A Flexible Architecture for Orchestrating Network Security Functions to Support High-Level Security Policies

Sanghak Oh[1], Eunsoo Kim[2], Jaehoon (Paul) Jeong[3], Hoon Ko[2], Hyoungshick Kim[1]
[1]Department of Software, Sungkyunkwan University, Republic of Korea
[2]Department of Computer Science and Engineering, Sungkyunkwan University, Republic of Korea
[3]Department of Interaction Science, Sungkyunkwan University, Republic of Korea
{osh09, eskim86, pauljeong, skoh21, hyoung}@skku.edu

ABSTRACT

Network Functions Virtualization (NFV) has provided a new way to design and deploy network security services, but it may fail to build a practically useful ecosystem that seamlessly integrates network security services if there is no standard interface between them. We propose a generic architecture for security management service based on Network Security Functions (NSF) using NFV. The proposed architecture allows users to define their security requirements in a user-friendly manner by providing the users with high-level security interfaces that do not require specific information about network resources and protocols. We design basic components (e.g., Security policy manager, NSF capability manager, Application logic, Policy updater and Event collector) and interfaces for the proposed architecture. We introduce three use cases: (1) blacklists of dangerous domains, (2) time-dependent access control policies and (3) detection of suspicious calls for VoIP-VoLTE services. We also explain how to implement our proposed architecture with an illustrative example. Furthermore, we discuss several technical challenges to deploy the proposed architecture in a real network environment.

CCS Concepts

•**Networks** → *Network architectures; Middle boxes / network appliances; Network management;*

Keywords

Security management, NFV, NSF, Security policy

1. INTRODUCTION

Network Functions Virtualization (NFV) is an emerging area for the network industry [1]. NFV promises to reduce the cost of deploying and maintaining networks by decoupling network functions from dedicated hardware appliances and implement those functions as pure software instances running on general purpose commodity servers [2]. Network Security Functions (NSF), such as firewall, Intrusion Detection System (IDS) and Intrusion Protection System (IPS), can also be provided as virtual network functions that might be automatically provisioned and dynamically migrated based on real-time security requirements. In this paper, we focus on NSF rather than general-purpose NFV.

To successfully deploy NFV-based security applications, standardization is critical because NSFs are often developed by different vendors and/or managed by different network operators. Recently, some basic standard interfaces to control NSFs are being developed by Interface to Network Security Functions (I2NSF) working group [3], which is part of an International Internet Standardization Organization called Internet Engineering Task Force (IETF) [4]. Thus, within a few years, various NSFs will be remotely controlled by a network entity called *security controller*, which is a central management entity for NFV-based security services, through standard interfaces [5].

However, there is still room for standards development in NFV-based security applications because security controllers must also communicate with any NSF client (e.g., I2NSF client) that is capable of creating and managing security policies on networks. In this paper, we propose a layered architecture to seamlessly integrate any NFV-based security application managed by a security controller into NFV-enabled networks. With this architecture, application users can enforce their high-level security policies in a user-friendly manner.

The rest of this paper is organized as follows. Section 2 presents the proposed architecture for security management services based on NFV. Section 3 explains the three key use cases using the proposed architecture. Section 4 describes how to implement the proposed architecture in practice with an illustrative case. Next, Section 5 discusses technical challenges on implementing our architecture. Related work is summarized and analyzed in Section 6. Finally, we conclude in Section 7.

2. ARCHITECTURE

In this paper, we propose a layered architecture that integrates additional components for security management services based on NFV. To support flexible and effective security policy enforcement, the proposed architecture consists of the three layers: (1) NSF client, (2) Security management system and (3) NSF instances (see Figure 1). Here, arrows denote communication between functional components. A bidirectional arrow indicates the interaction between two

IMCOM '17, January 05-07, 2017, Beppu, Japan
© 2017 ACM. ISBN 978-1-4503-4888-1/17/01…$15.00
DOI: http://dx.doi.org/10.1145/3022227.3022270

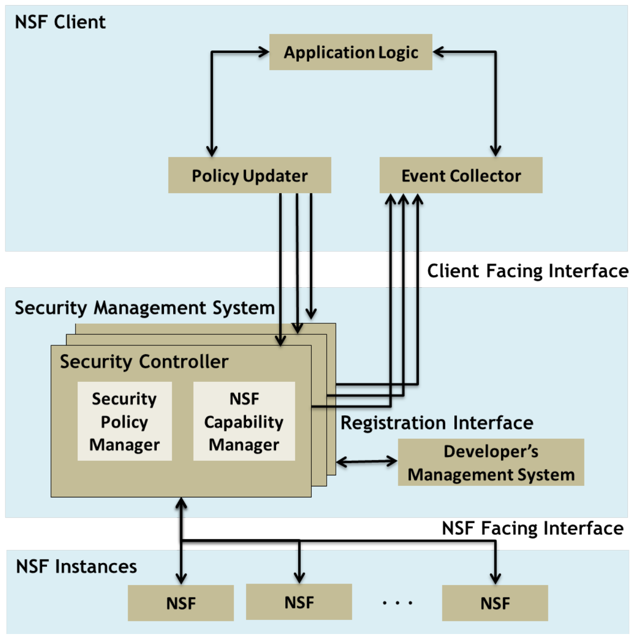


Figure 1: Generic architecture of security management based on NFV.

components in both directions and an unidirectional arrow indicates the interaction between two components only in the direction defined by the arrow.

The architecture is designed to support the enforcement of flexible and effective security policies. We use the term NSF client to refer to NFV-based security application. In a NSF client, Application logic generates high-level security policies; Policy updater distributes such policies to Security controllers through *Client Facing Interface*. In a Security controller, Security policy manager maps the high-level policies into low-level security policies relevant to NSF capability that is registered into NSF capability manager. After mapping, Security policy manager delivers those policies to NSF(s) through *NSF Facing Interface*. We explain those operations at the network components in detail.

2.1 Security policy manager

Security policy manager is a component which receives a high-level policy from Policy updater via *Client Facing Interface*, and maps the high-level policy into several low-level policies relevant to a given NSF capability that is registered into NSF capability manager. Moreover, Security policy manager delivers those policies to NSF(s) via *NSF Facing Interface*.

On the other hand, when an event that requires the low-level policy to be changed happens in NSF, NSF sends the event to Security policy manager via *NSF Facing Interface*. Security policy manager then sends it to Event collector via *Client Facing Interface*.

2.2 NSF capability manager

NSF capability manager is a component integrated into Security controller. It stores an NSF's capability registered by Developer's management system via *Registration Interface* and shares it with Security policy manager so that Security policy manager can generate low-level policies relevant to a given NSF capability. Moreover, whenever a new NSF is registered, NSF capability manager requests Developer's management system to register the NSF's capability into the management table of NSF capability manager via *Registration Interface*. When the existing NSF is deleted, NSF capability manager eliminates the NSF's capability from its management table.

2.3 Developer's management system

Developer's management system is a component which registers a new NSF's capability into NSF capability manager via *Registration Interface*. When there is an update in the registered NSF, it is delivered from Developer's management system to NSF capability manager.

2.4 Application logic

Application logic is a component which generates a high-level security policy to mitigate security attacks. It receives the event for updating (or generating) a high-level policy from Event collector and updates (or generates) a high-level policy based on the collected events. Application logic then sends the high-level policy to Policy updater in order to forward a recently updated policy. In Section 3, we will explain how Application logic is designed through the three use cases.

2.5 Policy updater

Policy updater is a component which receives a high-level security policy generated by Application logic and distributes it to Security controller(s) via *Client Facing Interface*.

2.6 Event collector

Event collector receives an event, which should be reflected on updating (or generating) a high-level policy in Application logic, from Security controller. The procedure of receiving an event in NSF is necessary because a low-level security policy can be updated according to an event that occurred in an NSF. After receiving it, Event collector forwards it to Application logic so that Application logic can update (or generate) a high-level security policy based on the event received from Security controller.

3. USE CASES

A generic architecture based on NFV is designed to react to possible security attacks. This section shows the procedure of the defense for security attacks in blacklists of dangerous domains, time-dependent access control policies, and detection of suspicious calls for VoIP-VoLTE services.

3.1 Blacklists of dangerous domains

Dangerous domain (e.g., used for malware distribution) blacklisting maintains and publishes the blacklists of IP addresses of possible attacking hosts, servers and networks that are suspicious of malicious activities. For the security management architecture for dangerous domain blacklisting, Dangerous domain manager takes the role of Application logic to perform security management.

Based on the dangerous domain blacklisting, the list of dangerous domains is stored in Dangerous domain database and can be updated either manually or automatically by Dangerous domain manager as Application logic. Also, Dangerous domain manager periodically loads the list of dangerous domains from Dangerous domain database and generates a new high-level security policy (e.g., blocking the list of dangerous domains using their IP addresses) to prevent the delivery of packets from/to those newly added dangerous domains. It sends the new high-level security policy to Policy updater, which distributes it to Security controller(s). Security controller maps the high-level policy into low-level policies and enforces the low-level security policies in NSF.

When NSF detects a new dangerous domain, the corresponding IP addresses are sent by an NSF to Security controller via *NSF Facing Interface*. Security controller delivers the IP addresses to Event collector. Event collector forwards the IP addresses to Dangerous domain manager, and then Dangerous domain manager updates the Dangerous domain database.

3.2 Time-dependent access control policies

Time-dependent access control policies manage a user's access to particular websites during a certain period of time. For example, in a company, a manager blocks employees' access to Youtube website, which is a big distraction during working hours.

Based on time-dependent access control, NSF clients register the list of blocking websites and time at Application logic. Application logic stores the list into database and generates a high-level security policy (e.g., blocking the access to websites by checking the blocking websites and time). Application logic delivers it to Policy updater, and then Policy updater forwards it to Security controller. In Security controller, Security policy manager maps the high-level policy to low-level policies, and then it sends and enforces them to NSFs.

3.3 Detection of suspicious calls for VoIP-VoLTE services

VoIP-VoLTE security management maintains and publishes the blacklists of IP addresses, source ports, expire time, user-agent and Session Initiation Protocol (SIP) URIs of an SIP device that are suspicious of illegal call or authentication. In our generic security management architecture, VoIP-VoLTE security manager acts as Application logic for VoIP-VoLTE security services in Figure 1.

Based on VoIP-VoLTE security management, the list of illegal devices information is stored in VoIP-VoLTE database and can be updated either manually or automatically by VoIP-VoLTE security manager as Application logic. Also, VoIP-VoLTE security manager periodically loads the list of illegal devices information from VoIP-VoLTE database and generates a new high-level security policy (e.g., the blocking list of illegal devices using IP address, source ports, etc) to prevent the delivery of packets from/to those newly added VoIP-VoLTE attackers. It sends the new high-level security policy to Policy updater, which distributes it to Security controllers. Security controller maps the high-level policy into several low-level policies and enforces the low-level security policies in an NSF.

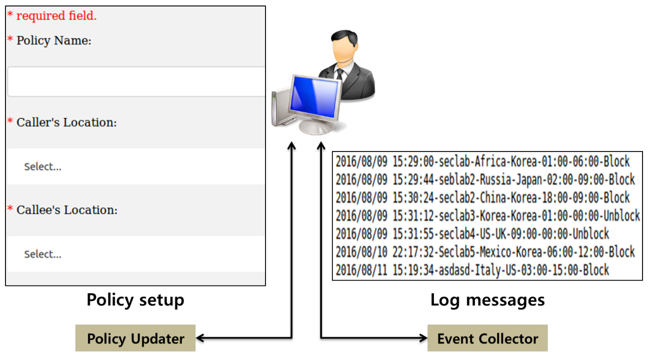
When the NSF detects an anomalous message or call delivered from a domain, the information of the domain such as an IP address, user-agents and expire time values is sent by an NSF to Security controller via *NSF Facing Interface*. Security controller delivers it to Event collector. Event collector forwards the detected domain information to VoIP-VoLTE security manager, and then VoIP-VoLTE security manager updates the VoIP-VoLTE database.

4. IMPLEMENTATION

In this section, we explain how to implement each component and interface in our proposed architecture. We consider the use case in Section 3.3 as our implementation scenario. Through this implementation, our goal is to block suspicious VoIP-VoLTE calls by checking whether an incoming call has fraudulent call behaviors; for example, the call is made from a blacklisted location at an unusual time of day.

4.1 NSF client

To provide user-friendly and more accessible management service to an administrator, we build a web server and create a couple of web pages to provide user interfaces for the administrator to set high-level security policies. In order for the administrator to manage the security policy, we consider the two web pages: (1) the policy setup page for Policy updater and (2) the log messages page for Event collector (see Figure 2). We adopt YANG [6] to define data models for the communication between NSF client and Security management system because YANG is popularly used to model configuration and state data manipulated by standard network protocols such as RESTCONF [7] that provides a programmatic interface over HTTP to access data that is defined in a YANG model.



```
2016/08/09 15:29:00-seclab-Africa-Korea-01:00-06:00-Block
2016/08/09 15:29:44-seblab2-Russia-Japan-02:00-09:00-Block
2016/08/09 15:30:24-seclab2-China-Korea-18:00-09:00-Block
2016/08/09 15:31:12-seclab3-Korea-Korea-01:00-00:00-Unblock
2016/08/09 15:31:55-seclab4-US-UK-09:00-00:00-Unblock
2016/08/10 22:17:32-Seclab5-Mexico-Korea-06:00-12:00-Block
2016/08/11 15:19:34-asdasd-Italy-US-03:00-15:00-Block
```

Figure 2: User interfaces in NSF client.

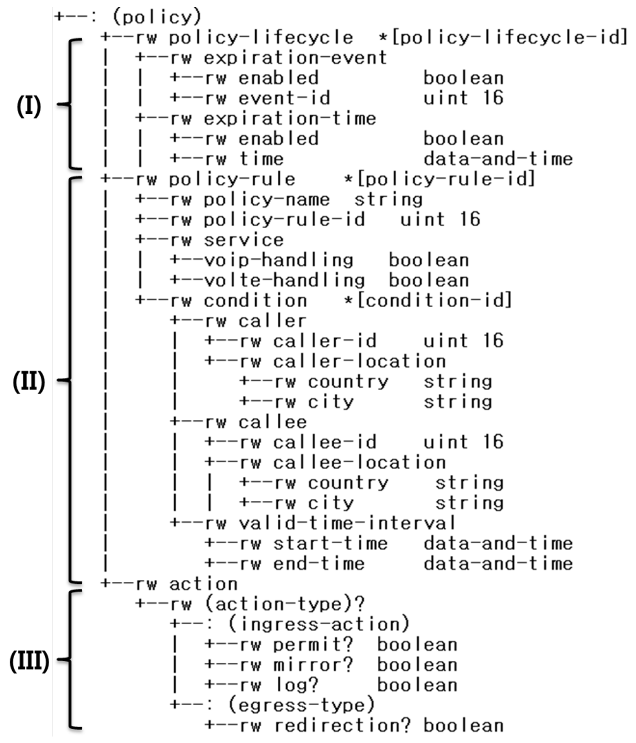
In the policy setup page, we create some fields for defining high-level security policies such as blacklisting countries during a specified time interval. If the administrator set a new high-level security policy, a data model parser in NSF client interprets the policy and generates an XML file in accordance with YANG data model.

In the log messages page, we show the information about events to report the results of security applications and/or the status of functional components at Security management system and NSF instances when the events are delivered from Security management system to Event collector.

4.2 Client Facing Interface

In order to enable interaction between NSF client and Security management system, we implement a communication channel based on RESTCONF. Moreover, we prefer RESTCONF instead of Network Configuration (NETCONF) protocol [8] since NSF client is based on web applications in our implementation.

We also design a data model based on security policy requirements [9] because there is no standardized data model for *Client Facing Interface* yet. In Figure 3, we show a part



```
+--: (policy)
   +--rw policy-lifecycle  *[policy-lifecycle-id]
   |  +--rw expiration-event
   |  |  +--rw enabled          boolean
   |  |  +--rw event-id         uint 16
   |  +--rw expiration-time
   |  |  +--rw enabled          boolean
   |  |  +--rw time             data-and-time
   +--rw policy-rule    *[policy-rule-id]
   |  +--rw policy-name  string
   |  +--rw policy-rule-id   uint 16
   |  +--rw service
   |  |  +--voip-handling   boolean
   |  |  +--volte-handling  boolean
   |  +--rw condition   *[condition-id]
   |     +--rw caller
   |     |  +--rw caller-id    uint 16
   |     |  +--rw caller-location
   |     |     +--rw country   string
   |     |     +--rw city      string
   |     +--rw callee
   |     |  +--rw callee-id    uint 16
   |     |  +--rw callee-location
   |     |  |  +--rw country    string
   |     |  |  +--rw city       string
   |     +--rw valid-time-interval
   |        +--rw start-time   data-and-time
   |        +--rw end-time     data-and-time
   +--rw action
      +--rw (action-type)?
         +--: (ingress-action)
         |  +--rw permit?  boolean
         |  +--rw mirror?  boolean
         |  +--rw log?     boolean
         +--: (egress-type)
            +--rw redirection? boolean
```

Figure 3: Data model in Client Facing Interface.

of our data model design related to a policy management for detecting suspicious calls in VoIP-VoLTE services.

We design a generic data model to apply a policy to unknown attacks and conditions. Our data model consists of (I) policy life cycle management, (II) policy rule, and (III) action. (I) The policy life cycle field specifies an expiration time and/or a set of expiration events to determine the lifetime of the policy itself. (II) The policy rule field represents the specific information about a high-level policy such as service types, conditions and valid time interval. (III) The action field specifies which actions should be taken. For example, call traffic from a blacklisted caller location at an unusual time of day (included in the `valid-time-interval`) could be blocked and sequentially forwarded to a pre-defined host for Deep Packet Inspection (DPI) when both `permit` and `mirror` are assigned `true`.

4.3 Security management system

The main role of Security management system is to translated a high-level policy into a set of low-level policy. For example, Security management system maps a country name into a set of IP addresses by using a geolocation database that provides IP addresses for the country. After translating the high-level security policy, Security management system generates low-level security policies to specify the actions network traffic from and/or to those IP addresses. The data model parser generates an XML file for a low-level security policy and delivers it to proper NSF instances. Security management system also interprets security events generated by NSF into a high-level log message in an YANG data model and delivers it to NSF clients in the opposite direction.

4.4 NSF Facing Interface

Similar to *Client Facing Interface*, *NSF Facing Interface* also uses the RESTCONF protocol and YANG data model in our implementation. I2NSF is currently working to define the standard data models and protocols for *NSF Facing Interface* [5].

4.5 NSF instances

In our use case, we select a firewall application as an NSF instance to determine whether a VoIP-VoLTE call is suspicious or not by checking the caller's and callee's locations and call time. When a call has suspicious behavior patterns, its network traffic could be effectively blocked by the firewall application according to the low-level security policy. The results for the firewall application would be delivered in an YANG data model to the Security management system through the RESTCONF protocol.

Multiple NSF instances can be considered depending on specific situations. For example, we can additionally use DPI for analyzing the network traffic from suspicious callers.

5. KEY CHALLENGES

In this section, we discuss technical challenges on implementing our architecture. Our technical challenges include the followings:

- As Policy updater updates Security controllers with the recent high-level policies, the update time instants on Security controller may be different, and the inconsistency of the high-level security policy could occur during this update process similar to the inconsistency of configuration during the update process which is commonly seen in SDN switches [10].
- A single Security controller will not be able to handle the increasing number of NSF clients because Security controller cannot scale up with the incoming policy flows, thereby causing scalability problem [11].
- A secure and authenticated communication channel should be established between network entities (e.g., NSF client and Security management system). Without ensuring such a communication channel, inappropriate security policies can be maliciously modified by attackers in transfer. Thus, an efficient key management is required in order to distribute keys properly to network entities.
- When Security controllers process high-level and low-level policies, processing sequences could cause a synchronization problem on both Security controllers and NSFs. We should define a proper scheduling model in Security controller to prevent this synchronization problem from happening.
- In order to create high-level policies which will be delivered through *Client Facing Interface*, we should first define a generic policy data model in *Client Facing Interface*. we can use the generic data model of Simplified Use of Policy Abstractions (SUPA) which makes it easy manage the policies regardless of the form and content [12].

On implementing our architecture, we should consider all these challenges to improve the performance of our system.

6. RELATED WORK

The interest in using network function virtualization for security services has been growing steadily in the networking community. Battula [13] described how NFV can be applied to a specific NSF. The network security functions could be used to detect, block or mitigate suspicious and dangerous network activities with flexible and dynamic service requirements. Mahdi et al. [14] discussed the security challenging issues in NFV and their corresponding solutions.

However, without standard interfaces and specifications for NSFs, it is not possible to integrate and manage NSFs in a seamless manner. In particular, lack of standard interfaces for high-level security policies makes it difficult to deploy NSF-based applications into the real world. I2NSF [5] and ETSI NFV [15] are actively working to define a reasonable standard framework using NSF based on NFV.

In this paper, we present an architecture that allows clients to configure and manage high-level security policies to control NSF instances without the detailed implementation related to the NSFs. We develop the data model for *Client Facing Interface* to satisfy the security requirements discussed in I2NSF [9] and follow the generic data model of SUPA [12].

7. CONCLUSION

In this paper, we presented a generic architecture for security management based on NSF using NFV. We also explained how the proposed framework can be to mitigate several practical attack scenarios such as the blacklists of dangerous domains, time-dependent access control policies and the detection of suspicious calls for VoIP-VoLTE services. We particularly introduced the detailed implementation of the proposed architecture with an illustrative example. In future work, we will fully implement our proposed framework to show its feasibility for mitigating various network attacks.

8. ACKNOWLEDGEMENTS

This work was supported in part by the NRF Korea (No. 2014R1A1A1003707), the ITRC (IITP-2015-H8501-15-1008), and the MSIP/IITP (R0166-15-1041, R-20160222-002755).

9. REFERENCES

[1] Hassan Hawilo, Abdallah Shami, Maysam Mirahmadi, and Rasool Asal. NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC). *IEEE Network*, 28(6):18–26, 2014.

[2] Arsany Basta, Wolfgang Kellerer, Marco Hoffmann, Hans Jochen Morper, and Klaus Hoffmann. Applying NFV and SDN to LTE mobile core gateways, the functions placement problem. In *Proceedings of the 4th Workshop on All things cellular: operations, applications, & challenges*, pages 33–38, 2014.

[3] IETF Interface to Network Security Functions (i2nsf) Working Group. https://datatracker.ietf.org/wg/i2nsf/charter/.

[4] The Internet Engineering Task Force (IETF). https://ietf.org/.

[5] Edward Lopez, Diego Lopez, Linda Dunbar, John Strassner, Xiaojun Zhuang, Joe Parrott, Ram (Ramki) Krishnan, and Seetharama Rao Durbha. Framework for Interface to Network Security Functions. IETF Internet-Draft draft-ietf-i2nsf-framework-02, July 2016. http://www.ietf.org/internet-drafts/draft-ietf-i2nsf-framework-02.txt.

[6] M. Bjorklund. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). IETF RFC 6020, October 2010. http://www.rfc-editor.org/rfc/rfc6020.txt.

[7] Andy Bierman, Martin Bjorklund, and Kent Watsen. RESTCONF Protocol. IETF Internet-Draft draft-ietf-netconf-restconf-16, August 2016. http://www.ietf.org/internet-drafts/draft-ietf-netconf-restconf-16.txt.

[8] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman. Network Configuration Protocol (NETCONF). IETF RFC 6241, June 2011. http://www.rfc-editor.org/rfc/rfc6241.txt.

[9] Rakesh Kumar, Anil Lohiya, Dave Qi, and Xiaobo Long. Client Interface for Security Controller : A Framework for Security Policy Requirements. IETF Internet-Draft draft-kumar-i2nsf-client-facing-interface-req-00, August 2016. http://www.ietf.org/internet-drafts/draft-kumar-i2nsf-client-facing-interface-req-00.txt.

[10] Mark Reitblatt, Nate Foster, Jennifer Rexford, and David Walker. Consistent updates for software-defined networks: Change you can believe in! In *Proceedings of the 10th Workshop on Hot Topics in Networks*, page 7, 2011.

[11] Soheil Yeganeh, Amin Tootoonchian, and Yashar Ganjali. On scalability of software-defined networking. *IEEE Communications Magazine*, 2(51):136–141, 2013.

[12] Joel M. Halpern and John Strassner. Generic Policy Data Model for Simplified Use of Policy Abstractions (SUPA). IETF Internet-Draft draft-ietf-supa-generic-policy-data-model-00, July 2016. http://www.ietf.org/internet-drafts/draft-ietf-supa-generic-policy-data-model-00.txt.

[13] Laxmana Rao Battula. Network Security Function Virtualization(NSFV) towards Cloud computing with NFV Over Openflow infrastructure: Challenges and novel approaches. In *Proceedings of the 3rd International Conference on Advances in Computing, Communications and Informatics*, pages 1622–1628, 2014.

[14] Mahdi Daghmehchi Firoozjaei, Jaehoon (Paul) Jeong, Hoon Ko, and Hyoungshick Kim. Security challenges with network functions virtualization. *Future Generation Computer Systems*, 2016.

[15] GSNFV ETSI. Network functions virtualisation (NFV): Architectural framework. *ETSI GS NFV*, 2(2):V1, 2013.