Network Functions Virtualization Access Control as a Service

Manel Smine[1(✉)], David Espes[2], Nora Cuppens-Boulahia[1,3], and Frédéric Cuppens[1,3]

[1] IMT Atlantique, Rennes, France
manel.smine@imt-atlantique.fr, {nora.cuppens,frederic.cuppens}@polymtl.ca
[2] University of Western Brittany, Brest, France
david.espes@univ-brest.fr
[3] Polytechnique Montréal, Montreal, Canada

Abstract. NFV is an important innovation in networking. It has many advantages such as saving investment cost, optimizing resource consumption, improving operational efficiency and simplifying network service lifecycle management. NFV environments introduce new security challenges and issues since new types of threats and vulnerabilities are inevitably introduced (e.g. security policy and regular compliance failure, vulnerabilities in VNF softwares, malicious insiders, etc.). The impact of these threats can be mitigated by enforcing security policies over deployed network services. In this paper, we introduce an access control as a service model for NFV services. The proposed approach can deploy several kinds of access control model policies (e.g. RBAC, ORBAC, ABAC, etc.) for NFV services and can be easily scaled.

Keywords: Network Functions Virtualization (NFV) · Access control · Policy enforcement · Domain type enforcement (DTE)

1 Introduction

Network Functions Virtualization (NFV) is a network architecture concept which virtualises network functions (firewalling, DNS, intrusion detection, etc.). It creates a Virtualized Network Function (VNF) instance that is deployed over a Virtualized infrastructure. Usually, a Virtualized infrastructure is able to host many VNFs of different types. These VNFs can be chained to provide virtual network services. NFV promises a number of advantages to network operators such as reducing hardware costs, deployment in fast time and scalability. Despite advantages, security concerns are an important obstacle for a wide adoption of NFV. New threats and vulnerabilities are inevitably introduced such as security policy violation [12], VNF softwares vulnerable to different kinds of software flaws [18], and malicious insiders that can be a serious threat for user privacy and can lead to data confidentiality exposure [12]. Solutions to enhance the security of VNF network services are (1) to control the access to the different

Published by Springer Nature Switzerland AG 2020
A. Singhal and J. Vaidya (Eds.): DBSec 2020, LNCS 12122, pp. 100–117, 2020.
https://doi.org/10.1007/978-3-030-49669-2_6

components of the VNF network service and (2) to control what information is authorized to be transferred between the different components of the VNF network service.

In this paper we propose a formal model that provides a software-defined access control as a service capability for network services. First, it allows to specify high-level access control requirements to be enforced over network services. Second, it uses a provably correct method for transforming the high-level access control requirement towards a domain type enforcement (DTE) specification. Finally, our model defines an efficient enforcement method, as illustrated by the different conducted experimental evaluations in Sect. 5. Compared to existing models, our model is: (1) generic since it takes into consideration any type of access control policy such as RBAC [20], ORBAC [11], ABAC [9], etc., (2) compliant with the ETSI-NFV infrastructure in the sense that it does not require any modification of the latter for policy deployment, (3) and scalable thanks to our enforcement method that allows to add as many enforcement points as needed (e.g., for load balancing purposes) without impacting the functioning of the network services.

The paper is organized as follows. Section 2 reviews previous related research on existing security orchestrators and access control models for NFV infrastructure. Section 3 provides some background knowledge for understanding the proposed architecture. Section 4 defines our proposed model. Section 5 provides an overview of the implementation of our model and presents the evaluation results. Finally, Sect. 6 concludes the paper and outlines future work.

2 Related Work

Policy management and deployment in NFV architecture have recently been the topic of several researches. Many approaches have been proposed to define and enforce security policies over NFV architecture to ensure their security. Basile et al. propose in [4] an approach aiming to provide specific security properties over Virtualized networks. This approach relies on a new software component called Policy Manager to handle high-level security policies specified by the users and refine them into configurations for specific VNF. Unfortunately, this approach does not specify what kind of properties can be handled and how these required security property are refined to deployable configurations. In [14], Montero et al. propose a user-centric model named SECURED, allowing to express and deploy security policies to protect users' security in NFV. Due to its user-centric characteristic, the proposed model is completely oriented to protect users' security when interacting with NFV network services and cannot be used to secure NFV network services themselves. In FlowIdentity [24], a Virtualized network access control function using OpenFlow protocol is proposed. It is a solution for network access control in SDN architectures with policy enforcement over a stateful role-based firewall on OpenFlow switches. Unfortunately, the proposed approach is deeply dependent on the SDN architecture and thus cannot be directly used in an NFV architecture. To overcome the previous limitations,

Leopoldo, et al. propose ACLFLOW [13] that is a Network Functions Virtualization (NFV)/Software-Defined Networking (SDN) security framework allowing to translate regular ACL rules into OpenFlow filter rules. ACLFLOW optimizes the evaluation of ACL rules by prioritizing the most popular rules to accelerate switching operations. Unfortunately, ACLFLOW cannot be used to enforce more advanced access control policy models such as Role Based Access Control (RBAC), Organization-Based Access Control (ORBAC), Attribute-Based Access Control (ABAC), etc.

To solve multi-propagation problems of the concept of NFV such as verification and authorisation issues, Guija and Siddiqui [7] use the NFV-based SONATA Service Platform for authentication and authorisation mechanisms, specifically for Identity and Access control of micro-services in 5G platforms for services Virtualization, orchestration and management. This solution relies mainly on OAuth 2 [8] and OpenID Connect [19] to form the implementation of the user management module allowing Role Based Access Control and Identity management to follow the centralized authorization approach. However, This dependency on OAuth 2 and OpenID Connect makes this solution applicable only on services where a HTTP-based communication is used between their different components.

The model proposed in the standard ETSI-NFV [6] describes the NFV Security Monitoring and Management architecture. The proposed architecture introduces two components: The NFV Security Controller that orchestrates system wide security policies and acts as a trusted third party and the NFV Security Monitoring Analytics System which performs secure Telemetry acquisition from the NFV system and can derive threats and anomalies from the telemetry. However, only the model is defined and no specification of how all this work is done. In addition, several interfaces that the architecture defines are not specified (e.g. the connection between the controller and the Operating Support System/Business Support System (OSS/BSS).

In the literature, several security orchestrators have been defined to control the access in NFV infrastructure. In [10], Jaeger et al. propose an SDN based security orchestrator which improves the ETSI NFV reference architecture with an extensive management of trust and offer a global view for fast and efficient topology validation. Unfortunately, no concrete use case and implementation of the provided security requirements are given. The authors of [23] present an architecture for NFV environments focusing on the automation of access control management deployment. Unfortunately, the authors do not provide any information about how access control policies are deployed in a VNF network service. In [15], Montida et al. develop a security orchestrator as an extension of the MANO NFV orchestrator to manage the security properties of network services in their entire lifecycles. They extend the Topology and Orchestration Specification for Cloud Applications (TOSCA) model [5] with particular security attributes that are required to create access control policies and finally enforced in the cloud infrastructure. They instantiate the proposed security orchestrator in [16,17] through the implementation of an access control model which consists of deploying an access control policy over a network service. However, the

proposed approach suffers from several limitations. First, it requires the modification of the NFV infrastructure since specific agents has to be deployed on the NFV that compose the considered network service to enforce the access control policy. Second, the proposed access control model is not generic enough since it can only handle policies specified using the RBAC model. Finally, it is not clear how the high level access control policy is transformed into a concrete deployable policy.

Compared to existing access control models, our model offers several advantages. First, it is a generic model as it can handle most kind of access control policy models such as RBAC, ORBAC, ABAC, etc. Second, it offers formal and efficient methods for deploying access control policies at the concrete level. Third, the deployment method proposed in our model does not require any modification at the NFV infrastructure level.

3 Background

This section provides background material about all main technologies to enable the deployment of our security policy.

VNF Forwarding Graph: ETSI defined the notion of a VNF forwarding graph (VNFFG) [3] known also as Service Function Chaining (SFC). It is used to manage a traffic through a sequence of network functions (NF) that should be traversed in an order list of VNFs. VNFFG are described by VNF Forwarding Graph Descriptors (VNFFGD). Each forwarding graph is composed of a set of forwarding paths.

Network Service: A VNF service is composed of a set of VNFs that are represented using a deployment template VNF descriptor (VNFD) which define their properties and a set of forwarding graphs that are defined using a deployment template VNF FG Descriptor (VNFFGD).

Domain and Type enforcement (DTE): The technique Domain and Type Enforcement (DTE) protects the integrity of military computer systems. It was designed to be used in combination with other access control techniques. As with many access control schemes, DTE views a system as a collection of active entities (subjects) accessing a collection of passive entities (objects) based on rules defined in an attached security context and groups processes into domains, files into types and restricts access from domains to types as well as from domains to other domains.

4 The Proposed Model

4.1 Adversary Model

To understand the scope of the problem and assess the risks, we have to develop an adversary model. The adversary model considered in this paper is composed

of an attacker, a NFV infrastructure hosting the multiple VNFs that compose the network services to be deployed, and an access control engine that is used to deploy the access control policy. In our model, the objective is to allow users including the likely attacker to perform operations that a normal VNF infrastructure user can do. It means that the attacker can generate and modify a flow, attack to try to compromise a VNF. We suppose that the adversary will be able to interact with the VNF composing the deployed network service but he cannot control or modify the behavior of the access control engine as well as the VNF that will be used to enforce the access control policy. This later is supposed to be hardened, i.e., keeping the operating system up to date, minimising the installed packages to minimize vulnerabilities, enable and correctly configure a firewall, etc.

4.2 New Enforcement Model

In this section, a formal modelization of the security policy to be deployed is proposed. The proposed model defines what is an access query and how it can be evaluated. Since DTE has made its evidence for the enforcement of access control policies at operating system level, a method for transforming an access control policy towards a DTE specification is proposed. The proposed transformation allows us to benefit from the advantages of DTE. In particular, it allows entities having the same access requirements to be collected into domains and types which allows to find an appropriate balance between security and policy deployment complexity. We prove that the proposed transformation method is correct and we show how the DTE policy is enforced.

Security Policy Specification

Definition 1 (Security Policy). *A security policy SP is composed of a set of access control rules* $\{r_1, \cdots, r_i\}$. *Each rule* r_i *comprises:*

- *A subject* S_i *that represents one or many entities that want to access the object, these entities are characterized by a set of properties* $\mathcal{P}_i^S = \{p_1^s, \cdots, p_n^s\}$.
- *An action* A_i *that represents the operation that is going to be performed by* S_i *on* O_i, *each action is characterized by a set of properties* $\mathcal{P}_i^A = \{p_1^a, \cdots, p_l^a\}$.
- *An object* O_i *represents one or many resources over which the action* A_i *is going to be performed.* O_i *are characterized using two types of properties: (1) a set of properties* $\mathcal{P}_i^E$ *that characterises the entities (e.g., VNF) and (2) a set of properties* $\mathcal{P}_i^R$ *that characterises the resources inside those entities and over which the action* A_i *will be performed.*
- *A context* C_i *under which the rule can be invoked.*
- *A decision* D_i *indicating whether it is a permission or denial rule.*

In our model, each rule r_i *in the security policy will be represented as follows:*

$$r_i = \langle S_i, A_i, O_i, C_i, D_i \rangle$$

We note that the properties that characterize the entities representing the subject S_i (resp. the object O_i) may include: attributes of S_i (resp. O_i) in the considered system (e.g., the IP address of a network to which the subject belongs, etc.), functional attributes representing the provided functionalities (e.g., routing, deep packet inspection, firewalling, etc.), and security attributes that represent the security properties that is associated to S_i (resp. O_i) (e.g., the security level, trust level, etc.). These properties are to be retrieved from the VNF descriptors that compose the service to be deployed. For instance, the rule saying that any VNF providing web client functionalities and having a high security level can read the content of any web page on a VNF providing a web server functionality and having a high security level if the client is using *https*, can be specified using the following notation:

$$\langle S = \{func : web\_client, sec\_level : high\}, A = Action : read, proto = https, O = \{\mathcal{P}^E = \{func : web\_server, sec\_level : high\}, \mathcal{P}^R = file\_name : any\}, C = \{\text{between 8am and 8pm}\}, D = allow\rangle$$

Our specification of the security policy represented in Definition 1 can be used to represent many access control model policies such as RBAC and ABAC. First, RBAC is based on the notion of *subject*, *permission* that is represented by a relation between an *action* and an *object*, and a specific attribute representing a *role*. The first three notions (i.e., subject, object and action) can be straightforwardly translated to our model. The notion of *role* can be seen in our model as a specific property of the subject. Similarly, the attributes used in the ABAC model can be translated in our model to properties that characterize a subject, an object, a context, or an action.

Definition 2 (Access Query). *An access query $\mathcal{AQ}$ is represented by the quadruplet $\langle S^q, O^q, A^q, C^q\rangle$ where S^q represents the subject performing the query, O^q the object over which the query is performed, A^q the action performed by the query, and C^q the request context under which the query is performed. Given a security policy $\mathcal{SP}$, $\mathcal{AQ}$ is allowed by $\mathcal{SP}$ if and only if the following condition holds:*

(i) $\exists r_i \in \mathcal{SP}$ *such that* $S^q \in S_i, O^q \in O_i, A^q \in A_i$, C^q *satisfies* C_i, *and* $D_i = allow$.

$\mathcal{AQ}$ is denied by $\mathcal{SP}$ if and only if one of the following conditions hold:

(ii) $\nexists r_i \in \mathcal{SP}$ *such that* $S^q \in S_i, O^q \in O_i, A^q \in A_i$, C^q *satisfies* C_i, *and* $D_i = allow$.

(iii) $\exists r_i \in \mathcal{SP}$ *such that* $S^q \in S_i, O^q \in O_i, A^q \in A_i$, C^q *satisfies* C_i, *and* $D_i = deny$.

Policy Transformation. In this section, we propose a method for transforming an access control policy as defined in Definition 3 towards a DTE specification. Then, we prove that the transformation we propose is correct.

Definition 3. *Given a security policy $\mathcal{SP}$ composed of n rules $r_1, \cdots, r_n$. $\mathcal{SP}$ is transformed to a DTE policy by performing, for each rule $r_i \in \mathcal{SP}$ the following steps:*

- ***step 1:*** *If there exist no $j < i$ such that $\mathcal{P}_i^S = \mathcal{P}_j^S$, define the domain $s\_\mathcal{P}_i^S\_d$ which will contain all entities of the considered system (i.e., the network service to be deployed) that have the set of properties $\mathcal{P}_i^S$ used to characterize the subject S_i. Otherwise, use the same domain $s\_\mathcal{P}_j^S\_d$ (i.e., $s\_\mathcal{P}_i^S\_d = s\_\mathcal{P}_j^S\_d$) defined for the subject S_j of the rule r_j.*
- ***step 2:*** *If there exist no $k < i$ such that $C_i = C_k$, define new type c_i_t. Otherwise, use the same type c_k_t defined for the context of the rule r_k (i.e., $c_i\_t = c_k\_t$).*
- ***step 3:*** *If there exist no $l < i$ such that $\mathcal{P}_i^R = \mathcal{P}_l^R$ define new type $o\_\mathcal{P}_i^R\_t$ which will be associated to all resources of the considered system that have the set of properties $\mathcal{P}_i^R$. Otherwise, use the same type $o\_\mathcal{P}_l^R\_t$ (i.e., $o\_\mathcal{P}_l^R\_t = o\_\mathcal{P}_i^R\_t$). In addition, if there exist no $l' < i$ such that $\mathcal{P}_i^E = \mathcal{P}_l^E$, define new domain $o\_\mathcal{P}_i^E\_d$ that will contain all entities of the considered system that have the set of properties $\mathcal{P}_i^E$. Otherwise, use the same domain $o\_\mathcal{P}_{l'}^E\_d$ (i.e., $o\_\mathcal{P}_{l'}^E\_d = o\_\mathcal{P}_i^E\_d$) defined for the object of the rule r_l.*
- ***Step 4:*** *When associated to the request context C^q of an access query $\mathcal{AQ}$ (i.e., the context of the rule r_i is satisfied by the context C^q of $\mathcal{AQ}$), allow the type c_i_t to be an entry point allowing to transit $\mathcal{AQ}$ from domain $s\_\mathcal{P}_i^S\_d$ to the domain $o\_\mathcal{P}_i^O\_d$.*
- ***Step 5:*** *Authorize access queries that transit from $s\_\mathcal{P}_i^S\_d$ to $o\_\mathcal{P}_i^E\_d$ to perform the actions A_i on any objects having the type $o\_\mathcal{P}_i^R\_t$.*

Finally, we denote $\mathcal{C}$ to be the set containing the set of DTE type c_i_t and their respective context of the rule C_i created in step 2 ($\mathcal{C} = \{(c_i\_t, C_i)\}$).

We note here that only the rules having an *allow* decision are considered in the previous definition. This choice is due to the fact that DTE is using by default closed policies.

In our model, when an access query is created by the system, the query inherits all the types associated to the subject S^q and belongs to the DTE domains of S^q. In addition, we suppose that the system associates types to C^q as follows. $\forall(c_i\_t, C_i) \in \mathcal{C}$: if C_i is satisfied in C^q, then associate the type c_i_t to the request context C^q of the access query.

Example 1. This example illustrate the security policy transformation method we defined in Definition 3. Let us consider that we have a security policy $\mathcal{SP}$ that is composed of three rules r_1, r_2 and r_3 such that:

- $r_1 = \langle S_1 = \{func : web\_server, sec\_level : high\}, A_1 = read, O_1 = \{\mathcal{P}_1^E = \{func : ftp\_server, sec\_level : high\}, \mathcal{P}_1^R = \{file\_name : any\}\}, C_1 = \{\text{between 8am and 8pm}\}, D_1 = allow\rangle$
- $r_2 = \langle S_2 = \{func : web\_server, sec\_level : high\}, A_2 = write, O_2 = \{\mathcal{P}_2^E = func : database\_server, sec\_level : low, \mathcal{P}_2^R = db\_name : service\_db\}, C_2 = \{\text{between 8am and 8pm}\}, D_2 = allow\rangle$

– $r_3 = \langle S_3 = \{func : web\_client, sec\_level : high\}, A_3 = access, O_3 = \{\mathcal{P}_3^E = func : ftp\_server, sec\_level : high, \mathcal{P}_3^R = file\_name : web\_config\}, C_3 = \{\text{between 8am and 8pm}\}, D_3 = allow\rangle$

According to Definition 3, the transformation of the policy $\mathcal{SP}$ to a DTE specification is illustrated using the schema in Fig. 1. Subjects S_1, S_2, of rules r_1, r_2 are respectively represented in the transformation by the DTE domain *s_web_server_high_d* while the subject S_3 of r_3 is represented by the domain *s_web_client_high_d*. The entities of objects O_1 and O_3 of r_1 and r_3 (described using the set of properties P_1^E and P_3^E) are represented in the DTE transformation by the DTE domain *o_ftp_server_high_d* and the resources of the objects O_1 and O_3 (described using the set of properties P_1^R and P_3^R). In the case of O_2, the entities described using the set of properties P_2^E and the resources are described using the set of properties P_2^R. The DTE domains *o_ftp_server_high_d* and *o_db_server_low_d* are respectively created by the transformation of the rules r_1 and r_3. After the transformation, *o_ftp_server_high_d* contains the ftp server having the security level high while *o_db_server_low_d* contains the database servers having the security level low. Finally, *c_t* is a DTE type that will be associated to any access query satisfying the context C_1, C_2, and C_3 of the rules r_1, r_2, and r_3. Let us consider the access query $\mathcal{AQ}$ $:\langle S^q = web\_client, O^q = ftp\_server, A^q = read, C^q = \{\text{query time} = 12 \text{ am}\}\rangle$ to be evaluated and performed on the considered system. According to our

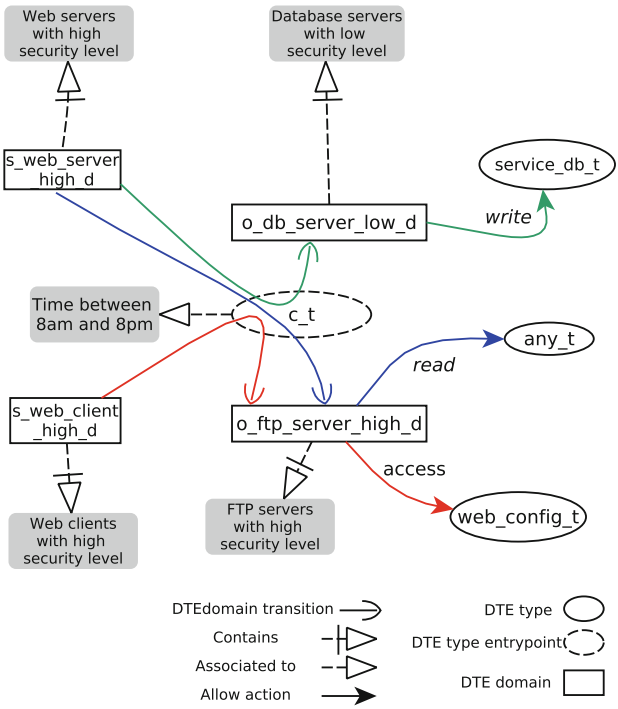

Fig. 1. Transformation from a specific policy to a DTE policy

transformation method, $\mathcal{AQ}$ will inherit the domain of its subject, so it will initially belongs to the domain *s_web_client_high_d*. Moreover, the context C^q of $\mathcal{AQ}$ satisfies the contexts C_1,C_2 and C_3 of the three rules r_1, r_2, and r_3. Hence, the type *c_t* will be associated to the query $\mathcal{AQ}$. According to the step 4 of our transformation method, the DTE type *c_t* will allow the access query $\mathcal{AQ}$ to transit from the domain *s_web_client_high_d* to the domain *o_ftp_server_high_d*. Furthermore, according to the transformation shown in Fig. 1, the action *access* is authorized to be performed by access queries belonging to the domain *o_ftp_server_high_d* over objects associated to the DTE type *o_ftp_server_high_t*. So, we conclude that the query $\mathcal{AQ}$ is to be authorized by the transformation of $\mathcal{SP}$.

Policy Transformation Correctness. A security policy transformation method is correct if, for any access query, no rule in the transformed security policy is violated when the transformation resulting policy is deployed. This is formalized using the following definition.

Definition 4. *Given a security policy* $\mathcal{SP} = \{r_1, \cdots, r_n\}$ *and its corresponding DTE transformation* $\mathcal{SP}_{DTE}$ *(as described in Definition 3). The transformation from* $\mathcal{SP}$ *to* $\mathcal{SP}_{DTE}$ *is correct if and only if for any access query* $\mathcal{AQ}$*: if* $\mathcal{AQ}$ *is allowed (resp. denied) by* $\mathcal{SP}$*, then it is allowed (resp. denied) by* $\mathcal{SP}_{DTE}$*.*

Theorem 1. *The policy transformation method proposed in Definition 3 is correct.*

Proof. We prove the previous theorem by contradiction. Let us denote by $\mathcal{SP}$ the transformed policy and $\mathcal{SP}_{DTE}$ the transformation resulting policy. According to Definition 4, the policy transformation method is not correct if one of the following cases hold:

- **case 1:** There exists an access query $\mathcal{AQ}$ such that it is allowed by $\mathcal{SP}$ and denied by $\mathcal{SP}_{DTE}$.
- **case 2:** There exists an access query $\mathcal{AQ}$ such that it is denied by $\mathcal{SP}$ and allowed by $\mathcal{SP}_{DTE}$.

For both cases, a contradiction is shown in the following.

Case 1: Formally, this case implies that $\exists r_i \in \mathcal{SP}, \exists \mathcal{AQ}$ such that: $S^q \in S_i, O^q \in O_i, A^q \in A_i$, C_i is satisfied in C^q, and D_i = allow. $S^q \in S_i$ means that S^q will belong to the same domain as S_i ($s\_\mathcal{P}^{S_i}\_d$) and that the query itself will belong to $s\_\mathcal{P}^{S_i}\_d$. According to the step 3 of our policy transformation method (Definition 3), $O^q \in O_i$ implies that the object O^q will have the type $o\_\mathcal{P}^{R_i}\_t$. In addition, according to the query initialization rules, C_i is satisfied in C^q means that the type c_i_t will be assigned to C^q. Then, according the step 4 of Definition 3, when executed, $\mathcal{AQ}$ will transit from the domain $s\_\mathcal{P}^{S_i}\_d$ to the domain $o\_\mathcal{P}^{E_i}\_d$. Subsequently, and according to the step 5 of Definition 3, since $\mathcal{AQ}$ transited to $o\_\mathcal{P}^{E_i}\_d$, it will have the permission to perform the set of actions

A_i on all entities having the type $o\_\mathcal{P}^{R_i}\_t$. Finally, since $O^q \in O_i$ and $A^q \in A_i$, then $\mathcal{AQ}$ will have the permission to perform the action A^q on the object O^q which contradicts the hypothesis of the case 1.

Case 2: This case happens if one of the following conditions hold:

case 2.1: Given the access query $\mathcal{AQ}$, there exists no rule in the policy $\mathcal{SP}$ that allow $\mathcal{AQ}$. Formally, $\nexists r_i \in \mathcal{SP}$ such that $S^q \in S_i, O^q \in O_i, A^q \in A_i$, C^q satisfies C_i, and $D_i =$ allow. Let us suppose that the $\mathcal{AQ}$ is allowed by $\mathcal{SP}_{DTE}$. According to the transformation method, action permission is only specified in step 5 of Definition 3. This step means that if $\mathcal{AQ}$ is allowed by $\mathcal{SP}_{DTE}$, then there exist a domain $s\_\mathcal{P}^{S_i}\_d$ and a type $o\_\mathcal{P}^{R_i}\_t$ such that $\mathcal{AQ}$ belongs to $s\_\mathcal{P}^{S_i}\_d$ and O^q has the type $o\_\mathcal{P}^{R_i}\_t$. This means that there exists $r_i \in \mathcal{SP}$ such that $A_q \in A_i$ and $O^q \in O_i$. In addition, according to step 3 of Definition 3, $o\_\mathcal{P}^{R_i}\_d$ $(i \in [1, n])$ does not contain any access query when created. These domains are only accessible for access queries thought the transformation rule defined in step 4 of Definition 3. Since we already showed that $\mathcal{AQ}$ belongs to $s\_\mathcal{P}^{S_i}\_d$, then there exits an entrypoint type c_i_t that allow $\mathcal{AQ}$ to transit to the domain $o\_\mathcal{P}^{E_i}\_d$ which allow us to deduce that C_i is satisfied in C^q and that both S_i and S^q belongs to the same domain $s\_\mathcal{P}^{S_i}\_d$ (since $S^q \in S_i$). Then, we deduce that $\exists r_i \in \mathcal{SP}$ such that $S^q \in S_i$, $O^q \in O_i$, $A^q \in A_i$, C_i satisfied in C^q and $D_i = allow$ which contradicts the case 2.1.

case 2.2: This case implies that given the access query $\mathcal{AQ}$, in one hand $\exists r_i \in \mathcal{SP}, \exists \mathcal{AQ}$ such that: $S^q \in S_i, O^q \in O_i, A^q \in A_i$, C_i is satisfied in C^q, and $D_i =$ deny and in the other hand $\mathcal{AQ}$ is allowed by $\mathcal{SP}_{DTE}$. $S^q \in S_i$ means that S^q will belong to the same domain as S_i ($s\_\mathcal{P}^{S_i}\_d$) and that the query itself will belongs to $s\_\mathcal{P}^{S_i}\_d$, since $\mathcal{AQ}$ inherit the domain of its subject then $\mathcal{AQ}$ belongs also to $s\_\mathcal{P}^{S_i}\_d$. Since the rule r_i is transformed using our transformation method, then there exists the type c_i_t that represents an entrypoint to the domain $o\_\mathcal{P}^{E_i}\_d$. Since C_i is satisfied in C^q, the type c_i_t will be associated to the C^q of $\mathcal{AQ}$, as a result, when executed, $\mathcal{AQ}$ will transit from $s\_\mathcal{P}^{S_i}\_d$ to $o\_\mathcal{P}^{E_i}\_d$. However, based on the transformation of r_i, the domain $o\_\mathcal{P}^{E_i}\_d$ will be denied to perform the action A_i on the type $o\_\mathcal{P}^{R_i}\_t$. Finally, since $O^q \in O_i$ and $A^q \in A_i$ then the query $\mathcal{AQ}$ will be denied by the $\mathcal{SP}_{DTE}$ which contradicts the case 2.2.

Service Requirements Specification

In our model, a security policy is going to be deployed on a VNF service. A VNF service $\mathcal{S}$ is composed of a set of VNFs $\{vnf_1, \cdots, vnf_n\}$ and a set of forwarding graphs $\{fg_1, \cdots, fg_m\}$. Each forwarding graph fg_i is composed of a set of forwarding paths $\{fp_1, \cdots, fp_d\}$, each fp_i can be represented using the following couple $\langle\langle vnf_1^i, vnf_2^i, \cdots, vnf_{n_i}^i\rangle, fp\_m_i\rangle$, where vnf_1^i is the VNF that is forwarding the traffic, vnf_n^i is the VNF to which the traffic is forwarded, and fp_m_i is the match policy that will be used to distinguish which traffic should traverse the path.

In our model, a traffic $\mathcal{T}$ is used to represent each exchange between two consecutive VNFs in the considered forwarding path. It is modeled as the quadruple $\langle vnf\_src, vnf\_dst, t\_context, t\_content\rangle$ where vnf_src, vnf_dst, $t\_context$, and $t\_content$ represent respectively, the VNF that is sending the traffic, the VNF destination of the traffic, the context and the content of the traffic. Formally, a forwarding path $fp = \langle\langle vnf_1, vnf_2, \cdots, vnf_n\rangle, fp\_m\rangle$ is represented using $n-1$ traffics $\mathcal{T}_i = \langle vnf_i, vnf_{i+1}, fp_m, t\_content\rangle$, $1 \leq i < n$.

It is worth highlighting that the action involved in the security policy to be deployed can be implemented in the content of a traffic. For example, the "write" action can be implemented according to the protocol that is used. If the FTP protocol is used, a traffic containing the "post" ftp command implements the action "writ" used in the security policy. Thanks to the previous observation, a traffic can be modelled as an access query as defined in the following.

Definition 5. *A traffic* $\mathcal{T} = \langle vnf\_src, vnf\_dst, t\_context, t\_content\rangle$ *will be modeled as an access query* $\mathcal{AQ} = \langle S^q, O^q, \mathcal{A}^q, C^q\rangle$ *where* vnf_src *equals to* S^q, vnf_dst *equals to* O^q, $\mathcal{A}^q$ *are the actions that can be implemented by the traffic content* $t\_content$, *and* $C^q = t\_context$.

To ensure a proper functioning of the VNF service to be deployed, the traffics that represent each forwarding path should be allowed to flow according to the latter. To meet the previous objective, for each traffic $\mathcal{T}_i = \langle vnf_i, vnf_{i+1}, t_i\_context, t_i\_content\rangle$ that is modeled as the access query $\mathcal{AQ}_i = \langle vnf_i, vnf_{i+1}, \mathcal{A}_i^q, C_i^q\rangle$, we define the following policy rule:

$$r_{\mathcal{T}_i} = \langle S = vnf_i, O = vnf_{i+1}, A = A_i^q, C = C_i^q, D = allow\rangle$$

The previous rule states that vnf_i is allowed to perform the action A_i^q (implemented by the content of the traffic $\mathcal{T}_i$) over vnf_{i+1} if the context C_i^q is satisfied in the considered system. Finally, The previous rule is transformed to a DTE specification as described in Definition 1.

DTE Policy Enforcement. The DTE policy obtained from the transformation of the access control policy to be enforced and the network service to be deployed is enforced using a special VNF that we called VNF_Filter. VNF_Filter will basically analyze the traffics exchanged between the different VNF that compose the deployed network service to evaluate the authorization of each access query. In order to allow this, we should modify (as described in Definition 6) the forwarding graphs used to orchestrate and manage traffic through the VNFs that compose the deployed network service to make sure that these traffics pass certainly through the VNF_Filter.

Definition 6 (Forwarding graph modification). *Given a network service* $\mathcal{S}$ *composed of a set of forwarding graphs* $\{fg_1, \cdots, fg_m\}$. *Each forwarding graph* fg_i *is composed of a set of forwarding paths* $\{fp_1, \cdots, fp_d\}$, *and each* fp_i *is represented by a sequence* $sq_i = \langle vnf_1^i, vnf_2^i, \cdots, vnf_{n_i}^i\rangle$ *of the VNF that represents*

the path that should be traversed by a traffic. Each $sq_i = \langle vnf_1^i, vnf_2^i, \cdots, vnf_{n_i}^i \rangle$ *of a forwarding path* fp_i *will be modified as following:*

$$sq_i = \langle vnf_1^i, \boldsymbol{vnf\_filter}, vnf_2^i, \boldsymbol{vnf\_filter}, \cdots, \boldsymbol{vnf\_filter}, vnf_{n_i}^i \rangle$$

To illustrate, let us consider a forwarding path fp composed of a sequence of three VNFs $\langle vnf1, vnf2, vnf3 \rangle$. The modification of fp according to Definition 6 makes sure that the traffic managed by fp will pass through the VNF_Filter as shown in Fig. 2.

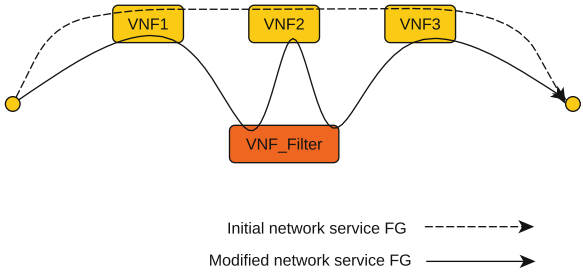

Fig. 2. Network Service forwarding graph modification

The observation of all traffics exchanged between the VNF that compose the considered network service gives VNF_Filter the ability to analyze those traffics and authorize only the ones that are allowed by the considered DTE policy. The pseudo-code in Algorithm 1 outlines the procedure used by the VNF_Filter to authorize a traffic exchanged between two VNF. It takes as inputs the traffic to be authorized, the sets $\mathcal{C}$ of types and their respective contexts created in the policy transformation process (Definition 3). It outputs a value (*allow_traffic* or *deny_traffic*) indicating whether or not the traffic is allowed by the DTE policy. The function *parse_traffic* allow to model the traffic $\mathcal{T}$ into an access query as defined in Definition 5. The functions *get_domains*, *get_types*, *get_transition_src*, and *get_transition_dst* are used to retrieve respectively, the set of domains to which the subject of the access query belong, the set of types associated with the object of the access query, the source domain of the domain transition, and the destination domain of the transition. The function *assign_types* assigns to the access query $\mathcal{AQ}$ the types used in $\mathcal{C}$ if their respective context matches the context of $\mathcal{AQ}$. Finally, the function *check_permission* check whether a DTE domain is allowed or denied to perform the actions in $\mathcal{A}^q$ on a given DTE type.

Input: $\mathcal{T}$ /* the traffic to be authorized */
$\mathcal{C} = \{(c\_r_i\_t, C_i)\})$ /* Definition 3 (step 2) */

$\mathcal{AQ} : \langle S^q, O^q, \mathcal{A}^q, C^q \rangle =$ **parse_traffic**$(\mathcal{T})$
subject_domains = **get_domains**(S^q)
object_types = **get_types**(O^q)
$\mathcal{AQ}$*_types* = **assign_types**$(\mathcal{AQ}, \mathcal{C})$
possible_domains = $\emptyset$
foreach $\mathcal{AQ}$*_type* $\in$ $\mathcal{AQ}$*_types* **do**
  if $\mathcal{AQ}$*_type is not a DTE entrypoint* **then continue** ;
  if ***get_transition_src***$(\mathcal{AQ}$*_type*$) \notin$ *subject_domains* **then**
    continue ;
  possible_domains = *possible_domains* $\cup$
    get_transition_dst$(\mathcal{AQ}$*_type*$)$
end
is_allowed = false
foreach *type* $\in$ *object_types* **do**
  foreach *domain* $\in$ *possible_domains* **do**
    if ***check_permission***$($*domain*, *type*, $\mathcal{A}^q)$ = *allow* **then**
      is_allowed = true;
    else if ***check_permission***$($*domain*, *type*, $\mathcal{A}^q)$ = *deny* **then**
      return deny_traffic;
  end
end
if *is_allowed* **then return** allow_traffic ;

Algorithm 1: Access query authorization

5 Implementation and Experimental Evaluations

This section presents the implementation details of a prototype of our proposed access control model. The design architecture of the prototype implementing the proposed model is illustrated in Fig. 3. The major functional components are described in the following.

- **OpenStack Tacker** [22]**:** it is an official OpenStack project that orchestrates and manages infrastructure resources and maintain the lifecycle management of network services and VNF instances over the OpenStack infrastructure.
- **Access control engine (ACE):** it works together with the VNF orchestrator (Taker) security policy enforcement to the deployed VNF service components (e.g., VMs, VNFs.)
- **OpenFlow Manager of OpenDaylight** (ODL) [1] is an open-source application development and delivery platform. OpenFlow Manager (OFM) [2] is an application that runs on top of ODL allowing to visualize OpenFlow topologies, program network traffic flow paths and gather network traffic stats.

– **OpenStack Infrastructure:** OpenStack as a virtual infrastructure manager (VIM) layer is used to give a standardized interface for managing, monitoring and assessing all resources within VNF infrastructure.

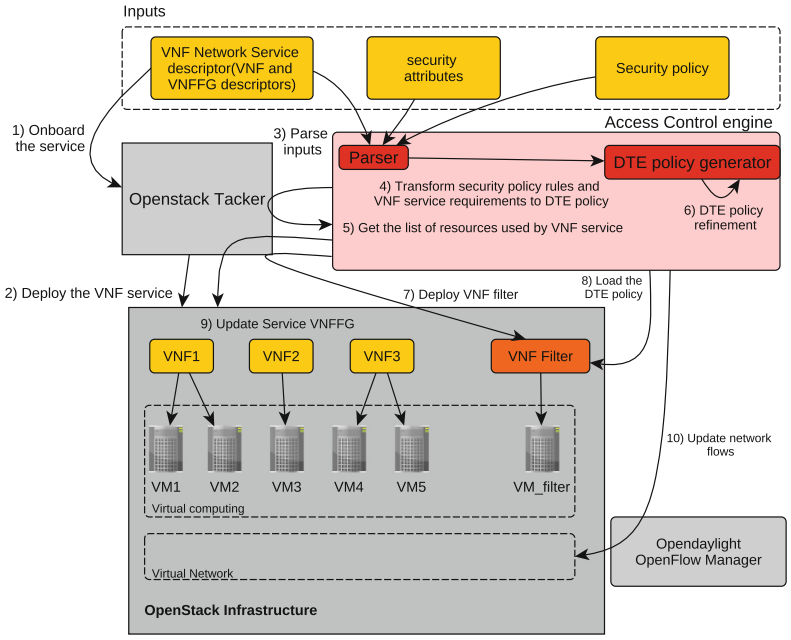

Fig. 3. Design architecture of the implementation of the proposed model and the operational flow of an access control policy deployment

In addition, Fig. 3 illustrates the different steps that are implemented in order to deploy an access control policy on a VNF network service. In the following, more details on each step are given.

– Onboard and deploy the network service (Steps 1 and 2): In these steps, Tacker uses the network service descriptor provided as an input to onboard and deploy the network service on the OpenStack infrastructure.
– Access control policy parsing and transformation (Steps 3 and 4): In these steps, the access policy engine parses the VNF service descriptor, the security properties associated which the different VNF that compose the network service (e.g., security level, trust level, etc.), and the access control policy to be deployed. Then, it transforms the access control policy to a DTE policy as described in Definition 3.
– DTE policy refinement (Steps 5 and 6): The ACE engine queries Tacker to get the set of resources (e.g., VMs, Connection points, networks, etc.) that are used to deploy the different VNF that compose the deployed service. Then it refines the DTE-policy at the resources level of the VNF service.

– Policy enforcement (Steps 7, 8, 9 and 10): To enforce the DTE policy, the ACE first uses Tacker to deploy VNF_Filter which is a special VNF that implements a DTE engine we developed in python [21]. Second, it loads the refined DTE policy to be enforced over the deployed VNF service on VNF_Filter. Third, ACE updates the forwarding graphs of the deployed network service (as illustrated in Fig. 2) and uses OpenFlow Manager of ODL to make sure that all network flows exchanged between the VNF that compose the deployed network service will transit through VNF_Filter. Once VNF_Filter receives a network traffic, it starts by parsing the traffic to extract its source and its destination as well as the actions that are implemented by its content. Finally it uses the DTE engine to check whether the traffic is allowed to transit from its source to its destination i.e., the actions that are implemented by the content of the traffic are allowed to be performed by the traffic source component over the traffic destination component.

We experimentally evaluate the performance of our approach. Our access control engine prototype is hosted in a server running Linux with an Intel Xeon E5-2680 v4 Processor with 8 vCPU and 16 GB of RAM while our implementation of the VNF filter including the implementation of the DTE engine is hosted in a virtual machine running Linux having a processor with 2 vCPU and 2 GB of RAM. In our empirical evaluation, we aim to quantify the following characteristics of our approach. First, the time needed to transform an access control policy to a DTE specification as a function of the number of rules of the considered access control policy is quantified. The obtained results are depicted in Fig. 4.

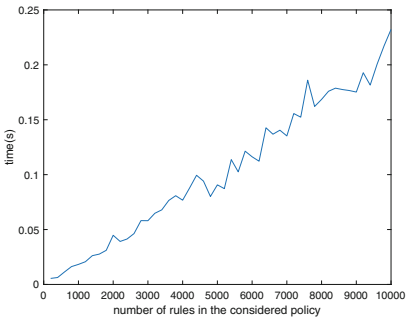

Fig. 4. Policy transformation time

They show that transformation method we are proposing is quite efficient since it takes around 230 ms to transform an access control policy composed of 10^4 rules to a DTE specification. The time needed to transform a security policy to a DTE specification grows linearly in function of the number of rules in policy. Second, we quantify the round-trip time (RTT) required for a packet as a function of the activation of our VNF_Filter (i.e., we aim to compare the

RTT when our VNF_filter is used and when it is not) and the number of rules in the considered access control policy.

Figure 5 reports a linear growth of the measured RTT in function of the number of rules in the policy to be deployed. It shows that our implementation introduces less than 2 ms delay when a policy composed of 500 rules is considered.

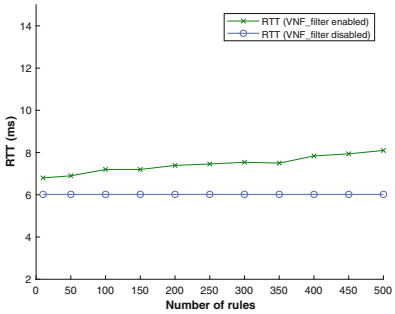

Fig. 5. RTT as a function of the number of rules in the access policy to be deployed

6 Conclusion

This paper proposes an access control as a service model to improve security management in the context of NFV. We firstly investigated several existing NFV orchestrators and several existing access control model in NFV infrastructure and observed that (1) none of them provides a generic model and (2) they are often not fully compliant with the ETSI NFV infrastructure in the sense that the deployment of the access control policies requires often the modification of the NFVI infrastructure. The previous observations motivate us to define a new software-defined access control as a service model. Compared to existing models, our proposition offers several advantages to VNF users. First, it is generic in the sense that they can deploy most types of access control policy such as RBAC, ORBAC, ABAC, etc. Second, it complies with the ETSI-NFV infrastructure because it does not require any modification of the latter for policy deployment. The conducted experimentations show that the implementation of the proposed model is quite efficient. The deployment of a security policy composed of 500 rules introduces less than 2 ms delay for the round-trip time of a network packet. As a future work, we aim to extend our proposed model to allow to check whether a given network service deployment satisfies the requirements of a given security policy.

References

1. The OpenDaylight Platform. https://www.opendaylight.org/. Accessed 30 Jan 2019

2. Openflow Manager. https://github.com/CiscoDevNet/OpenDaylight-Openflow-App. Accessed 30 Jan 2019
3. VNFFG. https://docs.openstack.org/tacker/latest/user/vnffg_usage_guide.html. Accessed 1 Jan 2019
4. Basile, C., Lioy, A., Pitscheider, C., Valenza, F., Vallini, M.: A novel approach for integrating security policy enforcement with dynamic network Virtualization. In: Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft), pp. 1–5. IEEE (2015)
5. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: portable automated deployment and management of cloud applications. In: Bouguettaya, A., Sheng, Q., Daniel, F. (eds.) Advanced Web Services, pp. 527–549. Springer, New York (2014). https://doi.org/10.1007/978-1-4614-7535-4_22
6. ETSI NFV: Management and orchestration; VNF packaging specification. Technical report, DGS/NFV-IFA011
7. Guija, D., Siddiqui, M.S.: Identity and access control for micro-services based 5G NFV platforms. In: Proceedings of the 13th International Conference on Availability, Reliability and Security, pp. 1–10 (2018)
8. Hardt, D.: The OAuth 2.0 Authorization Framework. RFC 6749, October 2012. https://doi.org/10.17487/RFC6749. https://rfc-editor.org/rfc/rfc6749.txt
9. Hu, V.C., Kuhn, D.R., Ferraiolo, D.F., Voas, J.: Attribute-based access control. Computer **48**(2), 85–88 (2015)
10. Jaeger, B.: Security orchestrator: introducing a security orchestrator in the context of the ETSI NFV reference architecture. In: 2015 IEEE Trustcom/BigDataSE/ISPA, vol. 1, pp. 1255–1260. IEEE (2015)
11. Kalam, A.A.E., et al.: Organization based access control. In: Proceedings of the IEEE 4th International Workshop on Policies for Distributed Systems and Networks, POLICY 2003, pp. 120–131. IEEE (2003)
12. Lal, S., Taleb, T., Dutta, A.: NFV: security threats and best practices. IEEE Commun. Mag. **55**(8), 211–217 (2017)
13. Mauricio, L.A., Rubinstein, M.G., Duarte, O.C.M.: ACLFLOW: an NFV/SDN security framework for provisioning and managing access control lists. In: 2018 9th International Conference on the Network of the Future (NOF), pp. 44–51. IEEE (2018)
14. Montero, D., et al.: Virtualized security at the network edge: a user-centric approach. IEEE Commun. Mag. **53**(4), 176–186 (2015)
15. Pattaranantakul, M., He, R., Meddahi, A., Zhang, Z.: SecMANO: towards Network Functions Virtualization (NFV) based security management and orchestration. In: 2016 IEEE Trustcom/BigDataSE/ISPA, pp. 598–605. IEEE (2016)
16. Pattaranantakul, M., He, R., Zhang, Z., Meddahi, A., Wang, P.: Leveraging Network Functions Virtualization orchestrators to achieve software-defined access control in the clouds. IEEE Trans. Depend. Secure Comput. (2018)
17. Pattaranantakul, M., Tseng, Y., He, R., Zhang, Z., Meddahi, A.: A first step towards security extension for NFV orchestrator. In: Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, pp. 25–30 (2017)
18. Reynaud, F., Aguessy, F.X., Bettan, O., Bouet, M., Conan, V.: Attacks against Network Functions Virtualization and software-defined networking: state-of-the-art. In: 2016 IEEE NetSoft Conference and Workshops (NetSoft), pp. 471–476. IEEE (2016)
19. Sakimura, N., Bradley, J., Jones, M.B., de Medeiros, B., Mortimore, C.: OpenID connect core 1.0. https://openid.net/specs/openid-connect-core-1_0.html

20. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. Computer **29**(2), 38–47 (1996)
21. Smine, M.: DTE engine. https://github.com/msmine/vnf-access-control-as-a-service
22. Tacker, O.: Tacker-openstack NFV orchestration (2017)
23. Thanh, T.Q., Covaci, S., Corici, M., Magedanz, T.: Access control management and orchestration in NFV environment. In: 2017 IFIP Networking Conference (IFIP Networking) and Workshops, pp. 1–2. IEEE (2017)
24. Yakasai, S.T., Guy, C.G.: FlowIdentity: software-defined network access control. In: 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), pp. 115–120. IEEE (2015)