An Efficient Defense Method For Compromised Switch and Middlebox-Bypass Attacks In Service Function Chaining

Nguyen Canh Thang and Minho Park

Abstract: **Service function chaining (SFC) has a special and powerful ability to define an ordered list of required network services as a virtual chain and makes a network more flexible and manageable. However, there are many vulnerabilities to SFC, such as compromised switches and middlebox-bypass attacks, which can damage the operation and security of the network. In this study, we propose a mechanism that not only detects both middlebox-bypass attacks and compromised switch attacks in multiple service function chains scenario but also prevents such attacks and protects the network. The proposed mechanism uses both probe-based and statistics-based methods to handle the probe packets and collect statistics from middleboxes for detecting any attacks in SFC. After detection, the mechanism changes the network topology to eliminate the compromised switches, while meeting the initial requirements of the service chains. By combining probe-based and statistics-based methods, our proposal overcomes the disadvantages of other proposed solutions and brings about a robust protection to SFC. As the experimental results indicate, the proposed mechanism is an effective and relevant approach for detecting and preventing compromised switches and middlebox-bypass attacks in SFC.**

Index Terms: **Compromised switches, middlebox-bypass attack, service function chaining (SFC).**

I. INTRODUCTION

In recent years, service function chaining (SFC) has emerged with the robust development of software defined networking (SDN) and network function virtualization (NFV). SFC defines ordered virtual chains of service functions (e.g., firewalls, load balancing, and network address translation) and steers the network traffic through them, leading to many different benefits of a virtualized software-defined infrastructure [1]–[3]. Service functions are provided by specialized network entities called middleboxes. One middlebox is commonly connected to a switch, and SFC connects switches to make a chain with the required services. Middleboxes are responsible for packet processing and packet forwarding to the attached switches in the service chain [4]–[8].

Manuscript received February 24, 2020; revised September 20, 2020; approved for publication by B. Byunghoon Kang, Division III Editor, October 18, 2020.

This work was partly supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2020R1F1A1076795), and by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.2018-0-00254, SDN security technology development).

N. C. Thang is with the Department of Information Communication Convergence Technology, Soongsil University, Seoul, South Korea, email: nct@soongsil.ac.kr.

M. Park is with the School of Electronic Engineering, Soongsil University, Seoul, South Korea, email: mhp@ssu.ac.kr

M. Park is the corresponding author.

Digital Object Identifier: 10.23919/JCN.2020.000028

A. Problem Statement

Despite the robust advantages of SFC, some drawbacks may destroy the operation of the entire network. A compromised switch is one of the most serious security vulnerabilities, affecting the traversing packets in SFC. The problems of SFC in terms of compromised switches can be listed as follows:

- The switch is one of the most important components in SFC, which has the task of forwarding packets between entities within the network. Attackers can exploit this component to create many types of attacks to destroy the network operation or steal the network information for other attacks in the future. Compromised switches can be controlled to drop, duplicate, incorrectly forward, or modify packets without notifying the controller. Packets and all network information can be sent to attackers, and any of these problems can breach the SFC policy.
- K. Bu *et al.*–FlowCloak [9] identified the *middlebox-bypass attack*, which occur when compromised switches forward packets to the next-hop middlebox in the SFC without sending them to the attached middlebox. This means that packets are not processed by all service functions inside the middleboxes, and thus the original goal of SFC is not met. Attackers, therefore, can bypass some important service functions, e.g., a firewall or an IDS, and conduct more attacks.

Based on the above problems, we can observe that compromised switches cause serious consequences to SFC. Besides, to the best of our knowledge, there have been no other studies on effectively solving both types of problems.

B. Our Proposal

To resolve the above issues related to compromised switches in the SFC scenario, we propose a mechanism that can simultaneously detect compromised switch and middlebox-bypass attacks in multiple SFC scenario. The proposed scheme uses a hybrid of probe-based and statistics-based methods, which overcomes the disadvantages of other solutions. A probe-based method uses probe packets to investigate the operations of the network components in SFC. Middleboxes are programmed to handle a random pre-assigned key in a probe packet and transmit it back to the attached switch. If the next-hop middlebox defines an incorrectly handled key verification, which means a middlebox-bypass attack has occurred, an alarm is triggered. Statistics-based methods help the controller find irregularities by monitoring all information on the packets that pass through the

1229-2370/19/$10.00 © 2020 KICS

Creative Commons Attribution-NonCommercial (CC BY-NC).
This is an Open Access article distributed under the terms of Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/3.0) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided that the original work is properly cited.

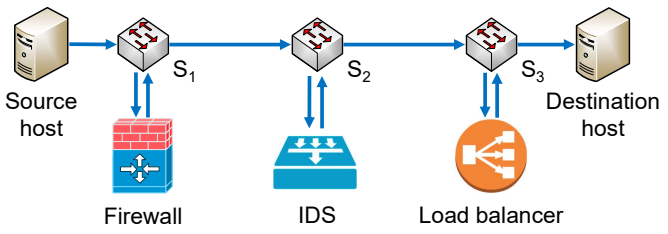


Fig. 1. Simple service function chain architecture example.

middlebox (e.g., the packet type, packet size, processing time, and the number of packets). By combining two methods, we can protect SFC from the compromised switch and middlebox-bypass attacks.

C. Contributions

In summary, the contributions of our research are described as follows:

- We describe the existing security vulnerabilities for a traversing packet in SFC, that is, compromised switch and middlebox-bypass attacks. We also discuss the disadvantages of the current solutions for such vulnerabilities.
- We propose a mechanism that can detect both middlebox-bypass attacks and other compromised switch attacks in multiple SFC scenario. The proposed mechanism combines probe-based and statistics-based methods to resolve the issues inherent to other current solutions.
- Additionally, we describe the experimental studies conducted and evaluate the performance and efficiency of the proposed mechanism.

The rest of this research is organized as follows. Section II provides brief introductions of SFC, compromised switches, middleboxes, and middlebox-bypass attacks and their vulnerabilities. Section III presents several previous studies related to our research. A detailed analysis of our system is described in Section IV, and the system designs are shown in Sections V and VI. Section VII focuses on the detailed experiments and evaluation of our scheme. Finally, some concluding remarks and areas of future research are given in Section VIII.

II. BACKGROUND KNOWLEDGE

A. Service Function Chaining

SFC is a network function that provides the ability to define a chain of service functions and dynamically steer the network traffic through various service function paths [1]. SFC enables carriers to benefit from a virtualized software-defined infrastructure, offering the flexible management of specific service/application traffic. It provides solutions for classifying network flows and enforcing policies along the flow routes according to the service requirements and considering the availability of the network [2]. A simple service function chain architecture is depicted in Fig. 1. Packets in this chain should follow this path: $SourceHost - S_1 - Firewall - S_1 - S_2 - IDS - S_2 - S_3 - LB - S_3 - DestinationHost$, to create a simple function chain.

SFC provides flexible network management by determining an ordered list of abstract service functions. Network policies typically require packets to go through a sequence of service functions. Achieving the benefits from such service functions and ensuring that the network traffic is directed through the desired sequence of service functions typically requires significant manual effort and operator expertise. Using SFC, the entire process becomes easier and more convenient. Service functions (e.g., network address translation, firewall, and load balancing) are traditionally deployed on dedicated hardware components, which are described in subsection II.C.

SFC makes use of a technology called software-defined networking (SDN). Similar to other software-defined architectures, SFC architectures are also vulnerable to security attacks. If a virtualized function is attacked, the complete service chain may collapse. Although there are already many solutions for protecting software components against attacks, such solutions must be revisited for application to SFC scenarios. Many potential security challenges remain as the future directions for SFC security, such as how to defend against distributed denial-of-service (DDoS) attacks and how to detect compromised components and mitigate their impact [3]. The next subsections describe two of the most serious problems to SFC security.

B. Compromised Switch Attacks

A compromised switch attack is a serious issue for an SDN in general and an SFC in particular. M. Antikainen [11] and P. Chi [12] defined several types of compromised switch attacks, such as packet dropping, packet duplication, packet manipulation, incorrect forwarding, eavesdropping, weight adjustments, man-in-the-middle attacks, state-spoofing, and control-channel hijacking. These attacks occur when compromised switches conduct certain attack actions in addition to forwarding the packets as the commands from the controllers. By controlling the compromised switches to apply one or all of these attacks, attackers can cause serious problems to the entire network.

For example, as shown in Fig. 2, when a compromised switch receives a packet, it can drop the packet (packet dropping attack), forward the packet multiple times to the next-hop switches (packet duplicating attack), modify the packet (packet manipulation attack), or even send that packet to an attacker or other switches outside of the chain (incorrect the chain (incorrect forwarding, eavesdropping, or man-in-the-middle attacks). Such attacks can ruin the operation of both the SFC and the network because one switch typically belongs to multiple SFC chains, which means that a compromised switch can connect to many other network entities. Furthermore, if there are other compromised switches in the network and if they join together, they can spoof information and breach all of the detection mechanisms.

C. Middlebox And Middlebox-bypass Attack

Middleboxes (such as firewalls, NATs, and load balancers) have grown into a major part of modern network infrastructure [5]. A middlebox can be defined as any intermediary network device applying functions other than standard functions of an IP forwarding between two end hosts [6]. Network deployments now use the deployment of middleboxes to handle changing applications, workloads, and policy requirements. Surveys

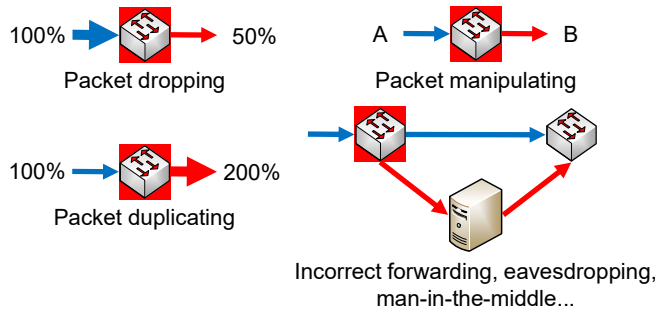


Fig. 2. Attack model using the compromised switch. The compromised switch can conduct duplication, packet manipulation, incorrect forwarding, eavesdropping, or man-in-the-middle attacks.

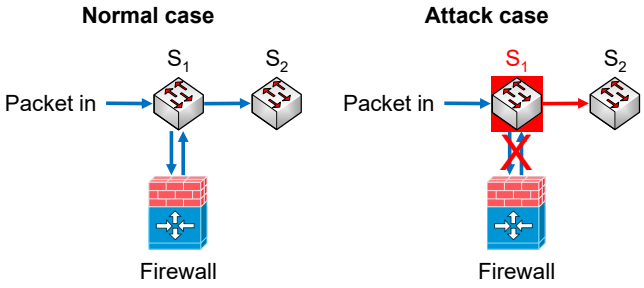


Fig. 3. Example of the middlebox-bypass attack.

show that middleboxes play a critical role in many network settings [7].

Middleboxes offer valuable benefits, such as improved security, improved performance, and policy compliance capabilities [8]. Middleboxes classify, filter, and shape the traffic, therefore interfering with the performance of the application, which can be categorized into two main types: security enhancements (enhancing the visibility of network traffic and enabling the enforcement of security policies) and performance enhancements (through traffic shaping, caching, and transparent proxying) [5]. However, middleboxes are also widely recognized to cause significant problems including a high cost, inflexibility, and complex management requirements, leading to high operational expenses and administrative issues. Because of the complex and specialized processing, as well as variations in management tools across devices and vendors, there are many vulnerabilities with middleboxes such as deployment, management, overloads, and failures [8].

A middlebox-bypass attack is a new vulnerability proposed by FlowCloak [9], which is another special type of incorrect forwarding attack through compromised switches. This attack occurs when a compromised switch does not forward packets to its attached middlebox. A simple example of a middlebox-bypass attack is shown in Fig. 3. Here, S_1 is connected to important service in the SFC chain, which in this case is a Firewall preventing harmful packets from coming inside the chain. In the normal case, the packets are forwarded from S_1 to Firewall and comeback to S_1. However, in the middlebox-bypass attack case, attackers try to bypass the Firewall by control S_1 to send packets to S_2 without forwarding to its attached middlebox (Firewall). This attack breaches the policy of the SFC and the middleboxes and requires the packets to pass through a certain chain of services within the middleboxes.

III. RELATED WORKS

A. Compromised Switch Attacks

Various countermeasures have been proposed to protect SFC and SDN from compromised switches [11] – [24]. There are two main categories of such a solution: probe-based and statistics-based method. With a probe-based method, probe packets are sent to each flow or specific switch, and the path and integrity of those packets are then checked. M. Antikainen [11] showed how to exploit compromised OpenFlow switches to attack SDN networks. P. Chi [12] defined some attack models through compromised switches and designed a probe-based detection mechanism to find these compromised devices. Y. Chiu [13] designed a method to detect disobedient forwarding in the flow table by installing new flow entries, sending probe packets, and checking the path of probe packets. Y. Ke [14] proposed SDNProbe, a lightweight SDN application that sends a provably minimized number of probe packets to pinpoint malfunctioning switches. C.-T. Kuo [15] addressed the SDN security challenge of compromised switches (single and collusion attack models) and proposed detection mechanisms by adding ghost switches into the network. N.Y. Vijayvergiya [16] proposed several algorithms to determine the compromised switch and the collaboration between them. Y. Zhang [17] presents a novel troubleshooting tool that can discover the forwarding path taken by any packets. However, the probe-based method can be disabled if compromised switches can recognize the probe packets and forward them as commanded.

With a statistics-based method, all information should be collected from the data plane (e.g., the number of transmitted / received / dropped packets, the packet type, the packet size, and the arrival/departure times) and compared to determine the compromised switches. T. Chao *et al.* [18] explored practical solutions for localizing and mitigating malicious switches by three techniques: active probing, statistics checking and packet obfuscation. A. Kamisiński proposed FlowMon [19], a methodology for SDN controllers to detect malicious switches in SDN networks by the statistics-based method. P. Zhang *et al.* [20] presented FOCES, a network-wide forwarding anomaly detection method in SDN, which can check the forwarding behaviors of all flows in the network simultaneously. K. Zhang [21] proposed a proactive detection mechanism to detect MITM attacks in SDN by collecting all information of each flow in the network and checking if any flow satisfies the traffic characteristic of MITM attack. H. Zhou *et al.* [22] hires a backup controller to collect and compare the information of network events from master controllers and switches. R. A. Eichelberger [23] tries to track every packet, notify the controller every time a packet comes through a switch in SFC. Y. Qiu [24] defines and checks the key information of SFC policy, collect the flow rule of network and compare the relations of flow rules and flow pattern. The statistics-based method can find out the compromised switches, but it does not support real-time detection because it

Table 1. Compromised switch attack solutions based on category

References	Solution type	
	Probe-based	Statistics-based
[12]	✓	
[13]	✓	
[14]	✓	
[15]	✓	
[16]	✓	
[17]	✓	
[18]	✓	✓
[19]		✓
[20]		✓
[21]		✓
[22]		✓
[23]		✓
[24]		✓
[25]	✓	✓

needs time to gather data and only works after packets are forwarded. Moreover, packets can be forwarded without being sent to middleboxes, which bring us to the middlebox-bypass attack in the next subsection. In our previous work, we also use the hybrid of probe-based and statistics-based method, but only for single service function chain [25]. The compromised switches attack solutions based on categories are shown in Table 1.

B. Middlebox-Bypass Attack

There are several solutions to defeat this type of attack [9], [10]. With such solutions, tags are added to unused packet header fields to track the process on the middleboxes. When a middlebox receives a packet, it needs to check these special tags to make sure the received packet was correctly processed by the previous middlebox within the chain, and that the packet was forwarded back to the switch. However, this method can only work when compromised switches only bypass the middleboxes. In other attack cases of compromised switches (for example, if a packet is dropped), the next-hop middlebox cannot verify the tags, which interferes with the detection mechanism. If a packet is duplicated, the tag verification process is still correct, although the number of packets inside the SFC is increased numerous times. Besides, adding tags to every packet and processing them can cause delays and overhead for the network. Moreover, this mechanism requires the egress switches to have the ability to untag the packet before sending it to the end hosts. In SFC, every switch can be an ingress or egress switch of a service chain, which means every switch must be modified to handle this task. An egress switch can also be compromised, which cannot be detected through this mechanism. The tagging scheme has been said to be lightweight and effective, although it needs to be integrated with other solutions to handle other compromised switch attacks.

Recognizing the need to resolve SFC vulnerabilities based on the related studies above, we propose an approach to detect both the middlebox-bypass attacks and all other compromised switch attacks.

IV. SYSTEM OVERVIEW

In this section, we first analyze the objectives of our mechanism and then describe how the system is designed using the main components. The detailed operating procedures are described in Section V and VI.

A. Objectives

To resolve the mentioned SFC vulnerabilities in Section II, our mechanism needs to achieve the following objectives:

- **Efficiency**: The mechanism needs to efficiently detect both a middlebox-bypass attack and all compromised switch attacks mentioned in Section II that are not fully detected by the other solutions.
- **Effectiveness**: Per-packet checking, as mentioned earlier, can be a good approach for real-time checking but may cause an overhead in the network. Moreover, per-packet checking needs an egress switch to modify the packet back to the original state before sending it to the end host, which requires another technique for this switch [9]. We decided to use the per-flow checking (a probe-based method) to minimize the overhead, and thus the middleboxes only have to check the probe packets.
- **Security**: Is one of the most essential requirements for the detection process. If an attacker can determine the detection method, the mechanism will be compromised. The combination of multiple algorithms can protect the detection process from attackers.
- **High Performance**: Packet loss can inevitably become a challenge if the packet processing procedure is insufficiently fast. One of the solutions is to use a temporary buffer to store the packets before processing, although this can be difficult with some middleboxes that do not have sufficient physical resources. The packet processing procedure needs to be fast and effective. With our mechanism, only probe packets with a key are analyzed, and other packets are only checked to obtain the necessary information, which ensures a fast processing speed. Although this may not support real-time detection, if we increase the number of probe packets, we can detect abnormal actions more quickly.
- **Applicability**: The detection procedures should be sufficiently simple to apply in practice. We only set up the detection programs on controllers and middleboxes, where other administrative programs and service functions are installed. By contrast, in a real system, the switches normally have limited resources and do not support additional detection functions other than packet forwarding.

To satisfy the above objectives, there are a few assumptions that we should rely on when designing the system. First, we must exclude cases in which there are collaborations between compromised switches. Although a few solutions have been proposed to detect this type of attack, they have a high delay and low accuracy or try to prevent collaboration from the beginning [14]–[16]. Most other solutions also try to avoid such collaborations because it is difficult to detect instances in which compromised switches can help each other spoof the statistics and share information. Finally, middleboxes, controllers, and the connections between them need to be trustworthy. If attackers can access the middleboxes or controllers, the pre-assigned key

can be exploited and all of the detection and prevention mechanisms will be destroyed.

B. Methodology

Our mechanism detects compromised switch and middlebox-bypass attacks by sending probe packets for each SFC chain (probe-based method) and continuously collects information from the middleboxes (statistical-based method). Middleboxes are programmed to handle packets and alert the controller whenever a probe packet is received without a correctly processed key, which is caused by a middlebox-bypass attack. By monitoring all information of the packets that also pass through the middlebox, the controller can find other compromised switch attacks. We only modify the controller and middleboxes to handle our developed algorithms and put the operation of switches out of our work because of the potential for compromise and the lack of resources on such devices. The task blocks of the system are shown in Fig. 4, and can be described as follows:

- Topology setup: Sets up network topology based on the requirements of applications and the Statistics Analyzing Module.
- Key sets generation: Creates and assigns special key sets for each middlebox.
- Key verification: Detects the middlebox-bypass attacks by checking the key in probe packets.
- Statistics generation: Prepares statistics of packets that come through middleboxes and sends them to the controller to find out other attacks by compromised switches.
- Statistics analysis: Analyze statistics from middleboxes to find out other attacks by compromised switches, and send a request to the Controller Module to change topology.

The key sets generation and key verification modules belong to a probe-based method, which can be used to detect middlebox-bypass attacks. The statistics generation and statistical analysis are used in statistics-based methods and can find other compromised switch attacks. By combining both probe-based and statistics-based methods, we can take advantage of the strength of each method and solve the problems inherent to the other proposals. A summary comparing our proposal with other current solutions is provided in Table 2.

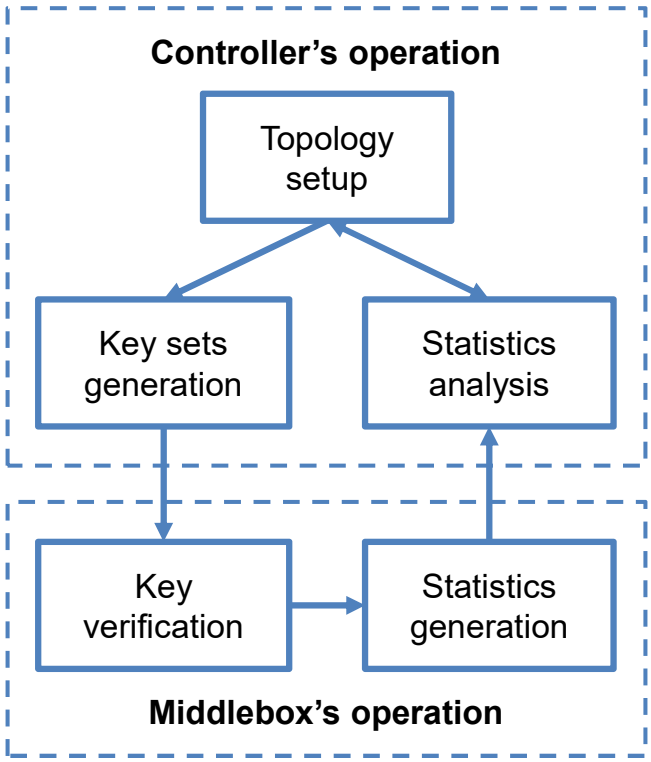


Fig. 4. Tasking blocks in the system.

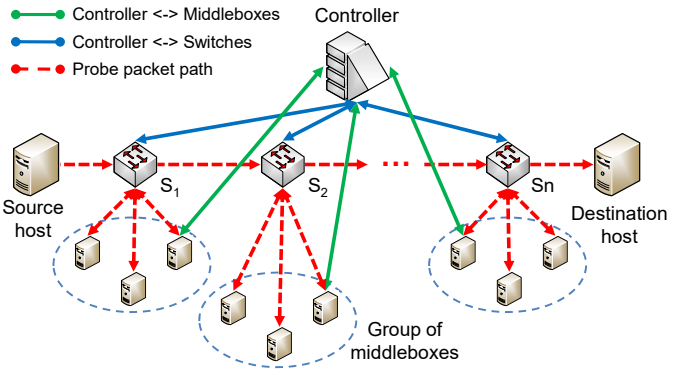


Fig. 5. General system architecture. Middleboxes with the same service function can be grouped to perform load balancing function, which prevents middleboxes from being overwhelmed.

C. System Architecture

We designed our system for use under two scenarios: single and multiple service function chains. The operations of the system in a single service function chain are described in Section V, and Section VI describes the operations for multiple service function chains. The general system architecture is shown in Fig. 5. The system under both scenarios has the same three main components as follows:

C.1 Controller

Consists of three modules, which is depicted in Fig. 6. We only consider the case of a single controller in the system. Multiple controllers will require synchronization between controllers, which is not the main goal of our research.

- Controller module: Defines the service function chains in the network. This module installs the flow rules on switches as well as connects them to middleboxes and sends the updated network topology to Key Generator Module and Statistics Analyzing Module.
- Key generator module: Based on the most up-to-date network topology, this module creates and assigns new key sets to middleboxes. These key sets are used to check the integrity of probe packets in the service chains.
- Statistics analyzing module: Based on the most up-to-date network topology, this module analyzes statistics from middleboxes to find out abnormal actions and send a request to Controller Module to set up new topology to protect SFC if an attack is detected.

Table 2. Comparison between our proposal and current solutions

Methods		Current solutions	Our proposal
Probe-based method	For compromised switch attacks	- Can be disabled if compromised switches can recognize the probe packets.	- Adds special keys into unused packet header fields, and therefore only middlebox can recognize the probe packets.
	For middlebox-bypass attack	- Can only detect middlebox-bypass attack. - Causes delay and overhead for the network by adding keys to every packet. - Needs to modify egress switches to have the ability of untagging packets.	- Uses statistics-based method to detect other compromised switch attacks. - Adds keys only upon probe packet and processes only probe packets. - Only modifies the middlebox, which has better computing resources than switches.
Statistics-based method	For compromised switch attacks	- Does not support real-time detection. - Cannot detect middlebox-bypass attack.	- Uses probe-based method to support real-time detection and detect middlebox-bypass attacks.

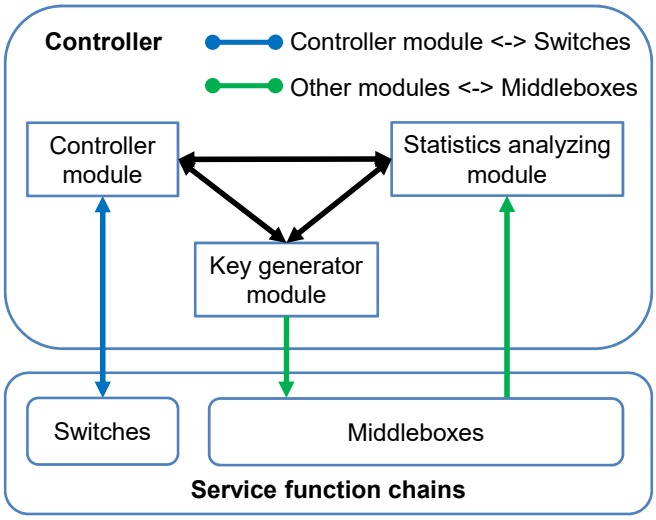


Fig. 6. Modules in the controller.

C.2 Switches

Follow the command from the controller to connect middleboxes to make service function chains.

C.3 Middleboxes

Besides the main service functions of each middlebox, they are programmed to check every received packet from switches, record the packet information to make statistics, process the probe packet, and send statistics to the controller. Middleboxes with the same service function can be grouped to perform load balancing function, which prevents middleboxes from overwhelmed.

In the next sections, we will propose more details about the operations of the above tasks in the view of the controller and middleboxes.

V. SYSTEM DESIGN IN SINGLE SERVICE FUNCTION CHAIN

The system architecture in the single service function chain scenario is depicted in Fig. 7. For ease of understanding, we assume our system with a single controller and a single service function chain (contains hosts, switches, and middleboxes, each middlebox connects to one switch). The detailed operating algorithms of the controller and middleboxes are depicted in Figs. 8 and 9 respectively.

A. Controller's Operation

A.1 Key Sets Generation

Take the service function chain in Fig. 7 as an example. The packet path is $SourceHost - S_1 - Middlebox_1 - S_1 - ... - S_n - Middlebox_n - S_n - DestinationHost$. If we set the chain so that packets are sent from the controller and come back to the controller, compromised switches can realize this and operate like normal switches. After the Controller Module set up the above service chain, it sends the updated topology to the key generator module and statistics analyzing module. The key generator module creates and assigns new key sets to each middlebox randomly, as shown in Table 3. These key sets can be used to check the integrity of probe packets in the service chains. Each key set contains many keys, and each key is the hash result of the compatible key in the previous set. For example, the key can be calculated as in (1). We use a 10-bit key which is also the same condition with FlowCloak [9]. This research claimed that the success rate of random guessing under 0.1%. We put our key into the unused header field of the packet, which has limited capacity. If we use longer keys (e.g, 128 or 256-bit), we need to put it in another part of the packet (e.g, payload), which can bring other risks for security. Longer key also needs a longer time to be processed, which can cause more delay in the packet processing procedure.

$$\begin{aligned} key_{21} &= hash(key_{11}) \\ key_{31} &= hash(key_{21}). \end{aligned} \tag{1}$$

After a certain period, we refresh the key sets by re-executing the above procedures. By creating new key sets and randomly assign different keys for each probe packet in the same service chain, we reduce the probability that an attacker can guess the exact key and spoof the probe packet. If attackers can do that, the possibility they can spoof two probe packets continuously is nearly zero, except the case that they know our key set and

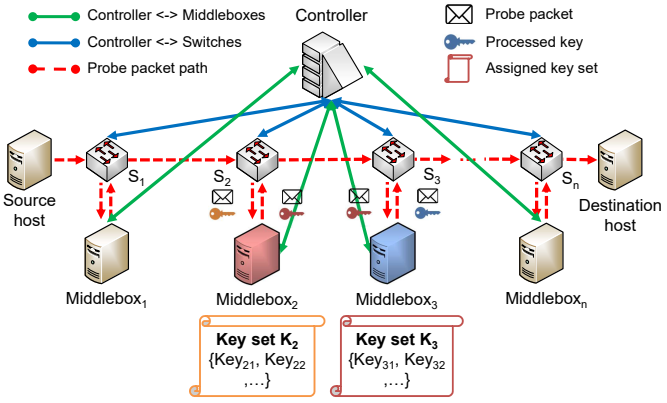


Fig. 7. System architecture in a single service function chain. At first, the middlebox receives a key set. When receiving a new probe packet, the middlebox checks the integrity of the packet, then modifies the key in the probe packet and sends the packet to the next-hop middlebox. The middlebox also prepares the statistics and send them to the controller.

Table 3. Key set for middleboxes

Key set K_1	**Key set** K_2	**Key set** K_3	**...**	**Key set** K_n
Key_{11}	Key_{21}	Key_{31}	...	Key_{n1}
Key_{12}	Key_{22}	Key_{32}	...	Key_{n2}
Key_{13}	Key_{23}	Key_{33}	...	Key_{n3}
...	...	...	...	...

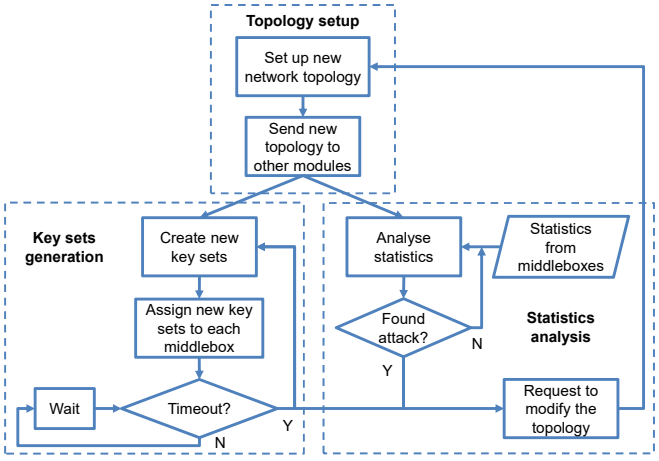


Fig. 8. Operating algorithms of the controller are depicted in three blocks: Topology Setup, Key Sets Generation and Statistics Analysis.

random key order. Furthermore, the numerical order and the key value of the probe packet are also monitored by the controller, which restricts other guessing methods.

A.2 Statistics Analysis

After receiving the statistics from middleboxes and the most up-to-date network topology from the key generator module, the statistics analyzing module analyzes and compares this information to find out the abnormal actions. If an attack is detected, this module requests other modules to set up a new network topology and create new key sets to protect the network. The detailed operations are described in Section VI.

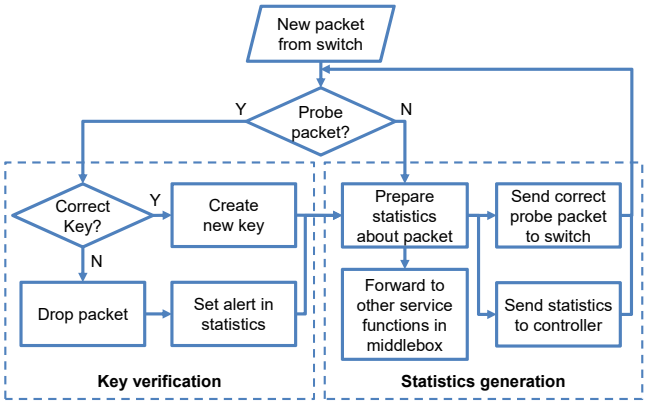


Fig. 9. Operating algorithms of middleboxes are depicted in two blocks: Key Verification and Statistics Generation.

B. Middlebox's Operation

B.1 Key Verification

Take the service function chain in Fig. 7 as an example again. When $Middlebox_2$ receives a new packet from its attached switch S_2, it first checks if this is the probe packet or normal packet. We use an unused bit in the header to help middleboxes recognizes the probe packet. As mentioned in Section IV.A, only probe packets are processed by middleboxes to ensure the effectiveness and high performance of the system. If this is a probe packet, $Middlebox_2$ needs to check whether it was processed correctly or not by referring to the key set. For example, after receiving a probe packet with the key named key_x, $Middlebox_2$ defines the integrity of this packet by checking whether the key_x is in the key set K_2 or not. If this probe packet was correctly processed, $Middlebox_2$ will replace the key_x by key_y, which is calculated by hash function as in (2). After this process, $Middlebox_2$ forwards the probe packet back to S_2 to transfer to the next-hop middlebox ($Middlebox_3$). The $Middlebox_3$ now needs to check this probe packet with the same procedure as above. If the probe packet is not correct, which means this can be a middlebox-bypass attack, $Middlebox_3$ drops the packet and triggers an alarm to the controller by creating an alert in

the statistics.

$$key_y = hash(key_x). \quad (2)$$

In practice, we do not need an additional method to check the integrity of the last switch in the chain, while FlowCloak [9] needs another technique on egress switches. As mentioned above in subsection III.A, a switch typically belongs to multiple SFC chains, which means that it can be checked through the operation of other chains. In the case of only one chain as the example above, we run a program on the Destination Host to check the probe packets from Sn just like other middleboxes.

B.2 Statistics Generation

For other packet types, the parameters that are shown in Table 4 are recorded to make the statistics report. Those packets are then forwarded to other service functions in the middleboxes. The middlebox sends the statistics report to the controller, deleted the statistics, and waits for new packets.

VI. SYSTEM DESIGN IN MULTIPLE SERVICE FUNCTION CHAINS

The operation of each middlebox in multiple service function chains is the same as in a single service function chain. In the controller, to detect other compromised switch attacks (e.g., packet dropping, packet duplication, packet manipulation, and a weight adjustment), the statistics processing module applies a statistical analysis function, in which it always listens to statistics sent from the middleboxes. By comparing these statistics between middleboxes and checking the alert signal, this module can detect the compromised switches and middlebox-bypass attacks.

Take the SFC chain in Fig. 10 as an example. For the chain number 1, the packet path is $...-S_1-Middlebox_2-S_1-S_2-Middlebox_5-S_2-....$ If $Middlebox_2$ reports that it forwarded 100 packets to S_1 (75 normal packets and 25 probe packets) in a period (calculated by the controller) so that $Middlebox_5$ should report that it also received 100 packets with the same number of normal and probe packets in the same period. We set several thresholds for the difference of statistics (because of the latency for packet processing by other service functions in the middlebox, transmission delay or other reasons). For example, if the threshold for the number of received packets is 5%, this means that $Middlebox_5$ should receive at least 95 or at most 105 packets in the same period. If $Middlebox_5$'s report shows that it only gets 90 packets, this means that S_2 does not forward all of the packets to $Middlebox_5$ (missing at least 5 packets), and this can be a packet dropping attack. In another case, if $Middlebox_5$ reports that it received 150 packets in that period, this means an attack is happening (packet duplicating or weight adjusting attack). Furthermore, to detect other attack cases, the Statistics Processing Module need to analyze other statistics parameter in Table 4.

On the other hand, multiple middleboxes can be connected to a switch to perform a load balancing function. When the network traffic increases robustly, which can overwhelm a single middlebox by forcing it to process every packet, the switches

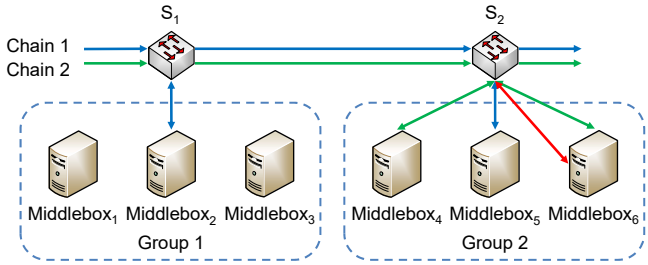


Fig. 10. Multiple service function chains with groups of middleboxes. If S_2 is compromised, it can be commanded to forward all the packets in chain 2 to only $Middlebox_6$ instead of steer equally to $Middlebox_{4,5,6}$, which can make $Middlebox_6$ overwhelmed (weight adjusting attack).

Table 4. Statistical parameters

Parameter	Description
ChainID	Assigned ID for each service function chain
SrcIP	IP address of source host
DesIP	IP address of destination host
PktType	Type of packet
PktLeng	Payload size in bytes
ProcTime	Total time for processing packet
TxNum	Total number of received packets
DrNum	Total number of transmitted packets
RxNum	Total number of dropped packets
Alert	Alert signal is raised when finding out an incorrect key from probe packet

can be used to dynamically steer the network traffic to the same middleboxes in a group. For example in Fig. 10, S_2 can divide equally the traffic from chain 2 to $Middlebox_{4,5,6}$. If S_2 is compromised, it can be commanded to forward all the packets to only $Middlebox_6$, which can make it overwhelmed (weight adjusting attack). By comparing the statistics from $Middlebox_{4,5,6}$, the controller can calculate the load balancing ratio in group 2 and detect this kind of attack.

After detecting any attacks or alerts from the statistics, the statistical analysis module requests the other modules to set up a new network topology and create new key sets to eliminate the compromised switches, thus protecting the network. We leave the handling of compromised switches after detection for future studies. In the next section, we describe the implementation and evaluation of our scheme in detail.

VII. EXPERIMENTS AND EVALUATION

In this section, we evaluate the performance and efficiency of our proposed mechanism through three experimental studies. The first study deals with the latency for processing probe packets on the middleboxes. In the second study, we evaluate the CPU and memory consumption caused by our mechanism on both middleboxes and the controller, respectively. These two studies will help us evaluate the performance and applicability of our mechanism. In the third study, the detection rate, accuracy, and false alarm rate are taken into account, which can be used to evaluate the efficiency of our solution.

The experiment topology is depicted in Fig. 11. Our proto-

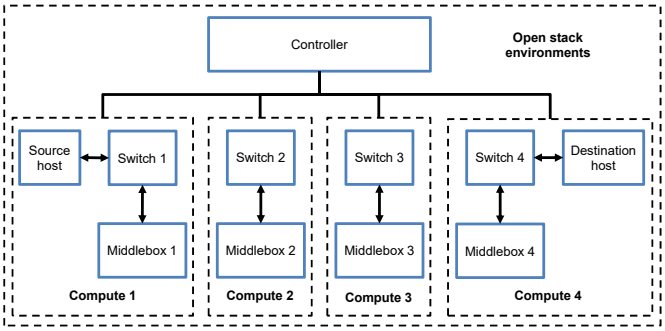


Fig. 11. Experiment topology.

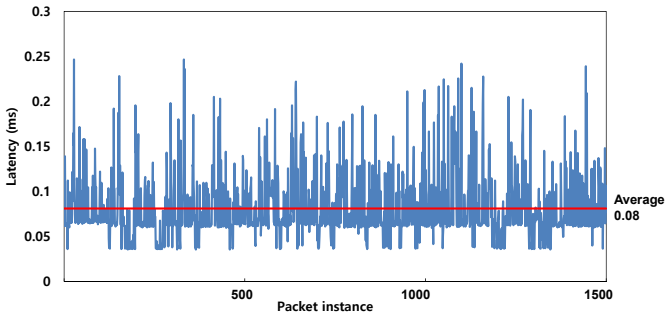


Fig. 12. Probe packet processing latency on middleboxes.

type consists of a controller program that also runs in this environment, as well as a service function program that runs on middleboxes to handle the packets. These programs need only approximately 800 lines of C code and can easily be modified and integrated with other service functions and controllers. We implemented the prototype on a computer with 12 3.4-GHz Intel i7-8700 CPUs and 48 of GB memory. Each virtual machine is allocated 2 GB of memory and 1 CPU. We use OpenStack [26] (Stein version), a free open-source software platform used for cloud computing, to create the service function chaining environment. By using OpenStack, we can easily create virtual middleboxes and switches, as well as create service function chains.

A. Performance Evaluation

A.1 Probe Packet Processing Latency

As mentioned in Section IV.A, the packet processing procedure needs to be fast and effective, and we only handle the probe packet to reduce the overhead on the packet processing. Normal packets are only checked to generate the statistics, which requires an insignificant amount of time. We conducted our first study by measuring the latency for processing the probe packets. We generated network traffic including normal and probe packets (1500 bytes with a pre-assigned 10-bit key, and applying the hash function FNV-1a [27], which are the same conditions as [9]) into the chain. The results in Fig. 12 show the probe packet processing latency on middleboxes at 100 probe packets per second (1.172 Mbps).

We also compared the results with other similar approaches that also need packet handling in Fig. 13. Whereas FlowCloak [9] and SFC Path Tracer [23] require 0.3 ms, the average probe packet processing latency of our work is only 0.08 ms.

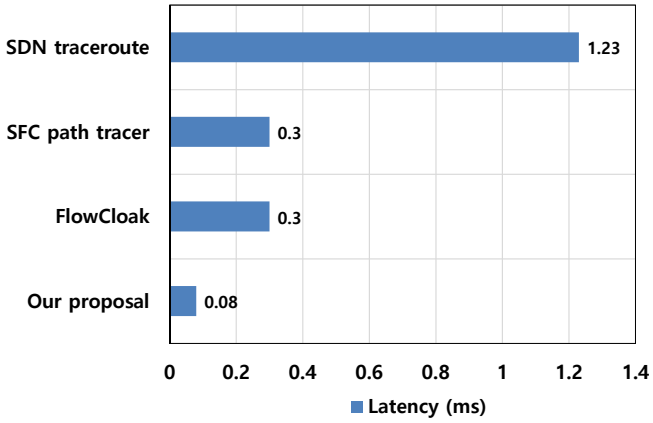


Fig. 13. Comparison of probe packet processing latency on middleboxes between related works.

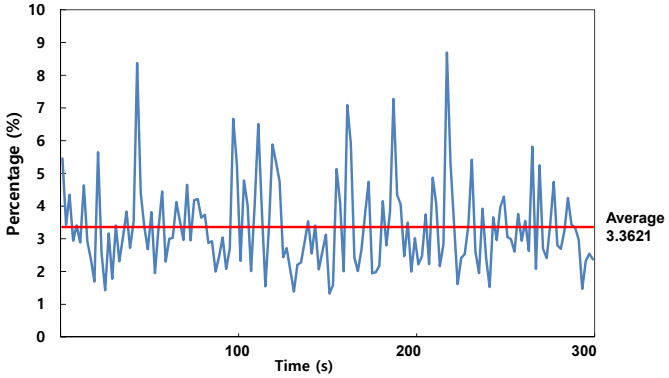


Fig. 14. CPU utilization on middlebox during the experiment.

From this result, we found that our mechanism is fast and effective. With a low-latency packet handling of 0.08 ms, we validated that our scheme can detect middlebox-bypass attacks in real time. This procedure can also be easily integrated into other service functions without noticeable delays, which increases the applicability of our mechanism.

A.2 Resources Consumption

The CPU and memory utilization on the middleboxes and controller are important parameters for evaluating the performance of our system. The results in Figs. 14 and 15 show the CPU and memory utilization of our program on middleboxes during the experiment. Our program uses only 3.3621% of the CPU and 0.01013% of memory (2 GB total) on average, which are good values for devices with limited resources, such as a middlebox. These results demonstrate that our program works well and causes minor obstacles to the operation of the middlebox. Figs. 16 and 17 show the CPU and memory utilization of our program on the controller applied during the experiment. With 7.5326% CPU and 0.00287% memory (36 GB in total) use on average, we can see that our program generates relatively little overhead on the controller. We can consider a network entity such as a controller to require enormous resources, which indicates that the resource utilization of our program is negligible.

Moreover, we compared our mechanism with other solutions

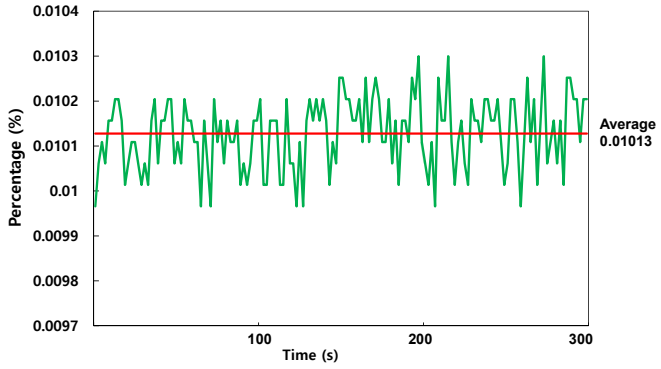


Fig. 15. Memory utilization on middlebox during the experiment.

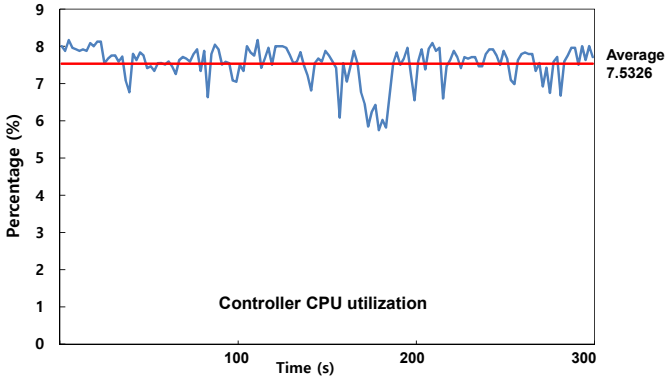


Fig. 16. CPU utilization on the controller during the experiment.

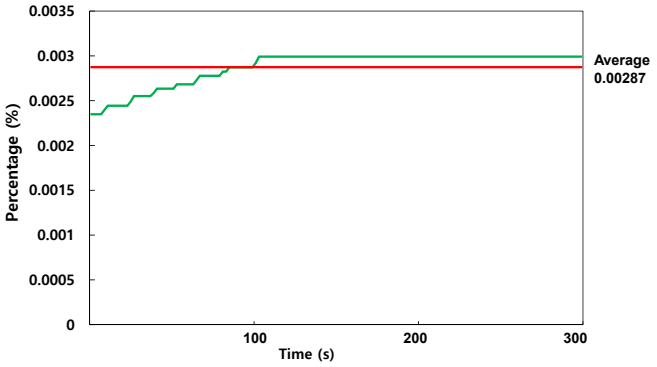


Fig. 17. Memory utilization on the controller during the experiment.

that also apply a statistics-based method (a process using the most resources). SDN-RDCD [22] is a mechanism that applies a backup controller to collect and handle the network information. This program uses only 8.8633% of the CPU and 0.8783% of the memory (16 GB in total), which are higher than those of our approach. Other solutions only focus on evaluating the efficiency of the method and omit the consumption of resources. From the above results, we can validate that our mechanism works with noticeable overhead for the system.

B. Efficiency Evaluation

B.1 Study Setup

For this study, we created some attacks and evaluated how well our system works against these attacks. We focused on two types of attacks, as mentioned above: middlebox-bypass attacks and other compromised switch attacks. The evaluation parameters are the detection rate, accuracy, and false alarm rate. We injected some incorrect probe packets to create the middlebox-bypass attacks. As mentioned in subsection IV-B, we did not intervene in the operation of the switches, and other compromised switch attacks could be easily detected through the statistical analysis procedure; thus, we did not need to tamper with the other attack cases. After the attack detection, the network topology was changed to protect the SFC. If switch 2 is compromised, and an attack near switch 2 is detected, and the system will remove the chain through this switch. After some period, we switch the network topology back to the original version. We only focus on evaluating the operation of our mechanism under a multiple service function chain scenario. The optimization of network traffic steering was not our main goal.

B.2 Results Evaluation

The detection results are divided into four categories based on particular conditions, as described in Table 5. True positive (TP) is the number of incorrect probe packets that are detected. False positive (FP) is the number of correct probe packets that are detected as incorrect packets. False negative (FN) is the number of incorrect probe packets that are detected as correct, and true negative (TN) is the number of correct probe packets that are detected as correct.

With a 10-bit key, FlowCloak [9] achieves a successful random guessing rate of under 0.1%. Because we also use the same conditions as in [9], we can count the number of false negative results equals to 0.1% of the number of packets detected as attacks (TP + FP). We then can calculate the detection rate (DR), accuracy (AC), and false alarm rate (FAR) of our system are 99.8148%, 99.539%, and 0.4625% respectively by (3), (4), and (5). The false positive, as shown in Table 5, reduces the accuracy and increase the false alarm rate. We can program the controller to ignore the alarms when the network topology is updating, which eliminates the FPs. However, we allow the system to continue to receive any alerts, even false alarms because problems with the network topology that are not caused by the controller can also occur.

$$DR = \frac{TP}{TP + FN}, \tag{3}$$

$$AC = \frac{TP + TN}{TP + TN + FP + FN}, \tag{4}$$

$$FAR = \frac{FP}{FP + TN}. \tag{5}$$

Finally, we again compared our results to the other solutions, as shown in Fig. 18. With 100% detection rate, SDN-RDCD [22] shows an impressive results with 100% accuracy, while Generic [10] has 98%, Track [17] has more than 95.08% and FlowMon [19] has 77% accuracy. From the comparison, we can see that our mechanism is effective in detecting middlebox-bypass attacks and compromised switches.

Table 5. Detection Conditions

	Detected as attack	**Detected as normal**
Actual attack	True positive (TP) Conditions: - Incorrect key. - Incorrect statistics.	False negative (FN) Conditions: - Attackers can guess the key and spoof probe packets.
Actual normal	False positive (FP) Conditions: - Incorrect key: when the network topology is changed, the controller needs time to update the key set on middleboxes while probe packets are still being sent. - Incorrect statistics: because of packet processing latency, transmission delay, etc.	True negative (TN) Conditions: - Correct probe packet. - Normal packet.

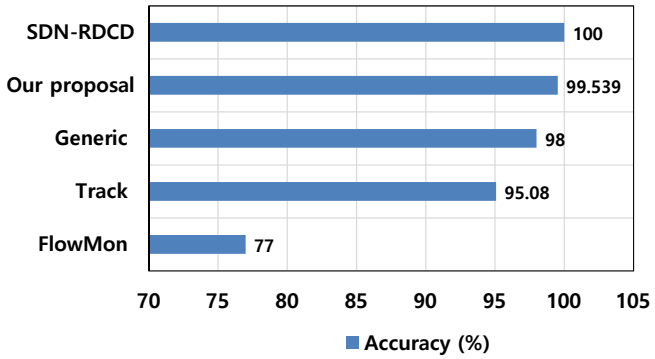


Fig. 18. Comparison of accuracy among related works.

VIII. CONCLUSION AND FUTURE WORK

Compromised switches and middlebox-bypass attacks are serious issues affecting the SFC. We presented a mechanism that not only detects both middlebox-bypass attacks and compromised switch attacks in multiple SFC scenario but also prevents them and protects the network. By combining probe-based and statistics-based mechanisms, we overcome the disadvantages of other proposed solutions and bring about robust protection to the SFC. Based on the experimental results, we found that our mechanism can detect attacks efficiently with a low packet processing latency on middleboxes, with low resource consumption, high detection rate, and high accuracy.

In the future, our mechanism can be extended to additional algorithms to increase the performance and efficiency, apply machine learning to predict attacks, improve the packet processing speed, and cover more attack cases from compromised switches.

REFERENCES

[1] D. Bhamare, R. Jain, M. Samaka, A. Erbad, "A Survey on service function chaining," *J. Netw. Comput. Applications*, vol. 75, pp. 138-155, Nov. 2016.

[2] A. M. Medhat *et al.*, "Service function chaining in next generation networks: State of the art and research challenges," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 216-223, Feb. 2017.

[3] G. Mirjalily and Z. Luo, "Optimal network function virtualization and service function chaining: A survey," *Chinese J. Electronics*, vol. 27, no. 4, pp. 704-717, July 2018.

[4] L. Guo, J. Pang and A. Walid, "Dynamic service function chaining in SDN-enabled networks with middleboxes," in *Proc. IEEE ICNP*, 2016.

[5] S. Huang, F. Cuadrado, and S. Uhlig, "Middleboxes in the internet: A HTTP perspective," in *Proc. TMA*, 2017.

[6] S. W. Brim, B. E. Carpenter, "Middleboxes: Taxonomy and issues," RFC 3234, Mar. 2013.

[7] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simplifying middlebox policy enforcement using sdn," in *Proc. ACM SIGCOMM*, 2013.

[8] J. Sherry *et al.*, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proc. ACM SIGCOMM*, 2012.

[9] K. Bu, Y. Yang, Z. Guo, Y. Yang, X. Li and S. Zhang, "FlowCloak: Defeating middlebox-bypass attacks in software-defined networking," in *Proc. IEEE INFOCOM*, 2018.

[10] X. Zhang, Q. Li, J. Wu, and J. Yang, "Generic and agile service function chain verification on cloud," in *Proc. IEEE/ACM IWQoS*, 2017.

[11] M. Antikainen, T. Aura, M. Särelä, "Spook in Your network: Attacking an SDN with a compromised OpenFlow switch," in *Proc. NordSec*, 2014.

[12] P. Chi, C. Kuo, J. Guo and C. Lei, "How to detect a compromised SDN switch," in *Proc. IEEE NetSoft*, 2015.

[13] Y. Chiu and P. Lin, "Rapid detection of disobedient forwarding on compromised OpenFlow switches," in *Proc. IEEE ICNC*, 2017.

[14] Y. Ke, H. Hsiao and T. H. Kim, "SDNProbe: Lightweight fault localization in the error-prone environment," in *Proc. IEEE ICDCS*, 2018.

[15] C.-T. Kuo, P.-W. Chi, V. Chang, and C.-L. Lei, "SFaaS: Keeping an eye on IoT fusion environment with security fusion as a service," *Future Generation Computer Systems*, vol. 86, pp. 1424–1436, Sept. 2018.

[16] N.Y. Vijayvergiya, "Detecting collaborative attacks caused by compromised switches in SDN," Masters thesis, Indian Institute of Technology Hyderabad, 2017.

[17] Y. Zhang, L. Cui, F. P. Tso, and Y. Zhang, "Track: Tracerouting in SDN networks with arbitrary network functions," in *Proc. IEEE CloudNet*, 2017.

[18] T. Chao et al., "Securing data planes in software-defined networks," in *Proc. IEEE NetSoft*, 2016.

[19] A. Kamisiński and C. Fung., "FlowMon: Detecting malicious switches in software-defined networks," in *Proc. ACM CCS*, 2015.

[20] P. Zhang et al., "FOCES: Detecting forwarding anomalies in software defined networks,"in *Proc. IEEE ICDCS*, 2018.

[21] K. Zhang and X. Qiu, "CMD: A convincing mechanism for MITM detection in SDN," in *Proc. IEEE ICCE*, 2018.

[22] H. Zhou et al., "SDN-RDCD: A real-time and reliable method for detecting compromised SDN devices," *IEEE/ACM Trans. Netw.*, vol. 26, no. 5, pp. 2048-2061, Oct. 2018.

[23] R. A. Eichelberger, T. Ferreto, S. Tandel, and P. A. P. R. Duarte, "SFC path tracer: A troubleshooting tool for service function chaining," in *Proc. IFIP/IEEE IM*, 2017.

[24] Y. Qiu, X. Qiu and Y. Cai, "Service function chaining policy compliance checking," in *Proc. IEEE/IFIP NOMS*, 2018.

[25] N. C. Thang and M. Park, "Detecting compromised switches and middlebox-bypass attacks in service function chaining," in *Proc. ITNAC*, 2019.

[26] OpenStack. [Online] Available: https://www.openstack.org/

[27] Fowler-Noll-Vo hash function. [Online] Available: https://tinyurl.com/qxcqjhk



Nguyen Canh Thang received the B.S. degree of Engineer in Control Engineering and Automation from the Hanoi University of Science and Technology, Hanoi, Vietnam, in 2018. He is currently pursuing a M.S. degree in information communication at Soongsil University, Seoul, Korea. His research interests include software-defined networks, service function chaining, and network security.



Minho Park received the B.S.and M.S. degrees in electronics engineering from Korea University, in 2000 and 2002, respectively, and the Ph.D. degree from the School of Electrical Engineering and Computer Science, Seoul National University, Seoul, Korea, in 2010. He is currently an Associate Professor with the School of Electronic Engineering, Soongsil University, Seoul. His current research interests include wireless networks, vehicular communication networks, network security, and cloud computing.