C_2, R_1, d_1)$$

Note that addition and subtraction do not alter the resolution strategy and the default action method of the first operand, as our main intent was to model addition of modules (that enable conditions and actions but do not change the core behavior of the NSF). The operations only merge the rule based capabilities and leaves untouched the policy capabilities.

As an example, we report how to define the capabilities of a generic packet filter that supports the FMR, supports explicit default actions and a another packet filter that also supports time-based conditions:

$$\begin{aligned} A_f &= \{\text{Allow}, \text{Deny}\} \\ C_{pf} &= \{\text{IPsrc}, \text{IPdst}, \text{Psrc}, \text{Pdst}, \text{protType}\} \\ C_{time} &= \{\text{timeperiod}, \text{days}, \text{datestart}, \text{datestop}\} \\ cap_{pf} &= (A_f; C_{pf}; \{\text{FMR}\}; A_f) \\ cap_{pf\_time} &= cap_{pf} + (\emptyset; C_{time}; ;) \end{aligned}$$

D. Use of the Capability Model

Our capability model has several applications. First of all, it can used to precisely asses security policy enforceability, as needed by our policy refinement approach. If the security function can enforce the actions required by the policy and can identify the packets/flows to which the security policy wants the action enforced, then the security control is *capable* of enforcing the policy, and a manager can use it. Note that, in some cases, a single function may not be able to enforce a policy (incapable), however, a set of security functions can be capable of enforcing that policy. In both cases, a simple capability matching algorithm is enough.

Then, the capability model entails information about the valid MLP policies for a given NSF. Information about actions and classification features is actually used to determine how to write valid rules, policy-based capabilities allow setting up the remaining details. Indeed, it is not possible to use specific conditions or actions when an NSF that does not own the proper capabilities.

Furthermore, we have used the capability model to describe generic categories of security functions and specific products in unambiguous way. We used the model to describe the categories of security controls we have analyzed.

We have created a set of *generic security functions* (g-NSFs), one for each category, and we have associated them to a set of core capabilities owned by (almost) all the controls in a category. The purpose is to remove ambiguity, as with our formalization the behavior of each g-NSF is well understood, and use security function with the same clearness as network components (i.e., a switch is a switch and experts know how to use it even if it may have some vendor-specific functions). In our idea, vendors and developers should describe their products starting from the g-NSFs. That is, generic functions become templates that can be customized by adding and removing capabilities by means of the capability algebra in Section III-C.

IV. Refinement

The most significant achievement in this paper is the definition of the policy refinement model.

For the definition of the refinement model needed for automatic enforcement of security policies and dynamic adaptation to network changes, we assume the following design principles: a) the refinement process translates high-level policies into configurations directly usable by the NSFs; b) the refinement must warn users against cases of non-enforceability of the input policy on the target network. Moreover, we added another requirement, the vendor

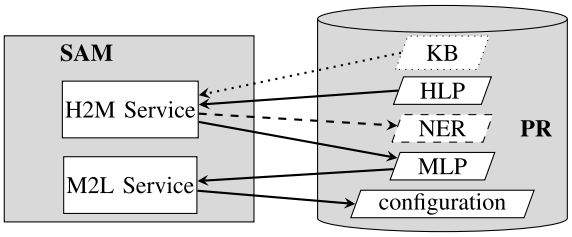

Fig. 2. The SAM inputs and outputs.

independence, that is, it should be easy to substitute an NSF with another one able to enforce the same types of policies.

Practically, the final goal of our approach is to generate concrete configurations for the NSFs into the target NFV network, which are the rightmost element in Figure 2. The syntax and expressiveness of the concrete configurations of the actual NSF is not under our control. We don't have the possibility to ask for new features nor to change the specific languages to cope with our needs. Hence, for our purposes, low level languages are constraints.

A. Approach and SAM Workflow

We propose to use two refinement steps and an intermediate abstract language to translate high-level policies into low level configurations (see Fig. 2). This is, the internal workflow performed of the SAM. The *High- to Medium-level Refinement Service* (H2M Service) first performs the semantic interpretation of the HLP policies to determine the capability needed to enforce them and infer the NSFs available in the network that can enforce them. Then, it outputs the set of all the MLP policies for all the NSFs in the NFV network. If the NSFs in the target NFV network do not satisfy these requirements, the user is provided with a detailed *non-enforceability report* (NER) and a proposed remediation strategy. The semantic interpretation of HLP policies, its relations to the capability model and the model-driven generation of valid MLP policies is presented in Section IV-D.

When the MLP policies are generated, the Medium- to Low-level Refinement Service (M2L Service) is in charge of translating the abstract policies in the concrete configurations for the selected NSFs, as presented in Section IV-E.

B. High-Level Policy Language

We defined the HLP language [14], [15] to express the security requirements collected from various use cases, such as parental control, corporate environments, user-driven paradigms, and IoT. The HLP has been designed as an authorization language that follows the subject-action-object-attributes paradigm (also referred to as target-effect-condition) [16]. HLP policies are composed of HLP statements in the form: `[sbj] action obj {(field_type,value),...}` where attributes are (`field_type,value`) pairs. HLP statements map to equivalent user friendly sentences, close to natural language, which are preferred for interactions with administrators. Examples of security requirement are "scan email for malware", "block access to gambling sites", "only allow Internet traffic from 18:30 to 20:00 for Alice".

It is worth noting that this language is an example to prove the validity of our approach in our use cases thus we don't claim that this is the universal language for enforcing security policies, although it can be extended to support more complex scenarios. Nevertheless, the scope of our language allows the specification of high-level policies for a number of security controls, such as packet filtering, application firewall, anti-malware, spam detector, VPN gateway, traffic anonymizer, IDS, DPI, file and traffic analyzer. Thus, as it stands, the HLP can be useful in daily security administrators' tasks as well as for normal users. The detailed list of security control categories that are supported can be found in the supplemental material together with examples of HLP statements.

C. Medium-Level Policy Language

The MLP language has been designed to abstract the configurations settings of NSFs. Even if in the practice these settings are conveyed with different languages, the capability model allows us to easily know, in an abstract way, all the features owned by each NSF. Therefore, the MLP language is specified with a general model that defines the abstract concepts (i.e., policies, rules, conditions, actions, resolution strategies) and the required associations among them (e.g., rules are aggregations of conditions and actions, a policy is association to its default action). However, when instantiating a policy for a specific NSF, only instances of concepts present in the capabilities of the NSF can be used. That is, a condition on URL cannot be used if the NSF does not own a URL-related classification feature.

D. H2M Service

The refinement process performed by the H2M Service can be split into two logical tasks: selecting the NSFs that will enforce the policy, and deriving configurations for them.

The decision of the NSFs is performed in two steps. First, the H2M Service interprets the HLP statements to determine the capabilities to enforce the policy. To have greater flexibility and to easily adapt to other contexts, this interpretation is not static, it is performed by using the information (i.e., the semantic mappings) in the Knowledge Base. Examples of association rules and inference rules are reported in the supplemental material. Specifically, the H2M Service performs the following steps. 1) It maps the values HLP *subject*, *object* and *fields* fields to low-level concepts by means of direct associations in the KB. For example, 'Employees' maps to a set of IP addresses, 'Madrid Subnet' to a subnet, 'lunch time' is a time period and 'gambling sites' is a set of IP and URLs. After this phase, each HLP statement becomes an *enriched* HLP statement. 2) It uses a set of inference rules, also stored in the KB, to associate concepts from each enriched HLP statement to the *rule-based capabilities* (i.e., *action* and *classification futures*) needed to enforce them. For example, for the HLP "*Employees are authorized to access Internet traffic (time period, {12:30-13:30 UTC})*", the H2M Service first deduces that an Allow filtering action is needed, because 'are authorized to' is associated to 'Internet traffic' which maps to 'any IP address'. Then it deduces that the target NSF must be able to enforce a condition on time (timestart).

Note that the system also deduces that filtering on IP destination is not needed as 'any IP address' is not needed if condition clauses are in DNF form. Moreover, it deduces that all the other communications not explicitly allowed by the HLP policy must be denied (i.e., the default action is Deny). The result is the set of the *required capabilities*. 3) It generates, with a process driven by the MLP policy model, a set of policy rules where the MLP concepts are instantiated from the low-level parameters in the enriched HLPs and the inferred rule-base capabilities.

After this semantic interpretation, the H2M Service identifies the NSFs in the target that can be used to enforce the HLP policy. This is performed by matching the required capabilities with the ones owned by the NSFs. However, according to (the action implied by) the HLP policy, there are different ways to select the NSFs and perform the matching, which we have categorized in single function enforcement, coupled functions enforcement, and path functions enforcement. As a result of the application of these three selection ways, all or a subset of the NSFs in each of the paths will be configured.

A policy requires *single function enforcement* when the activation of a single NSF in the target network is enough to enforce the policy. As an example, having a security control that performs anti-malware checks over the corporate email attachments is enough to enforce policies that ask for verification of email attachments even if network flows may need to be redirected to the selected NSF with SDN features. Therefore, checking the capabilities of individual NSFs is enough to determine capable functions.

A policy requires *coupled functions enforcement* when the policy enforcement requires the coordinate configuration of two security functions. As an example, configuring a VPN tunnel between two corporate subnets that communicate through the Internet (e.g., the Madrid and Turin ones) requires the use of two gateways. Moreover, the H2M Service also checks that the two coupled NSFs are also compatible. For instance, both IPsec- and TLS-based controls could be used to enforce a VPN policy, but both gateways must speak the same protocol. Also in these cases, network flows may need to be redirected to the selected NSFs (e.g., the gateways). Here network topology info is used to explore the neighbors of the entities involved by the policies and find the ones having the capabilities to be coupled (e.g., a VPN concentrator at the same security level as the connecting entity).

A policy requires *path functions enforcement* when the policy enforcement requires the simultaneous configuration of all the security functions *with specific capabilities* in the communication path. Communication paths are determined by processing topological information. Moreover, routing information is used to establish the actual communication paths from the alternative ones. As an example, the configuration of an allowed communication requires the configuration of all the filtering devices (e.g., packet filter or HTTP filter) in the path from the source to the destination. In these cases, the decision is not on the security control to use, rather on the path to choose, therefore, all the security controls in the path must own the required capabilities. Also in these cases, network flows may need to be redirected to the selected NSFs (e.g., the gateways).

If some required capability is not available or it cannot be properly coupled, the H2M Service generates the NER. It is not interesting discussing the format of the NER. Briefly, it contains the reason for non enforceability, that is, the policies that cannot be enforced, the missing capabilities in the target NSFs, or the impossibility to find one or more devices to couple to enforce the policy (like VPN gateways) in the set of NSFs that can be selected for a given user, together with a list of NSFs that can be added in the network (and where) to make the policy enforceable.

In case of availability of several NSFs that can enforce the HLP statements, the distribution of the generated policy rules to the available NSFs can be optimized. We have developed an optimization model that performs this distribution also based on network and device performance and security metrics (see Section V and VI).

Finally, the H2M Service generates the MLP policy for each one of the NSFs selected by the optimization process by 1) collecting the associated policy rules, and 2) finalizing the policy gathering from the KB the policy-based capability information, collecting each policy rules with the same *rule base capability*, specifying the resolution strategies and the default action (retrieving these information in the KB).

The derivation of the configurations in MLP is a simpler task. Once the security controls have been identified, generating the configuration is quite straightforward, as the required rules are determined when processing the HLP statements.

E. M2L Service

The M2L Service performs the syntax change to adapt the policy specified in MLP into actual NSFs configurations directly usable by the target NSFs. As an example, it is in charge to translate the MLP policy for a generic packet filter into a valid iptables configuration file. As it bridges the gap to real NSFs, it is a crucial component for the adoption of our refinement approach toward the automatic enforcement of security policies and the dynamic adaptation to network changes. It was also needed to validate and test our approach in OpenMANO. Nonetheless, its design did not require to address peculiar research issues.

We have implemented an M2L service as a framework that dynamically loads (from a remote repository) the plug-ins that translate the MLP into the NSF-specific syntax. Indeed, each NSF typically has a different configuration language, thus, an NSF-specific translation module proved to be the most convenient way to perform the mapping. Since the MLP policy already includes all the concepts that will be needed at the lowest level, every module has to perform a change of syntax and add the necessary fixed information (e.g., initialization, global options). Note that, supporting future NSFs that provide new capabilities does not pose issues to the MLP. Indeed, changes to the MLP are model-driven from the capability descriptions, that is, it is just required to extend the KB to support the new policies that can be enforced by these NSFs, as described in Section II-C.

V. Optimization

As anticipated, the optimization phase performed by the H2M Service selects the NSFs to use to enforce the HLP

policy, when alternatives security functions can be used. The idea is that an optimization phase can help in improving the performance and achieving a better use of the available resources, especially for large networks where multiple choices are often available. Instead of developing ad hoc allocation strategies, we have preferred tackling the decision problem formally, with an optimization model (in standard form). The optimization model is an instance of the set covering problem, which is NP-Complete in worst case but it is solved with good performance by standard solvers (there is also a greedy algorithm that works well in practice). Moreover, our formulation of the problem can be can be easily customized to support a large set of optimization scenarios by simply adding constraints and customizing the parameters, as shown below.

Let $\mathbb{P} = \{p_1, \ldots, p_t\}$ be the MLP policy rules derived from the refinement of the input HLP policies to enforce, and let $\mathbb{S} = \{s_1, \ldots, s_n\}$ be the NSFs in the target network, we define two functions to map them to capabilities, namely $\gamma_p : \mathbb{P} \rightarrow \mathbb{C}$ returns the capabilities needed to enforce a policy, and $\gamma_s : \mathbb{S} \rightarrow \mathbb{C}$ returns the capabilities owned by a NSFs.

A set of Boolean variables $x_{i,j} = \{0, 1\}$ reports if the NSFs $s_i \in \mathbb{S}$ is selected to enforce the policy rule $p_j \in \mathbb{P}$.

A set of metrics has been proposed to generalize the target function used to choose of the NSFs. That is, we assume that there are l metrics $\mu_1, \ldots, \mu_l$ that associate each NSF s to a real value r depending on the policy rule p it has to enforce:

$$\mu_i : \mathbb{S} \times \mathbb{P} \rightarrow \mathbb{R} \quad (s, p) \mapsto r$$

Note that this definition also covers, as subcase, metrics that do not depend on the policy to enforce.

A generic function K, named *cost*, computes the aggregated cost associated to the use of the NSF s to enforce the policy rule p, which is expressed as a real number and defined as a linear combination of different metrics values μ_i computed over a NSF s:

$$K : \mathbb{S} \times \mathbb{P} \rightarrow \mathbb{R} \quad (s, p) \mapsto \sum_{i=1}^{m} w_i \mu_i(s)$$

where w_i weights the importance of the metric μ_i.

For our prototype we have defined several sample single objective target functions by customizing the cost functions and defining the corresponding metrics. We have introduced metrics to estimate the (economic) costs associated to buying or using a NSF, metrics that capture user rating of NSFs, and experts trustworthiness expectations (i.e., by trivially counting the number of patches and averaging them with their severity), and security evaluation of NSFs.

Moreover, we have considered metrics to quantify performance, based on the information available in the VNF descriptors (see the supplemental material provided with this paper for examples of VNF descriptors). For instance, the *delay metric* estimates the time needed by a NSF to process traffic and by links to transfer it. The difficult part was modeling the performance profiles associated to NSFs. Indeed, while for virtual resources the performance degrades (linearly) with the number of rules, physical NSFs may use ad hoc hardware to have constant time processing regardless of the rules [10]. Unfortunately, models that estimate performance degradation based on the configured policies are not linear and needed a linearization to be solved efficiently. Furthermore, we have introduced metrics to estimate the "distance" between solutions, which weight the differences among where policy rules are enforced. These metrics are used to add penalties to solutions that are too different from an initial configuration and have been used to support change management (see Section VI). As a future work, we planned to introduce new metrics to capture CTO estimation. Finally, multi-objective criteria have been defined to trade-off competing objectives by combining single objective target functions.

Our model also includes several constraints, some of them are configurable by the users. The first set of constraints avoids that the NSFs allocated on each physical node exceeds the node resources. The only resources that we have considered in our optimization models are the virtual CPU count, needed RAM, and required disk storage, that is, the only data available in the VNF descriptors (see the supplemental material for examples of OpenMANO VNF descriptors). For instance, there are constraints to cope with the different NSF selection strategies. When enforcing a single function enforcement policy p_j, the model excludes from the decision space all the security functions s_i that do not own the required capabilities: $x_{i,j} = 0 \Leftrightarrow \gamma_p(p_j) \supset \gamma_c(s_i)$. Capable functions are OR-ed with additional equations that limit the number of functions to use (i.e., at most one). For instance, defence-in-depth is achieved by playing with these equations. Then, when looking for NSF to associate for coupled functions enforcement, only the neighbors of the end nodes implied by the policies are considered (e.g., VPN gateways are selected at the border of the network to protect). Thus, the model excludes all the NSFs that are not in the neighborhood and do not own the required capabilities: $x_{i,j} = 0 \Leftrightarrow s_i \notin N(e_1(p_j)) \cup N(e_2(p_j))$. where $e_1(p_j)$ and $e_2(p_j)$ are the functions that determine the endpoints (determined by the refinement process performed by the H2M Service) and $N(\cdot)$ is the function that abstracts the selection of the neighborhood of endpoints to consider (e.g., in the same network or broadcasting domain, or the part of the network at the same security level).

Supporting path functions enforcement also adds constraints. The NSFs in the same path to activate are AND-ed. Given the paths $\{\pi_k(e_1(p_j), e_2(p_j))\}_k$ between the endpoints implied by a policy, we introduced a set of variables $v_k = \bigwedge x_{i,j}$ (for $s_i \in \pi_k$) to force the configuration of all the nodes in the path π_k. Also in this case, equations allow the selection of at least one path ($\bigvee v_k = 1$), exactly h path ($\sum v_k = h$), or all the paths ($\forall k, v_k = 1$). In case of static routing, there is only one path to consider, the other possible NSF are therefore excluded. The optimization model is instantiated by the H2M Service that automatically generates optimization programs that instantiate these logical formulas with values from the target network and for the input policies. Then, the H2M service solves the generated models with standard solvers. We used the MOEA framework, as we planned support for multi-objective optimization, however, any ILP solver can be used.

VI. Management of Changes

We have considered several scenarios to deal with changes. We have addressed dynamic adaptation to network changes,

a first step toward security policy aware network management and orchestration. This is still an ongoing research, however, the preliminary results are promising.

First, with our approach, substituting a NSF X with a new one Y owning the same capabilities as X can be managed with a quick procedure. Indeed, after the refinement, the H2M Service outputs an MLP policy for a generic NSF that exploits (some or all) the capabilities of X. The MLP policy M is then translated to be usable by X by the M2L Service. Hence, switching to Y just requires that the M2L Service translates the policy M for the syntax of Y. More generally, the quick procedure can be used with any NSF that owns at least the capabilities of X exploited in M (vendor independence).

The general case of dynamic adaptation to network changes, that is, changes that happen in the network and are not just substituting a NSF with an equivalent one, can be dealt with a complete refinement of the HLP policy onto the new network. However, this approach may originate too many changes and lead the network to a temporary inconsistent state, as a subset of NSFs may enforce the new policy while the others NSFs still behave as required by the previous policy. Therefore, to reduce the issues related to changes, we have introduced the distance metrics, anticipated in Section V, which favor solutions that are maybe a bit less optimal from the performance point of view, but minimize the impact on the network. A simple metrics counts the NSFs whose policy needs to be updated, others count the number of coupled functions that need to be reconfigured to re-establish secure channels and the paths that are affected by the changes. Linear combinations of distance metrics are then used in the target functions. However, these distance metrics and the target functions built on them will need further research effort to guarantee a better stability of the network over time, not only on the single change.

Also the management of dynamic changes of the policy can be dealt with a complete refinement of the new HLP policy, as for dynamic adaptation to network changes. However, we are investigating faster refinement algorithms that only process the differences between the old and the new policy (delta refinement), which could reduce the likelihood of negative impacts of the refinement performance. Indeed, the management of changes is essential for the practical adoption of our approach. Unfortunately, the consequences of only considering the differences have not yet been extensively investigated. Indeed, inconsistencies may arise when mixing configuration rules from different refinement processes. Currently, we are researching criteria that characterize the differences that do generate inconsistencies to determine when the complete refinement can be avoided. Indeed, an a posteriori inconsistency checking would render useless the performance improvement of the delta refinement.

Another ongoing research concerns algorithms to automatically assess the changes to the input policy and the target network and decide the best way to push the new configurations. The idea is to develop advanced configuration deployment strategies that decide how and when to push configuration of NSFs to allow seamless transition between solutions, i.e., network states (i.e., topology and NSFs configurations). These strategies should avoid the inconsistencies (and related consequences) that may arise when passing from a network state to a different one as a consequence of changes.

VII. Implementation and Validation

We have integrated our solution into the OpenMANO project and we have used OpenMANO to validate our approach on virtualized scenarios. The virtualized scenarios have been selected in the context of EC funded project SECURED. The SECURED use cases were perfect to show that it is possible to perform a policy-based management and orchestration of virtualized networks. Indeed, these use cases depict NSPs enforcing security policies requested by residential users (divided in classes of services) and/or SMEs by means of one or more chains of network security services formed by one or more NSFs. Use cases were representative of medium size networks but did not allow us to measure the scalability of our approach, which was one of the critical points for its adoption. Therefore, we have also designed a set of tests where the SAM was used as a black-box policy refinement engine to derive configurations from HLP for network that were only represented with their service graphs, i.e., without actual deployment. Hence, our validation is composed of two parts: validation of the policy-based configuration process and testing on synthetic networks with very large number of policies.

A. Implementation

The OpenMANO project has developed three components: 1) *Openmano*, the northbound interface, based on a REST APIs, which exposes a set of orchestration and management tasks (such as creation and deletion of VNF instances and network services); 2) *Openvim*, the NFV Virtualized Infrastructure Manager that directly interfaces with an OpenFlow controller and a compute nodes via a REST API; 3) *Openmano-gui*, the web interface to invoke the Openmano API.

To prove the validity of our approach we have extended Openmano and Openmano-gui. The SAM has been implemented as a new Openmano module. The SAM refinement services are exposed by means of a REST API, which is used to pass as input the HLP and receive in output the MLP, the NSD and the NSF configuration to store in the PR. The SAM has been developed as a single threaded Java 1.7 application that relies on two Java-based open source frameworks: Drools and MOEA. Drools is a Rule Engine that we have used to implement an Expert System that reasons about capabilities for HLP policy enforcement purposes. Drools uses the Rete-OO algorithm whose performance is excellent in practice, also considering the expressiveness of the inference rules, and whose complexity is presented in depth in a past work [17].

The MOEA framework is a Java library for developing multi-objective optimization programs and solve them with evolutionary algorithms. We have used to implement the optimization models (Section V).

We also implemented the PR (using the Django REST framework) and added it to the OpenMANO repositories. The PR API allows the addition, deletion, and update of all the artifacts needed by our approach that were not available in the OpenMANO distribution. These artifacts include information about HLP and MLP policies, and NSF configurations.

We extended the openmano-gui to allow users to specify the input HLP policies and visualize the all the artifacts used by our refinement approach.

Finally, to perform the experiments, we have also implemented one Element Manager for each of the NSFs we have used for validation purposes in order to push configurations and manage their lifecycle (start, stop, restart).

B. Validation With Use Cases

The validation on use cases aimed at proving that our approach is actually able to select and configure the NSFs in a policy-driven manner. It consisted in the following phases: 1) selecting a use case network (with all the information about the available NSFs, including the capabilities) and specifying security requirements for that network; 2) drawing the network in OpenMANO with the openmano-gui; 3) specifying the corresponding HLP policy with the extension to the openamano-gui; 4) deploying and configuring the network with the SAM-extended OpenMANO and measuring the performance of every phase; 5) manually inspecting the generated NSFs policies at the MLP level (i.e., not the low-level configurations); 6) testing with probe packets and ad hoc built policy violations to verify, also with traffic inspection, that the selected security requirements were actually enforced.

We only report here the entire validation results of the network service example presented in Section II-D. However, we have considered several use cases from the SECURED project that include parental control, corporate network protection, and user-centric policy enforcement.[9]

The input policies in HLP of the example were:

$\mathcal{H}_1$: `enable email scanning {(mimeType, pdf)}`

$\mathcal{H}_2$: `Developers are authorized to access Internet {(time period, 13-15 UTC), (specific URL, www.facebook.com)}`

$\mathcal{H}_3$: `protects integrity and confidentiality {(source, TurinNet),(target, MadridNet)}`

The SAM correctly inferred the capability required to enforce the HLP policies then determined that H_1 can be enforced by Bro, H_2 can be enforced by Squid, and H_3 can be enforced by strongSwan. The SAM first generated the MLP policies for three generic anti-malware, application layer filtering, and IPsec channel protection security functions (that own a subset of the capabilities owned by the three selected NSFs). Then, the SAM translated the MLP policies into the NSF configurations.

We have performed further advanced testings to prove an effective reaction to changes. First we have modified the attributes in the input policies (time period from 13-15 to 8-12, specific URL from www.facebook.com to www.tweeter.com, and from specific URL=www.facebook.com to specific IP=10.0.0.1) and verified that the SAM only updated the squid MLP and low level configuration and did it correctly. Finally, we have changed squid with privoxy, another application layer filter that owns the capabilities to enforce $\mathcal{H}_2$. As expected, the SAM only generated the configuration of privoxy from the same MLP policy used before for squid.

[9]Use cases are sketched in the deliverable D6.3, further details can be found on https://github.com/SECURED-FP7.

C. Scalability Testing

We have generated synthetic networks by means of an algorithm developed to connected chains of network services composed of one or more NSFs to form a tree (whose root was intended as the connection to the "outside") then added connections among other tree nodes to simulate a proper number of redundant paths. Chains of network services were available in a catalogue and explicitly designed by us (i.e., chains are not randomly generated). A number of users, i.e., the target of the policies, were connected at the beginning of each chain. Other creation rules ensured that the generated networks were close to reality, for instance, increasing the concentration of security services at the root or selected intermediate nodes to simulate border security, providing some security services with a centralized or distributed approach, and ensuring that enough capabilities were available at each path to reduce non-enforceability issues. Finally, HLP policies were randomly generated for the inserted users starting from a set of template policies that covered the whole HLP expressiveness.

We have performed our tests on an Intel i7-3630QM@ 2.4 GHz laptop with 16 GB RAM. Each test was run on each scenario 50 times; results have been averaged.

We have first measured the time spent to perform the refinement of the HLP policy into configurations, which is the sum of the time to translate HLP policies into MLP policies for all the NSFs in the network (by the H2M Service) and the time to translate all the MLP policies into configurations (by the M2L Service). We have generated networks with a fixed number of NSFs, while we have increased the number of HLP policies (ranging from 10 to 500.000.

Figure 3a depicts the results obtained respectively with 5 and 20 NSFs. As expected, there is a more than linear dependency with the HLP policy count. We noted that for a small number of HLP policies, the initialization time of the two frameworks (Drools and MOEA) gave an important contribution that has a minor impact when more than 500 rules are considered. This test gave us the most important result, as it proved that the approach scales well enough to be used in practice, despite the limited hardware resources used. We also measured that the time to generate configurations from MLP linearly depends on the number of rules in the MLP policy but is negligible. In the work case, the translation of five thousands rules (for Squid) took less than 100 ms. Furthermore, the MLP translation can be easily parallelized, an advantage when the number of NSF to configure is huge. Moreover, we have measured the time required by the Element Manager to configure the NSFs. Figure 3a plots the results obtained when measuring the time to push a configuration depending on the number of rules it contains (ranging from 10 to 1000 rules). The bottom line represents the results of Iptables, the NSF that has shown the best performance, and the top line Squid, the NSF that required more time to be configured. The configuration time is nearly constant and lasts a few tenths of second. Thus it can be considered negligible for scalability analysis purposes.

Finally, even if it is not directly related to our contributions, it interesting reporting here the time Openvim spent to instantiate a network service chain from its NSD, depending on the number of NSFs. Figure 3c depicts the results obtained to

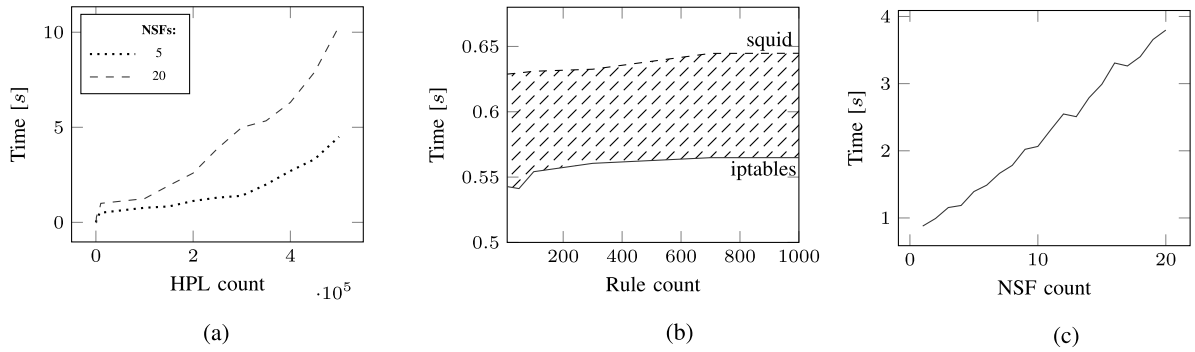

Fig. 3. Results of the scalability testing. (a) Time to perform the HLP refinement. (b) Time to configure the NSFs. (c) Time to instantiate a service chain.

instantiate up to 20 NSFs. Also in this case, the time to setup the network grows almost linearly, as expected since the virtual machines are deployed in order by OpenMANO. This result confirmed the feasibility of our approach. Given the current NFV technology, the optimized policy would be ready before the MANO completes the instantiation of the virtualized network.

VIII. Related Work

A. NFV Management, Orchestration and Modeling

In addition to the ETSI working group, which has defined the standard, a great number of researchers analyzed in depth the NFV challenges and benefits. The most significant papers are authored by Mijumbi *et al.* who proposed several unexplored NFV research topics. Authors stated that management and orchestration of NFV-based networks will become more challenging in future, the most promising proposal being the MANO framework by ETSI [8]. Mijumbi *et al.* also highlighted the importance of formally modeling network and security functions to exploit the NFV's ability to deliver high levels of automation and flexibility. Since the resources and functions in NFV will be provided by different entities, the availability of well understood, open and standardized descriptors for multi-vendor resources, functions, and services will be key to large-scale NFV deployment.

ETSI and others working groups have provided a possible set of information and data models for NFV resource and function modeling. Examples are OVF, TOSCA, YANG and SID. However these models are only used in the definition of software/hardware resource (their components, relationships, and the processes that manage them), but, at the best of our knowledge, none seems to address issues related to NSFs management and support. Giotis *et al.* [18] proposed a preliminary architectural model for policy-based VNF orchestration, which use an Information Model to abstracts network resources and VNFs capabilities. The main limitation is that the model only addresses access control and forwarding policies. Shen *et al.* proposed vConductor, a MANO architecture to support E2E Virtual Network Integration as a Service (VNIaaS) [19]. vConductor is a NFV-based service that provides a multi-objective resource scheduling and NFV-oriented inventory management capabilities. However, it does not support policy refinement, the configurations of NSFs are directly written by network administrators. Finally, Spinoso *et al.* proposed the use of functional descriptions of VNFs to correctly integrate and configure third-parties VNFs in NFV NSP networks [5].

B. Optimization

The provisioning of NFV brings up the resource allocation problem, also known as Virtual Network Function Placement (VNF-P). Moens and De Turck [20] proposed an Integer Linear Programming (ILP) model to minimize the number of used servers in mixed physical and virtualized environments. Yoshida *et al.* [21] proposed to optimize the resource allocation when enforcing stakeholder policies on a network infrastructure. Conflicting objectives are dealt with a Multi-objective Genetic Algorithm (MOGA), which produces approximate solutions in a reasonable computation time. Gember *et al.* [22] present a network-aware orchestration layer for Middleboxes (MBs), named Stratos. Stratos allows tenants to specify middlebox deployment by using a simple logical topology abstraction and three features, application-aware scaling, rack-aware placement, and network-aware flow distribution. Mehraghdam *et al.* [23] proposed an optimization model for building chains of VNFs that satisfy requirements of the tenants and the operator expressed with a formal language. These work are complementary to our optimization model. Their optimization criteria could be used, after extension and adaptation, to improve our optimization model.

C. Refinement

Policy refinement has received great attention in literature.

Many models and methods have been proposed to efficiently deal with both management and enforcement of security policies. However, the interest from the theoretical viewpoint has not yet shown real advancements in the practice. The main reasons are related to the lack (and in most cases of the unavailability) of formal representations of the huge amount of data that are needed to actually perform the refinement.

NFV networks, and in general, virtual environments, where orchestration and management have access to a detailed representation of the running infrastructure, are a promising field to finally see the refinement dream to come true. According to Weise and Martin [24], a security policy must be *implementable* through system administration procedures (e.g., publishing of acceptable use guidelines) and *enforceable* with security tools or controls, where appropriate, and with

sanctions, where actual prevention is not technically feasible. Unfortunately, in literature, enforceability analysis has received little or no attention and it has not been investigated in-depth. For instance, in a real scenario, some policies may be less precisely enforceable in some systems than in others or in worst case, completely non-enforceable. As suggested by [25], the access control on traditional UNIX systems is much less granular when compared with ACLs on modern implementations and some access control requirements may be not fully supported. Schneider proposed an approach to determine when a security policy is enforceable with Execution Monitoring [26]. A policy is non-enforceable when its set of executions is not a safety property and the Execution Monitoring does not have an enforcement mechanism. This concept has been improved by Bauer *et al.* who extended the model the Schneider's to support new classes of security policies [27], and Basin *et al.* [28] who proposed Execution Monitor able to distinguish actions that are controllable and those that are only observable. We tried to overcome current limitations by proposing the capability model, which allows precise identification of non-enforceability issues.

Refinement models exist for firewalls. Bartal *et al.* [29] proposed a solution, named Firmato, for the refinement of high-level filtering requirements into packet filter configurations. Firmato uses a knowledge base, formalized with an entity-relationship model that describes high-level filtering requirements and network topologies, and a translator, which refines high-level filtering requirements into packet filter rules. However, Firmato has been validated on a network with a single border firewall, hence its applicability to large and heterogeneous scenarios has not been proven. Verma *et al.* used a similar approach to develop FACE, a firewall analysis and configuration engine [30]. FACE takes as inputs the network topology and a global security policy written in a high-level language, and outputs the packet filter configurations.

Valenza *et al.* proposed the use of ontologies to capture the semantics of high-level filtering and channel protection policies (e.g., IPsec) [31]. By means of ontology inferences, high-level concepts (such as users and services) are mapped to low-level network concepts (such as IP addresses, ports, protocols) so that high-level security policies can be translated into configurations settings for target security controls. Our approach shares with this work the idea of an inference engine to perform HLP policy refinement. However, our approach is much more powerful, as ontology proved to be very difficult to use and extend, failed to capture concepts (like connections) in a simple way, and their performance scaled quite slowly.

IX. Conclusions

This paper presented the first steps toward a Security Policy Aware NFV MANO (SPA-MANO) by supporting two innovative scenarios: automatic enforcement of security policies and dynamic adaptation to network changes in NFV networks. We have developed refinement models that allow the transformation of high-level security requirements into configuration settings for the Network Security Function (NSF) in the target NFV network. These models also support optimal selection processes, which permit the selection of the NSFs to use in case of alternative functions. To build such an approach, we have defined an algebra of capabilities useful to describe the security functions implied by the security policy requirements and the functions actually available and the NSFs. We have also proved that supporting security awareness in NFV networks is feasible and requires limited changes in the current NFV architecture. Moreover, we have integrated the new component in the OpenMANO framework.

It is important to highlight that this approach cannot be used as it stands now in production networks. Indeed, policy languages need to be extended to support more security requirements and to be adapted to the various scenarios where this new paradigm needs to be applied. Also the capability model needs to be further detailed to be able to express a larger set of NSFs. However, both for policies and capabilities, an incremental effort is needed, whose purpose is support of a broader range of cases. Indeed, we have analyzed large but not exhaustive set of security controls, most of them are traditional security controls, which can be virtualized, but we have not addressed specific NSFs that are natively virtual. Thus, limited changes to the capability model can be expected.

Presently, scalability of this approach has been proved very promising also on off-the-shelf hardware. Advantages are expected when using more powerful resources, like the ones available at the data centers where NFV networks will be deployed. However, improvements to the refinement algorithms can further reduce the impact of the policy refinement.

However, the most important future works concentrate on the definition of the security aware resource allocation, that is, on the design of the missing parts to have a SPA-MANO that also performs optimized allocation based on policy information. This will add more degrees of freedom to the decision process, thus rendering the optimization problem more complex. On the other hand, this optimization can lead to a better use of available resources and a save of the expenditures at the NSP, especially if integrated with target functions that better support changes. Also the management of dynamic changes in the high-level policy is an interesting research topic. Indeed, only refining the differences between the old and new policy may consistently optimize the refinement process, as the whole security policy will not change frequently. Therefore, we are investigating criteria that ensure that a partial refinement does not introduce any inconsistency in the globally enforced policy. As future application case, we want to extend also Open Source MANO, which has been selected as the new reference implementation of NFV by ETSI. Since, Open Source MANO is built on OpenMANO, we expect limited effort.

References

[1] R. Mijumbi *et al.*, "Network function virtualization: State-of-the-art and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 236–262, 1st Quart., 2016.

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

[3] R. Mijumbi, J. Serrat, J.-L. Gorricho, S. Latré, M. Charalambides, and D. Lopez, "Management and orchestration challenges in network functions virtualization," *IEEE Commun. Mag.*, vol. 54, no. 1, pp. 98–105, Jan. 2016.

[4] J. D. Moffett and M. S. Sloman, "Policy hierarchies for distributed systems management," *IEEE J. Sel. Areas Commun.*, vol. 11, no. 9, pp. 1404–1414, Dec. 1993.

[5] S. Spinoso, M. Leogrande, F. Risso, S. Singh, and R. Sisto, "Seamless configuration of virtual network functions in data center provider networks," *J. Netw. Syst. Manage.*, vol. 26, no. 1, pp. 222–249, Jan. 2017.
[6] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying network-wide invariants in real time," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 467–472, 2012.
[7] R. Mijumbi *et al.*, "Topology-aware prediction of virtual network function resource requirements," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 1, pp. 106–120, Mar. 2017.
[8] *Network Functions Virtualisation (NFV): Management and Orchestration (GS/NFV-MAN-001)*, ETSI, Sophia Antipolis, France, 2014.
[9] *Network Functions Virtualisation (NFV): Architectural Framework (RGS/NFV-002)*, ETSI, Sophia Antipolis, France, 2014.
[10] D. E. Taylor and J. S. Turner, "Scalable packet classification using distributed crossproducing of field labels," Dept. Comput. Sci. Eng., Washington Univ., St. Louis, MO, USA, DC, Tech. Rep. WUCSE-2004-38, 2004.
[11] C. Basile and A. Lioy, "Analysis of application-layer filtering policies with application to HTTP," *IEEE/ACM Trans. Netw.*, vol. 23, no. 1, pp. 28–41, Feb. 2015.
[12] E. S. Al-Shaer and H. H. Hamed, "Modeling and management of firewall policies," *IEEE Trans. Netw. Service Manage.*, vol. 1, no. 1, Apr. 2004.
[13] C. Basile, A. Cappadonia, and A. Lioy, "Network-level access control policy analysis and transformation," *IEEE/ACM Trans. Netw.*, vol. 20, no. 4, pp. 985–998, Aug. 2012.
[14] C. Basile, A. Lioy, C. Pitscheider, F. Valenza, and M. Vallini, "A novel approach for integrating security policy enforcement with dynamic network virtualization," in *Proc. 1st IEEE Conf. Netw. Softw. (NetSoft)*, Apr. 2015, pp. 1–5.
[15] F. Valenza *et al.*, "A formal approach for network security policy validation," *J. Wireless Mobile Netw., Ubiquitous Comput. Dependable Appl.*, vol. 8, no. 1, pp. 79–100, Mar. 2017.
[16] S. Godik *et al.*, *eXtensible Access Control Markup Language (XACML) Version 3.0*, Standard 1, Jan. 2013. [Online]. Available: http://docs.oasis-open.org/xacml/access_control-xacml-2.0-core-spec-cd-01.pdf
[17] D. Sottara, P. Mello, and M. Proctor, "A configurable rete-OO engine for reasoning with different types of imperfect information," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 11, pp. 1535–1548, Nov. 2010.
[18] K. Giotis, Y. Kryftis, and V. Maglaris, "Policy-based orchestration of NFV services in software-defined networks," in *Proc. 1st IEEE Conf. Netw. Softw. (NetSoft)*, Apr. 2015, pp. 1–5.
[19] W. Shen, M. Yoshida, K. Minato, and W. Imajuku, "VConductor: An enabler for achieving virtual network integration as a service," *IEEE Commun. Mag.*, vol. 53, no. 2, pp. 116–124, Feb. 2015.
[20] H. Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *Proc. 10th Int. Conf. Netw. Service Manage. (CNSM) Workshop*, Nov. 2014, pp. 418–423.
[21] M. Yoshida, W. Shen, T. Kawabata, K. Minato, and W. Imajuku, "MORSA: A multi-objective resource scheduling algorithm for NFV infrastructure," in *Proc. 16th Asia-Pacific Netw. Oper. Manage. Symp.*, Sep. 2014, pp. 1–6.
[22] A. Gember *et al.*. (2013). "Stratos: A network-aware orchestration layer for virtual middleboxes in clouds." [Online]. Available: https://arxiv.org/abs/1305.0209
[23] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Proc. IEEE 3rd Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2014, pp. 7–13.
[24] J. Weise and C. R. Martin, "Developing a security policy," SANS Inst., Palo Alto, CA, USA, Tech. Rep. 1, Apr. 2003.
[25] M. Bishop and S. Peisert, "Your security policy is what?" Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. CSE-2006-20, 2006.
[26] F. Schneider, "Enforceable security policies," *ACM Trans. Info. Syst. Sec.*, vol. 3, no. 1, pp. 30–50, 2000.
[27] L. Bauer, J. Ligatti, and D. Walker, "More enforceable security policies," in *Proc. Workshop Found. Comput. Security (FCS)*, Jul. 2002, pp. 1–27.
[28] D. Basin, V. Jugé, F. Klaedtke, and E. Zălinescu, "Enforceable security policies revisited," *ACM Trans. Inf. Syst. Secur.*, vol. 16, no. 1, p. 3, Jun. 2013.
[29] 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.
[30] P. Verma and A. Prakash, "FACE: A firewall analysis and configuration engine," in *Proc. Symp. Appl. Internet*, Feb. 2005, pp. 74–81.
[31] F. Valenza, C. Basile, D. Canavese, and A. Lioy, "Classification and analysis of communication protection policy anomalies," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2601–2614, Oct. 2017.

Cataldo Basile received the M.Sc. degree *(summa cum laude)* and the Ph.D. degree in computer engineering from the Politecnico di Torino, in 2001 and 2005, respectively. He is currently a Research Assistant with the Politecnico di Torino. His research is concerned with the policy-based management of security in networked environments, policy refinement, general models for detection, resolution and reconciliation of specification conflicts, and software security.

Fulvio Valenza received the M.Sc. degree *(summa cum laude)* and the Ph.D. degree *(summa cum laude)* in computer engineering from the Politecnico di Torino, Italy, in 2013 and 2017, respectively. His research activity focuses on network security policies, access control policies, orchestration, and management and automatic configuration of network security functions in the context of SDN/NFV-based networks.

Antonio Lioy received the M.Sc. degree *(summa cum laude)* in electronic engineering and the Ph.D. degree in computer engineering from the Politecnico di Torino. He is currently a Full Professor with the Politecnico di Torino, where he leads the TORSEC Cybersecurity Research Group. His research interests include network security, policy-based system protection, trusted computing, and electronic identity.

Diego R. Lopez received the M.S. degree from the University of Granada in 1985, and the Ph.D. degree from the University of Seville in 2001. He joined Telefónica I+D in 2011 as a Senior Technology Expert on network infrastructures and services after several years in the academic sector. His current interests are related to network infrastructural services, new network architectures, and network programmability and virtualization.

Antonio Pastor Perales received the M.Sc. degree in industrial engineering from the Carlos III University of Madrid, Spain, in 1999. Since 1999, he has been with Telefónica I+D, where he is currently a Network Security Expert and is involved in the research and engineering of different worldwide Telefónica's networks.