A Framework for Managing User-defined Security Policies to Support Network Security Functions

Eunsoo Kim
Sungkyunkwan University
Republic of Korea
eskim86@skku.edu

Kuyju Kim
Sungkyunkwan University
Republic of Korea
kuyjukim@skku.edu

Seungjin Lee
Sungkyunkwan University
Republic of Korea
jine33@skku.edu

Jaehoon (Paul) Jeong
Sungkyunkwan University
Republic of Korea
pauljeong@skku.edu

Hyoungshick Kim
Sungkyunkwan University
Republic of Korea
hyoung@skku.edu

ABSTRACT

Network Functions Virtualization (NFV) and Software Defined Networking (SDN) make it easier for security administrators to manage security policies on a network system. However, it is still challenging to map high-level security policies defined by users into low-level security policies that can be applied to network security devices. To address this problem, we introduce a framework for effectively managing user-defined security policies for network security functions based on standard interfaces that are currently being standardized in an IETF working group. To show the feasibility of the proposed framework, we implemented a prototype based on the RESTCONF protocol and showed that the proposed framework can be applied in real-world scenarios for network separation, DDoS mitigation and ransomeware prevention.

CCS CONCEPTS

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

KEYWORDS

Security management; Security policy; NSF;

ACM Reference Format:
Eunsoo Kim, Kuyju Kim, Seungjin Lee, Jaehoon (Paul) Jeong, and Hyoungshick Kim. 2018. A Framework for Managing User-defined Security Policies to Support Network Security Functions. In *Proceedings of The 12th International Conference on Ubiquitous Information Management and Communication (IMCOM '18),* Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3164541.3164569

IMCOM '18, January 2018, Langkawi, Malaysia

ACM ISBN 978-1-4503-6385-3/18/01…$15.00
https://doi.org/10.1145/3164541.3164569

1 INTRODUCTION

Defining security policies for a network system is a difficult and complicated task that often requires deep knowledge of a particular vendor's protocols and network commands. This has been the biggest challenge for system administrators who are responsible for managing network systems. This problem is further amplified for network environments with Network Security Functions (NSFs) provided by multiple vendors with proprietary interfaces [11]. In many situations, NSFs can be used to achieve security goals such as integrity, confidentiality and availability to protect a network system by detecting malicious traffic and/or reducing the impact of cyber attacks on the network system [7]. In practice, however, it is very cumbersome to manage and enforce various security policies and configurations on NSFs due to various business requirements and the complexity of security practices for satisfying those requirements. The detailed challenging issues are as follows:

First, it is not easy to consider new security requirements and the corresponding security rules in a timely way in response to adaptive and sophisticated attacks which are evolved over time. Second, the cost of managing security policies is likely to increase because multiple vendors' network devices and security solutions can typically be used in a mixed way for a network system. In general, each vendor uses its own proprietary interface, which makes system administrators harder to set up vendor-specific rules and configurations. Third, large companies generally require very complicated security requirements for various users and devices, which may produce complicated security rules.

To address those issues, several architectures were introduced based on Software-Defined Networking (SDN) and Network Functions Virtualization (NFV). For example, the Internet engineering Task Force (IETF) Interface to Network Security Functions (I2NSF) working group aims to define and implement standard interfaces for controlling and managing NSFs. This standardization defines an architecture and interfaces for network security services using SDN and NFV. However, it is still unclear how (relatively complicated) high-level security policies defined by users can be mapped into low-level security policies for network devices, and then the low-level security policies can be configured on those devices.

In this paper, we propose a framework to effectively translate high-level security policies for users into low-level security policies for network devices. To show the feasibility of the proposed

framework, we also implement a prototype to support RESTCONF protocol (RFC 8040 [2]), which is an HTTP-based protocol, providing a programmatic interface to access data defined in YANG [3]. Note that YANG is a data modelling language for network configuration. We also discuss how the proposed architecture can be applied in real-world situations for network separation, DDoS attack mitigation and ransomware prevention.

The rest of the paper is organized as follows. Section 2 provides some background knowledge for understanding the proposed architecture. Section 3 presents the proposed framework. Section 4 describes user-defined policies in the proposed framework. Section 5 explains the details of our prototype to implement the proposed framework. Section 6 introduces three important real-world scenarios where the proposed framework can effectively be applied to. Section 7 discusses the limitations of our work, and the related work is summarized in Section 8. Finally, our conclusion and future work are given in Section 9.

2 BACKGROUND

In this section, we provide a brief overview of NFV, NSF and RESTCONF protocol.

2.1 Network Function Virtualization (NFV)

NFV allows network functions to be performed on virtual machines in a cloud infrastructure rather than dedicated physical devices. The popularity of NFV is growing for network operators who want to deploy new network services in a flexible manner. NFV is a key enabling technology for providing customized network services in a flexible and scalable manner by providing network functions through software implementation rather than hardware resources. In the virtual environment, the functionality of hardware components can be easily emulated, and multiple virtual functions can share available resources and run simultaneously on a virtualized infrastructure.

2.2 Network Security Function (NSF)

NSF is a security function used to ensure integrity, confidentiality, or availability of network communication systems by detecting suspicious or malicious network activities and blocking them. Because NSFs should be deployed in increasingly diverse environments, various business and security requirements should also be considered. Users could consume network security services enforced by NSFs hosted by one or more providers, which may be their own network systems and services. Similarly, service providers may offer their customers network security services that are enforced by multiple security products and functions from different vendors, or open source projects. NSFs can be provided by both physical and virtualized infrastructures. Without standard interfaces to monitor and control the behaviors of NSFs, it is not easy for security service providers to automate various security services with various security functions from multiple vendors.

2.3 OpenDaylight & Open vSwitch

OpenDaylight first started as a collaborative project hosted by the Linux Foundation. The goal of the project is to promote SDN and NFV. The OpenDaylight software is written in Java programming language [15]. It is a modular open platform for customizing and automating networks of any size and scale. The OpenDaylight project arose out of the SDN movement, with a clear focus on network programmability. It was designed from the outset as a foundation for commercial solutions that address a variety of use cases in existing network environments. The OpenDaylight software architecture defines application creation and interaction patterns along with underlying application services (e.g., message routing, formatting and data storage) and has ultimately lead to new development tools for an environment enabling open source collaborations [16].

Open vSwitch is an open-source implementation of a distributed virtual multilayer switch [19]. The main purpose of Open vSwitch is to provide a switching stack for hardware virtualization environments, while supporting multiple protocols. Open vSwitch is designed to enable network automation through programmatic extensions, while supporting standard management interfaces and protocols such sFlow, Command Line Interface (CLI) and NetFlow. In addition, Open vSwitch is designed to support switching functionality across multiple physical servers by enabling the creation of cross server switches in a way that abstracts the underlaying server architecture, similar to the VMware vNetwork distributed vswitch or Cisco Nexus. Open vSwitch can operate as both SDN running within a virtual machine, and the control stack for dedicated switching hardware. As a result it has been ported to multiple virtualization platforms as switching chipsets [20]. Open vSwitch has also been implemented into various cloud computing software platforms and virtualization management systems, including OpenStack and OpenNebula.

2.4 RESTCONF protocol

RESTCONF protocol is an HTTP-based protocol that provides a programmatic interface for accessing data defined in YANG data model language [3]. Similar to NETCONF [5], RESTCONF supports GET, PUT, POST, DELETE operations and every request and response data can be in XML or JSON format. XML has a structure according to both YANG by XML-YANG, and JSON by JSON-YANG. RESTCONF provides a set of Create, Read, Update and Delete (CRUD) operations on conceptual datastores containing YANG-defined data, which are compatible with NETCONF. A RESTCONF server enumerates YANG modules that can be supported under "/restconf/modules" in the top-level API resource type, using a structure based on the YANG module capability URI format.

2.5 Network Access Control (NAC)

There are many different network tenants within an organization. The tenants can be various departments such as Human resources, Finance, Legal and etc. These network tenants or departments may wish to manage their own security policies due to regulatory, compliance or business reasons.

NAC is a solution for controlling and controlling access in the network. NAC has many functions such as authentication and verification. When a client, such as a new device, requests access to a network, NAC offers authentication and verification methods to identify it. Furthermore, Role Based Access Control (RBAC) is possible using NAC in order to control access to the network based on each client's role.

There are several popular opensource tools of NAC, such as packetfence, opennas and freenac [17]. If a network is divided into one with the Internet access and the other without the Internet access, an system administrator may give wrong permission, thereby causing confusion in the network. This kind of problems can be amended by using NAC equipment in the form of NSFs.

When NAC is first installed into a network, it collects IP and NIC (MAC) addresses of every device that is connected within this network, and maps each IP address with the corresponding NIC address. If a device with an IP or NIC address, which is not registered within the mapping table, tries to connect to the network, the NAC can detect this. With this method, a detected device can be blocked so that the device cannot attempt DNS and DHCP access.

3 PROPOSED FRAMEWORK

In this section, we propose a framework for translating user-defined high-level policies into low-level policies for NSFs in a flexible manner. The modeling of a user-defined policy is based on the information model for the client interface (called Consumer-facing interface) [21] [14].

A client interface is used to enable different users of a given NSF system to define, manage and monitor security policies for specific flows within an administrative domain. The location and implementation of these policies are irrelevant to the client. One example of the client interface could be an enterprise network administrators and management systems that need to request their provider network to enforce specific policies for NSFs for particular flows. Another example is an IoT management system sending requests to the underlay network to block flows that match a set of specific conditions. The main idea behind this model is to consider the corporate network as organizations. Each organization has its own abstract levels (subjects, actions, events, conditions and objects), and the hierarchy for the organizations. These abstract levels are an important step for the model. Our main interest here is to define the security policies associated with each NSF and policy type.

Our proposed framework plays a role as a middleware between the user client side and the network security function instances, that is, from the generation of a high-level policy using the client interface to the translation and parsing in order to perform the successful deployment and enforcement of low-level policy to the NSFs.

Figure 1 shows the general architecture of our proposed framework consisting of three main components in order to generate high-level user policies, parse them, and translate them into low-level policies. We name the three components to be policy generator, policy converter, and policy parser, and then describe their roles in the subsections below.

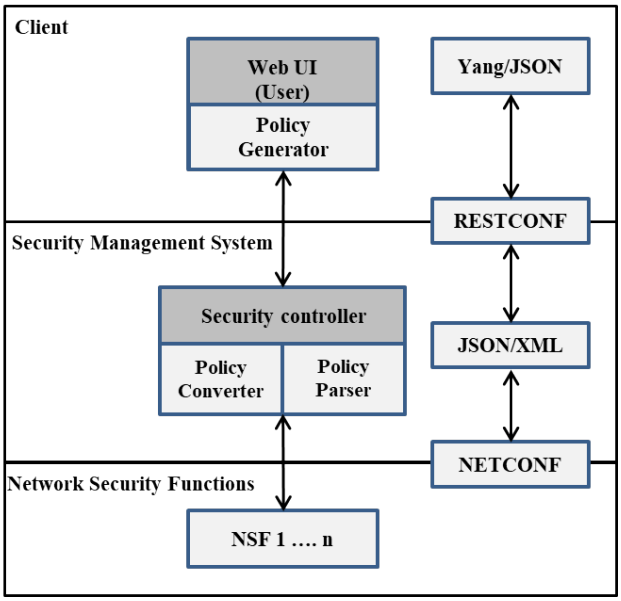

Figure 1: Proposed framework for security policy management.

3.1 Policy generator

The policy generator will generate high-level user-defined policy according to the needs of a system administrator regardless of the vendors providing different types of security function capabilities. Although it is difficult to implement one-to-one mapping of a high-level policy to low-level policy, it is possible to restrict and simplify the expressions used by providing a simple user friendly interface.

3.2 Policy converter

If a user, typically a system administrator defines a security rule using the user interface, the defined rule needs to have a structured form in order to be forwarded to the final stage of the policy translation. The policy converter is responsible for translating a general string of a policy into intermediate language code such as XML script.

3.3 Policy parser

The policy parser is responsible for importing the user-defined policy generated by the policy generator. The imported policy is in an intermediate form (i.e., a XML document) converted by the policy converter as described in the previous stage. After completing the import process successfully, the policy parser then creates security rules for a network security function by matching each data within the XML document.

The delivery of the high-level policy defined by a system administrator is done through RESTCONF protocol. When JSON/YANG format policy is generated at the policy generation step, an HTTP request will be sent to a RESTCONF server, and then the policy converter and parser will start converting and parsing the policy into an appropriate format, respectively.

4 USER-DEFINED POLICY

In this section, we discuss the process of generating user-defined policies in the proposed framework. Before discussing this process in detail, we need to define who the user is. A user is a role that is assigned to an NSF component that contains functions to provide information to other components. Our interface can be an example of providing rules to other network security components. The user-defined policy needs users who take the role of either an operator, end-user or application developers to represent multiple types of our event-condition-action based rules for network traffic selection and configuration of security policies.

In order to effectively generate and enforce user-defined policies, we take a policy-specification-based approach. A policy specification language is used to formalize the intent of the user into a form that can be read and interpreted by machines. We consider XACML to express access control policies as XML. XACML is an XML specification for expressing policies for information access over the Internet, and the language provides XML with a sophisticated access control mechanism that enables the initiator not only to securely browse SML documents but also to securely update each document element.

The client interface acts as the brain in our proposed framework which is responsible for validating to a security policy, while maintaining the consistency and accuracy of the security policy. Furthermore, it is responsible for managing the priorities to avoid permission conflicts between the policies, and to safely enforce those high-level user-defined policies in a user-friendly manner. Appendix A shows the web-based interface example to generate a user-defined policy for NAC.

The high-level policies defined by users should be delivered to a security controller in the proposed framework. The communication module and the protocol between the client interface and the security controller is based on the RESTCONF, and the server and client are developed based on the python package called JETCONF. JETCONF is an implementation of the RESTCONF protocol written in Python. It provides HTTP/2 over TLS, certificate-based authentication of clients, JSON data encoding, and per-user candidate datastores with transactions. We particularly modified the client and server codes in order to send HTTP requests through PHP and receive the corresponding responses in XML. Because the JETCONF package does not provide JSON-to-XML translation, we developed an XML parser to translate JSON formatted data strings into an XML format. The overall process is as follows:

1. Generate a user-defined policy from a web-based client interface. (This step involves creation of a JSON formatted file that will be stored in a Web server.)

2. Send an HTTP request to a RESTCONF server, which will then read in the JSON files based on the pre-defined YANG structure.

3. Perform necessary actions, either configuration or operation, based on the type of YANG files.

4. Parse the JSON/YANG file into XML document and enforce the new policy to the NSFs, and display in a well-formatted XML.

5 IMPLEMENTATION

In this section, we describe the implementation of the necessary components that consists the proposed framework, and show its feasibility by showing how it can be applied to NAC.

```
module: client-interface-nac
    +--rw client-interface-nac
       +--rw threat-prevention
       |  +--rw threat-feed* [threat-feed-id]
       |     +--rw threat-feed-id    uint16
       +--rw policy-endpoint-groups
       |  +--rw user-group* [user-group-id]
       |     +--rw user-group-id    uint16
       +--rw security-policy-instance
          +--rw policy-rule* [policy-rule-id]
          |  +--rw policy-rule-id      uint16
          |  +--rw name?               string
          |  +--rw date?               yang:date-and-time
          |  +--rw source?             -> .../threat-feed-id
          |  +--rw destination?        -> .../user-group-id
          |  +--rw event?              -> .../event-id
          |  +--rw condition?          -> .../condition-id
          |  +--rw action?             -> .../action-id
          |  +--rw exception?          boolean
          |  +--rw exception-detail?   string
          +--rw action* [action-id]
          |  +--rw action-id           string
          |  +--rw name?               string
          |  +--rw date?               yang:date-and-time
          |  +--rw primary-action?     string
          |  +--rw secondary-action?   string
          +--rw precedence* [precedence-id]
          |  +--rw precedence-id    string
          |  +--rw rule-exist?      boolean
          +--rw event* [event-id]
          |  +--rw event-id           string
          |  +--rw security-event?    string
          |  +--rw threat-map?        string
          |  +--rw enable?            boolean
          +--rw condition* [condition-id]
          |  +--rw condition-id       string
          |  +--rw service* [service-id]
          |  |  +--rw service-id    uint16
          |  |  +--rw name?         string
          |  +--rw nac-source* [source-id]
          |  |  +--rw source-id        uint16
          |  |  +--rw source-id-ip?    inet:ipv4-prefix
          |  |  +--rw source-id-mac?   yang:mac-address
          |  +--rw nac-destination* [destination-id]
          |     +--rw destination-id        uint16
          |     +--rw destination-id-ip?    inet:ipv4-prefix
          |     +--rw destination-id-mac?   yang:mac-address
          +--rw policy-instance* [policy-instance-id]
             +--rw policy-instance-id    string
             +--rw name?                 string
             +--rw date?                 yang:date-and-time
             +--rw rules?                -> .../policy-rule-id
```

Figure 2: YANG data tree for NAC.

5.1 Client interface & RESTCONF client

Since our proposed framework uses a web-based user interface as a client interface to generate a user defined high-level policy, we first created a web server based on an Apache HTTP server so that the web interface can send HTTP requests. We also created a virtual network topology using Mininet, and used OpenDaylight to control the network traffic. We used RESTCONF protocol in order to deliver a generated high-level security policy to a security controller.

In our case, the security controller not only monitors and manages network flows but also acts as a RESTCONF server. For successful communication between the web client and the security controller through RESTCONF, we created a web client in PHP that generates a security policy and stores it as a JSON file. It then

makes URI calls based on HTTP request and sends JSON format policy file to the RESTCONF server.

As soon as the server received the HTTP request, it starts converting the JSON format data and translates it into appropriate rule format. In order to setup the RESTCONF client and server environment, we leveraged an open source python package called JETCONF. JETCONF is an implementation of the RESTCONF protocol written in Python. It provides, HTTP/2 over TLS, certificate-based authentication of clients, JSON data encoding, and Per-user candidate datastores with transactions and support for NACM. The client interface generates a JSON file according to the YANG structure.

Figure 2 shows an example YANG data tree for generating a high-level policy. The security policy is an Event-Condition-Action (ECA) based policy, which is made up of three Boolean clauses: an event clause, a condition clause, and an action clause. A Boolean clause is a logical statement that evaluates to either true or false. In our NAC example, an unidentified device connected to a network is the event, MAC address matching is the condition, and either deny or allow is the action. This ECA-based policy is very general and easily extensible, and can avoid potential constraints that could limit the implementation of security capabilities.

```
<client-interface-nac:security-policy-instance>
  <precedence>
    <precedence-id>0</precedence-id>
    <rule-exist>false</rule-exist>
  </precedence>
  <event>
    <event-id>1</event-id>
    <security-event>Unidentified-connection</security-event>
    <mac-match>true</mac-match>
  </event>
  <condition>
    <condition-id>1</condition-id>
    <service>
      <name>NAC-example</name>
      <service-id>1</service-id>
    </service>
    <nac-source>
      <source-id>1</source-id>
      <source-id-mac>BA:AA:AA:AA:AA:AA</source-id-mac>
    </nac-source>
  </condition>
  <action>
    <action-id>1</action-id>
    <name>network separation</name>
    <primary-action>drop</primary-action>
    <secondary-action>log</secondary-action>
  </action>
  <policy-rule>
    <policy-rule-id>1</policy-rule-id>
    <name>nac-policy-example</name>
    <date>2017-09-15T18:00:00Z</date>
    <event>1</event>
    <condition>1</condition>
    <action>1</action>
    <source>BA:BA:BA:BA:BA:BA</source>
    <destination>192.168.1.0/24</destination>
  </policy-rule>
  <policy-instance>
    <policy-instance-id>1</policy-instance-id>
    <name>nac-policy-example</name>
    <date>2017-09-15T18:00:00Z</date>
    <rules>1</rules>
  </policy-instance>
</client-interface-nac:security-policy-instance>
```

Figure 3: Example of XML documents for NAC.

5.2 RESTCONF server

We also implemented the RESTCONF server using JETCONF. The RESTCONF server needs a start-up configuration with the directory path of YANG modules and a JSON file acts as a datastore storing a user-defined high-level policy. When the RESTCONF server starts, it first conducts YANG validation of the YANG modules in the directory path. Once the validation process and authentication of client are completed successfully, the RESTCONF server starts handling client requests.

For GET methods to get a user-defined high-level policy, the RESTCONF server returns the corresponding JSON formatted policy data stored in the datastore. For other methods, such as PUT or UPDATE, the RESTCONF server first validates and confirms whether the fields of request contain the valid data type according to the requested YANG module. If the validation of client request is completed successfully, the requested inputs are reflected to the policy data stored in datastore.

Finally, the JSON format policy data (which is based on YANG structure shown in Figure 2) is translated into a low-level policy rule, and then the rule, which is comprehensible to NSF, is passed to an NSF. The JSON formatted file is converted into the XML format document, which is then parsed into an appropriate policy rule by the policy parser component in the server. An example of XML documents for NAC is shown in Figure 3.

5.3 Implementation example: NAC

We describe how we set up an NAC environment using the implementation above. The security controller, which acts as a RESTCONF server in our case. In order to separate the internal server from the public network, NAC has a MAC table to check the IP and MAC addresses of the sources for the incoming packets. If a MAC address is not registered, it is initially blocked and needs the approval from NAC. When a new device is registered in the public network, the security controller compares the MAC address of the new device with those registered in the MAC table of the internal network and vice versa.

If the MAC address is not found within the MAC table, then it is considered to be an unauthorized access and sends alarm messages to the system administrator. The system administrator generates a user-defined high-level policy to the security controller, which then converts the policy into a low-level policy rule, such as an Openflow rule. As soon as the rule is applied, the system administrator can keep track of the unauthorized access by triggered alarms, and block it permanently if necessary.

Figure 4 shows the NAC environment set up and shows how the security policy generated by the system administrator can be applied to it. As it can be seen from the figure, both public and internal networks have NAC as NSFs, and they can be controlled by a security controller. Assuming that the host 4 with IP address 172.16.2.1, which has BA (AA:AA:AA: BB:BB:BB) as its MAC address tries to connect to the public network, the security controller compares it with the MAC addresses in the MAC table stored in the NAC of the internal network.

If the MAC address of the device is found out to be a device registered as an internal network device only, then an alarm message is sent to the system administrator. The system administrator then generates a high-level security policy and sends it to the security controller. The security controller then converts the high-level policy into a low-level policy, similar to that of a firewall policy, which

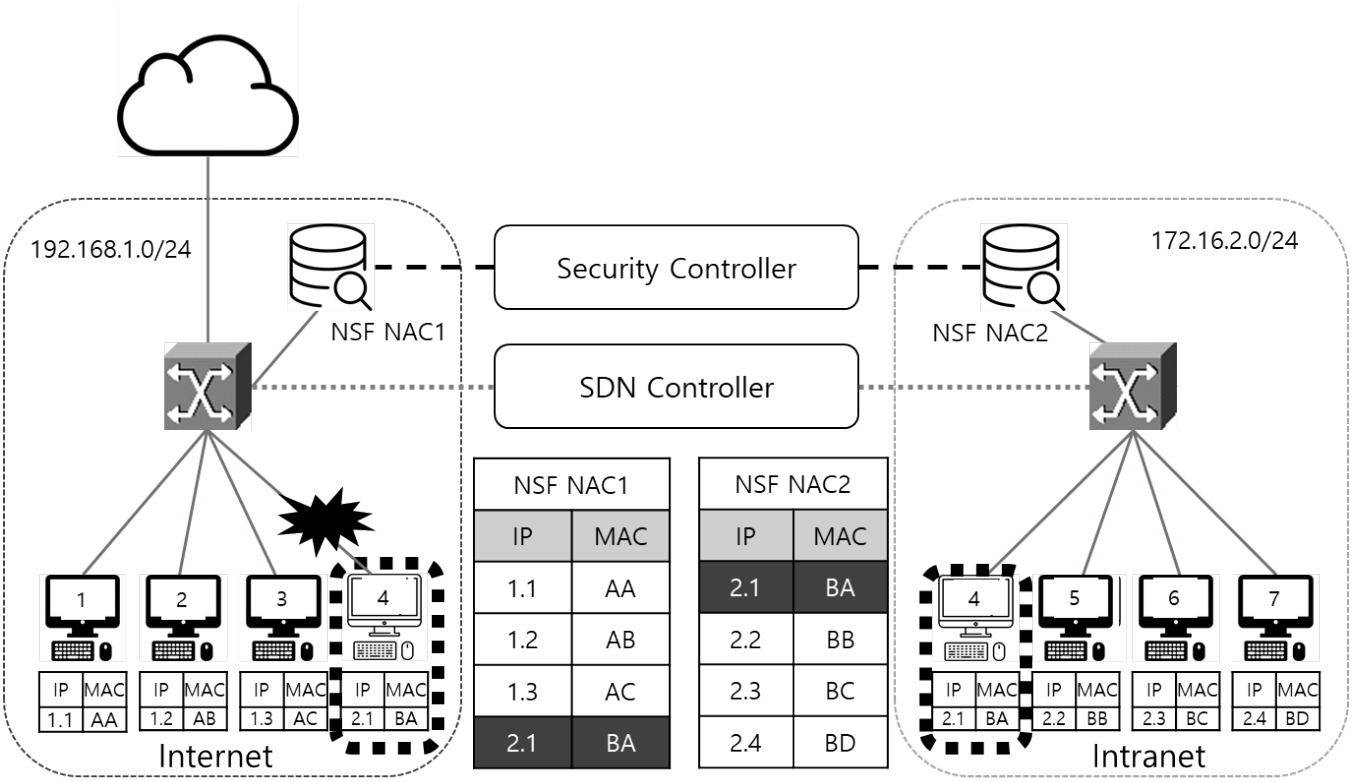

Figure 4: NAC environment.

can be enforced to the Open vSwitch through an SDN controller [22]. The Open vSwitch finally blocks the network traffic.

The XML structure follows the YANG tree, and therefore follows the same namespace. <precedence> is to check whether the same rule exists or not. <event>, <condition>, and <action> represents the security event and triggers the evaluation of the policy rule, the condition for the action to be applied or not, and the action which will be applied eventually if the <event> happens while the <condition> is true. <rule> is the rule of the policy, and refers to <event>, <condition> and <action> with <source> and <destination> information of the connected device to create a single rule. Lastly, <policy instance> contains the policy information and the rules. An example translated security policy rule is shown in Table 1.

6 CASE STUDIES

Our framework and user interface is designed to react to possible security attacks. This section shows the procedure of the security attack mitigation scenarios, such as DDoS mitigation. We also show how our proposed method can be implemented for separating networks.

6.1 DDoS mitigation

DDoS attacks have been a big challenge for many organizations as the impact of DDoS attacks is huge. Detecting DDoS attacks can be achieved using rules generated by our framework in accordance with our policy YANG model.

DDoS attacks can be detected using the rules generated by our framework. One possible mitigation for the DDoS attack is by setting a threshold for the network traffic, however, there are other methods such as signature-based detection [1]. There are several tools that are used for performing DDoS attack, such as TFN, TFN2K, Trin00 and Stacheldraht. Those DDoS attacks are performed using the tools mentioned above to generate distinguishable packets, so they can be easily tracked using the rules generated by our framework.

Those DDoS attacks (e.g., HTTP GET flooding, SYN flooding and NTP amplification attack) can be easily mitigated by a firewall if the service ports are known. However, this is not usually the case, so system administrators should set appropriate thresholds and monitor the network traffic to prevent such a denial of service.

6.2 Ransomware prevention

Ransomware is a software that encrypts some files and asking for money or Bitcoin as the cost of encryption key. The ransomware attacks have existed since the early 2000s, but lately, many new ransomware attacks that cause many problems socially are being produced and distributed newly. The mechanisms to distribute the ransomware use Drive by download, E-mail or P2P usually [8].

Ransomware are first distributed through drive-by-download, spam e-mail or P2P. When a user executes a malware, it connects

Table 1: Example of Low-level Security Policies at an Open vSwitch

MAC src	MAC dst	IP Src	IP Dst	TCP sport	TCP dport	Action
BA:BA:BA:BA:BA:BA	*	*	*	*	*	Drop
AA:AA:AA:AA:AA:AA	FF:FF:FF:FF:FF:FF	*	*	*	*	Drop
*	*	192.168.1.5	192.168.1.100	*	22	Accept
*	*	192.168.1.0/24	*	*	22,3389	Drop
*	*	*	192.168.1.100	*	389	Accept
*	*	*	192.168.1.100	*	135-139,445	Accept
*	*	*	*	*	135-139,445	Drop
*	*	192.168.1.0/24	*	*	*	Accept

to a distribution website and downloads a ransomware. Once the download is complete, the ransomware is executed successfully, and then it connects to a C&C server to encrypt the user files. In general, most ransomware operate through the C&C servers.

A system administrator can use a blacklist to block C&C servers. The blacklist contains the list of C&C server URLs. This method is much more effective when compared to the method using the URLs of a ransomware distribution website. This is because an attacker may use many different distribution websites, whereas the number of C&C servers is small. Moreover, there are various kinds of websites that offer ransomware related information, such as https://ransom-waretracker.abuse.ch/blacklist/, so a blacklist based on the URL information given in this website may significantly help mitigating ransomware attacks [4]. A high-level-policy may leverage this blacklist to create a rule to shield against such attacks, and a security controller may translate and enforce the low-level-policy for the NSFs.

7 LIMITATIONS

Although we provide a user-friendly web-based client interface to create simple high-level policies, it is not truly human language based policy translator. A high-level policy should be generated by involving natural language processing, however, this is out of scope in this paper. Natural language processing is concerned with the interactions between computers and human languages, and it is concerned with programming computers to fruitfully process large natural languages. The challenges in natural language processing frequently involve understanding, generation, connecting, machine perception, or the combination of the human language, which are not currently considered in this paper.

There are a numerous number of different vendors so it is not easy to develop a system to support all vendors. However, it is possible to develop a separate registration interface to register capabilities of each vendor's NSF to flexibly adjust the system when it is necessary to convert from a vendor to another.

8 RELATED WORK

Liu et al. [13] proposed a tool called *EASYACL* that generates the Access Control Lists (ACLs) rules automatically from natural language descriptions. The ACL is a list of permissions of access to a specific network asset. Thus, a mistaken configuration of ACL causes the network performance degradation as well as the network security vulnerabilities. The configurations of network properties, especially ACL rules, are considered as a difficult task disturbing many end users, because of the many options contained in configuration commands and the platform dependency of rule syntax. The *EASYACL* translates the end users' natural language descriptions to ACL configuration commands with a rule-based natural language processing method, so that the end users easily configure ACL rules and the system administrators troubleshoot the ACL configuration errors. Our framework is motivated by the *EASYACL*, but we considered the network configuration rules in the SDN/NFV environment.

Similarly, Hassan et al. [10] suggested a framework extending the Organization Based Access Control (OrBAC) model for automating the process of translation of high-level security policy into a low-level security policy. The framework however did not consider the SDN/NFV environment. We developed a YANG [3] data model based on SUPA [9] used for the high-level policies in the I2NSF (Interface to Network Security Functions) architecture, and in our work, we suggested a framework for translating user-defined security policies into low-level policies in NSF by extending Hassan et al.'s framework.

The work for standardization of the architecture to control NSF instances has been actively performed. I2NSF [12] and ETSI NFV [6] have been defining a framework for controlling NSF instances in a virtual environment using NFV so as to allow end users to define their own security policies. Furthermore, Oh et al. [18] suggested an architecture providing policy specification services for I2NSF users to control and manage NSF instances through the high-level policies and described how the proposed architecture is used for VoIP-VoLTE services. However, the translation of the user-defined high-level policies has difficulties inherently such as consistency or verification, and in the above works, they do not specify how to implement the translation of user-defined high-level policies into low-level policies and to map the policies to NSF instances.

9 CONCLUSION

In this paper, we presented a framework for managing user-defined security policies for NSFs based on the network interfaces that are currently being discussed in the IETF working group for interfaces to network security functions (i.e., I2NSF). Our ultimate goal is to translate high-level security policies into low-level policies in order to provide user-friendly interfaces and tools for system administrators on network systems. To show the feasibility of the proposed framework, we implemented a prototype based on the

RESTCONF protocol and demonstrated that the proposed framework can be applied to real-world scenarios for network separation, DDoS mitigation, and ransomeware prevention.

As part of future work, we plan to develop techniques to parse natural language-like high-level security policies and process them for the proposed framework. We also intend to conduct real-world experiments through the deployment of the proposed framework at a university network, and analyze its performance for configuring security rules to mitigate popular network attacks in a real-world setting.

ACKNOWLEDGMENTS

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2016-0-00078, Cloud based Security Intelligence Technology Development for the Customized Security Service Provisioning) and the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2017-2015-0-00403) supervised by the IITP(Institute for Information & communications Technology Promotion).

REFERENCES

[1] Narmeen Zakaria Bawany, Jawwad A Shamsi, and Khaled Salah. 2017. DDoS Attack Detection and Mitigation Using SDN: Methods, Practices, and Solutions. *Arabian Journal for Science and Engineering* 42, 2 (2017), 425–441.

[2] Andy Bierman, Martin Bjorklund, and Kent Watsen. 2017. *RESTCONF Protocol.* RFC 8040. https://tools.ietf.org/html/rfc8040.txt

[3] M Bjorklund. 2010. *YANG-A data modeling language for the Network Configuration Protocol (NETCONF).* RFC 6020. https://tools.ietf.org/html/rfc6020.txt

[4] Krzysztof Cabaj and Wojciech Mazurczyk. 2016. Using software-defined networking for ransomware mitigation: the case of cryptowall. *IEEE Network* 30, 6 (2016), 14–20.

[5] Rob Enns, Martin Bjorklund, and Juergen Schoenwaelder. 2011. *Network Configuration Protocol (NETCONF).* RFC 6241. https://tools.ietf.org/html/rfc6241.txt

[6] ETSI. [n. d.]. Industry Specification Group for NFV. http://www.etsi.org/technologies-clusters/technologies/689-network-functions-virtualisation. ([n. d.]). Online; accessed 2017.

[7] Mahdi Daghmehchi Firoozjaei, Jaehoon Paul Jeong, Hoon Ko, and Hyoungshick Kim. 2017. Security challenges with network functions virtualization. *Future Generation Computer Systems* 67, 2 (2017), 315–324.

[8] James B Fraley and James Cannady. 2016. Enhanced detection of advanced malicious software. In *Proceeding of the 7th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference.*

[9] Joel M Halpern and John Strassner. 2017. *Generic Policy Data Model for Simplified Use of Policy Abstractions (SUPA).* IETF Internet-Draft draft-ietf-supa-generic-policy-data-model-04. https://www.ietf.org/id/draft-ietf-supa-generic-policy-data-model-04.txt

[10] Ahmed Hassan and Waleed Bahgat. 2009. A framework for translating a high level security policy into low level security mechanisms. In *Proceeding of the 7th IEEE/ACS International Conference on Computer Systems and Applications.*

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

[12] IETF. [n. d.]. Interface to Network Security Functions (I2NSF) Working Group. http://datatracker.ietf.org/wg/i2nsf/charter/. ([n. d.]). Online; accessed 2017.

[13] Xiao Liu, Brett Holden, and Dinghao Wu. 2017. Automated Synthesis of Access Control Lists. In *Proceeding of the 3rd IEEE International Conference on Software Security and Assurance.*

[14] Edward Lopez, Diego Lopez, Linda Dunbar, John Strassner, and Rakesh Kumar. 2017. *Framework for Interface to Network Security Functions.* IETF Internet-Draft draft-ietf-i2nsf-framework-07. https://www.ietf.org/id/draft-ietf-i2nsf-framework-07.txt

[15] A Mayoral, Ricard Vilalta, Raul Muñoz, Ramon Casellas, Ricardo Martínez, and J Vilchez. 2014. Integrated IT and network orchestration using OpenStack, OpenDaylight and active stateful PCE for intra and inter data center connectivity. In *Proceedings of the 40th IEEE European Conference on Optical Communication.*

[16] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. 2014. Opendaylight: Towards a model-driven SDN controller architecture. In *Proceeding of the 15th IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks.*

[17] Henry Nunoo-Mensah, Emmanuel Kofi Akowuah, and Kwame Osei Boateng. 2014. A Review of Opensource Network Access Control (NAC) Tools for Enterprise Educational Networks. *International Journal of Computer Applications* 106, 6 (2014), 28–33.

[18] Sanghak Oh, Eunsoo Kim, Jaehoon Paul Jeong, Hoon Ko, and Hyoungshick Kim. 2017. A flexible architecture for orchestrating network security functions to support high-level security policies. In *Proceeding of the 11th ACM International Conference on Ubiquitous Information Management and Communication.*

[19] Ben Pfaff, Justin Pettit, Keith Amidon, Martin Casado, Teemu Koponen, and Scott Shenker. 2009. Extending Networking into the Virtualization Layer.. In *Proceeding of the 8th ACM Workshop on Hot Topics in Networks.*

[20] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martín Casado. 2015. The Design and Implementation of Open vSwitch.. In *Proceeding of the 12th USENIX Symposium on Networked Systems Design and Implementation.*

[21] Kumar Rakesh, Lohiya Anil, Qi Dave, Bitar Nabil, Palislamovic Senad, and Xia Liang. 2017. *Client Interface for Security Controller : A Framework for Security Policy Requirements.* IETF Internet-Draft draft-kumar-i2nsf-client-facing-interface-req-03. https://www.ietf.org/id/draft-ietf-i2nsf-client-facing-interface-req-03.txt

[22] Seungwon Shin, Lei Xu, Sungmin Hong, and Guofei Gu. 2016. Enhancing Network Security through Software Defined Networking (SDN). In *Proceeding of the 25th IEEE International Conference on Computer Communication and Networks.*

A WEB INTERFACE FOR HIGH-LEVEL SECURITY POLICIES

Figure 5 shows an example of web page for generating high-level security policies in the proposed system. The simple interface is designed using PHP.

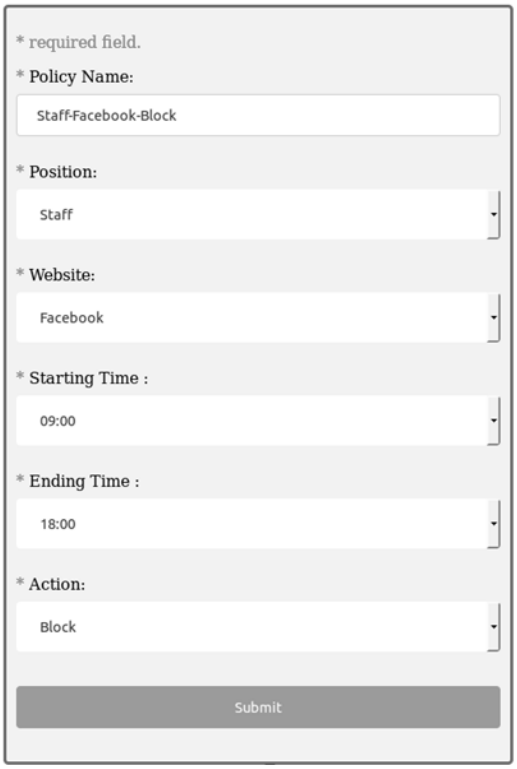

Figure 5: Web UI example.