A hybrid prevention method for eavesdropping attack by link spoofing in software-defined Internet of Things controllers

Article *in* International Journal of Distributed Sensor Networks · November 2017

DOI: 10.1177/1550147717739157

CITATIONS
0

READS
3

2 authors, including:

Nguyen Tri-Hai
Soongsil University

9 PUBLICATIONS **1** CITATION

SEE PROFILE

Research Article

International Journal of Distributed Sensor Networks

International Journal of Distributed Sensor Networks
2017, Vol. 13(11)

DOI: 10.1177/1550147717739157
journals.sagepub.com/home/ijdsn
SAGE

A hybrid prevention method for eavesdropping attack by link spoofing in software-defined Internet of Things controllers

Tri-Hai Nguyen[1] and Myungsik Yoo[1,2]

Abstract

With the rapid growth of Internet of Things technologies, the management and control of Internet of Things networks face remarkable challenges. As such, software-defined networking, which decouples the control layer from data layer, results in various advantages. An association of software-defined networking and Internet of Things, which is referred to as software-defined Internet of Things, provides a robust platform to improve the management and control abilities of Internet of Things networks. However, these benefits have resulted in an increase in the number of malicious attacks on logically centralized controllers. For that reason, we have performed a specific vulnerability analysis in the link service, where the controller learns network topology through discovering every link between switches. In addition, we demonstrate link spoofing attacks on the link service, and discuss a hybrid countermeasure to address this security problem.

Keywords

Software-defined Internet of Things, link service, eavesdropping, authentication, Link Layer Discovery Protocol latency

Date received: 16 May 2017; accepted: 3 October 2017

Handling Editor: An Liu

Introduction

The Internet of Things (IoT) comprises a network of physical devices incorporated sensors and embedded systems that interact with each other and connect to the Internet.[1] Many devices generate a huge amount of data that require significant effort and processing to convert it to valuable information. In addition, controlling a large volume of data requires new concepts in managing IoT networks to improve the efficiency.[2,3] As the next generation of networking architectures, software-defined networking (SDN) can help IoT technologies meet these demands by separating the control layer from the data layer in the network resulting in numerous benefits including centralized control, abstracted network devices, and flexible, automated reconfiguration of the network.[4,5]

The trend to integrate SDN with IoT, which is referred to as software-defined Internet of Things (SDIoT), has seen noticeable growth.[6–10] However, when the logic of the forwarding behavior of the network is centralized in the controller, a single point of failure may be caused.[5] In the controller, the link service is a fundamental and crucial service that determines the links between switches in the network. However, it is found that the link service is vulnerable to attacks because it accepts all Link Layer Discovery Protocol (LLDP) packets, which are used to discover

[1]Department of ICMC Convergence Technology, Soongsil University, Seoul, Republic of Korea
[2]School of Electronic Engineering, Soongsil University, Seoul, Republic of Korea

Corresponding author:
Myungsik Yoo, School of Electronic Engineering, Soongsil University, 369, Sangdo-ro, Dongjak-gu, Seoul 06978, Republic of Korea.
Email: myoo@ssu.ac.kr

links among switches. Thus, an attacker can inject fake LLDP packets or simply forward LLDP packets from a target switch to another to make a fake link between them. It affects to all topology-dependent services (e.g. packet routing) and leads to a denial of service due to link disruption and a novel eavesdropping attack, which is presented in this article.

In brief, this work contains following major contributions. First, a vulnerability assessment is performed on the link service. It is found that the attacker can launch a novel eavesdropping attack to overcome existing prevention methods. Second, a hybrid countermeasure is proposed and discussed. Finally, an experiment is carried out to show the effectiveness of the proposed method.

The rest of this article is organized as follows. Section "Background" introduces the overview of SDIoT and the link service in an SDIoT controller. Section "Related work" provides an overview of related work that is linked to attacks on the link service. Section "Novel eavesdropping attack" presents a new eavesdropping attack that can bypass the current security mechanisms. Section "Hybrid prevention method" discusses the proposed defense method and evaluates its detection performance. Finally, the summary is presented in section "Conclusion."

Background

In this section, we provide an overview of the SDIoT and the link service in the SDIoT controller.

Software-defined Internet of Things

IoT and SDN are two modern paradigms that will dramatically shape future computer networks.[3,5] Some prior works have introduced an integrated SDN with IoT, which has been referred to as SDIoT. Qin et al.[6] proposed an extended Multinetwork INformation Architecture (MINA) with a layered SDN controller for IoT that can deal with network heterogeneity, the difference in service quality, and so on. Jararweh et al.[7] proposed an SDIoT controller that operates as an orchestrating middleware between the data-as-a-service layer and the physical layer, composed of controllers of SDN, storage, security, and so on. Also, various use cases have been presented and implemented, including SDIoT for smart urban sensing,[8] mobility management in urban-scale,[9] and end-to-end service network orchestration.[10] Figure 1 shows the simplified view of the SDIoT architecture, which is described as follows:

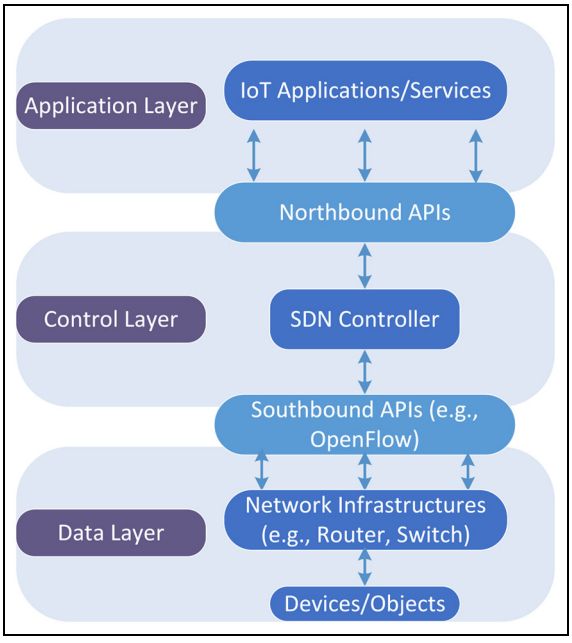

Figure 1. Overview of SDIoT architecture.

> The *application layer* provides various IoT applications and services. The application layer communicates with the control layer through northbound application programming interface (APIs), and the control layer provides an abstraction of the network infrastructures for the application layer.
> The *control layer* contains a centralized controller (e.g. Floodlight,[11] POX,[12] and OpenDaylight[13]), which logically maintains a global and dynamic network view, takes requests from the application layer via northbound APIs, and manages the network infrastructure via southbound APIs, for example, OpenFlow.[14]
> The *data layer* consists of network infrastructures. When a new packet goes through a switch, if it matches an existing rule in Flow Tables of the switch, the switch will handle the packet according to the matching rule. Otherwise, the switch sends a Packet-In message to the controller to require for proper operation. The controller then provides a Packet-Out message for the packet processing.

Link service in SDIoT controller

In contrast with a conventional network or IoT schemes, the link service in an SDIoT network is special due to its logically centralized controller. The link service uses the LLDP[15] to find the switch-to-switch links in the network and provides such visibility information to upper layer services and applications. Figure 2 illustrates the basic link discovery procedure between two switches as follows:

1. The controller sends a Packet-Out message to switch 1 along with its LLDP packet.
2. Switch 1 broadcasts this LLDP packet to all its active ports. Switch 2 receives the LLDP packet via a port that connects to switch 1.

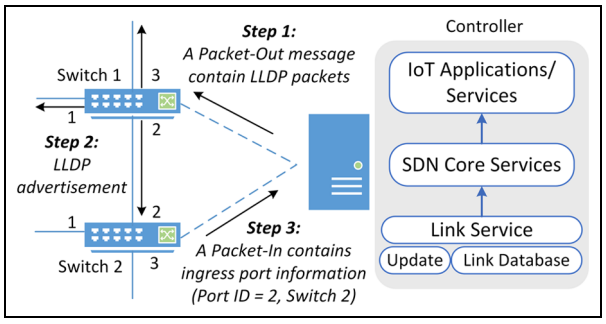

Figure 2. Link service and its basic operations in SDIoT controller.

3. Switch 2 sends the LLDP frame back to the controller through a Packet-In message. When the controller receives the Packet-In message, it creates a link between the combination of the port number (Port) and Datapath ID (DPID) of switch 1 and Port/DPID of switch 2.

The procedure must be repeated to create a link from the opposite direction. The process is periodically performed for every switch in the network with a new discovery cycle initiated at fixed intervals.

Related work

Due to a lack of integrity detection of the LLDP packets in the link service of the controller, an attacker can create a fake link among two switches by a link spoofing attack that is LLDP Spoofing attack or LLDP Forwarding attack.[16,17] An example is shown in Figure 3.

In *LLDP Spoofing attack*, an attacker (host 1) generates a new LLDP packet or modifies the content of a received LLDP packet, that is, the sender is port number 1 of switch 3 and the receiver is port number 1 of switch 1, and sends it back to switch 1. The controller receives the fake LLDP packet from switch 1 and updates the link database based on the sender and receiver information of the packet. As a result, a fake link is created between switch 1 and switch 3.

In *LLDP Forwarding attack*, the attacker must control host 1 and host 3. As part of the normal link discovery procedure, host 1 captures a LLDP packet that the sender is port number 1 of switch 1 and then sends this packet to its partner host 3 via a private connection. After that, host 3 forwards the received LLDP packet to port number 1 of switch 3. The controller receives and accepts the LLDP packet from switch 3 because this packet is valid. A spoofing link from switch 1 to switch 3 is successfully built.

To guard against these attacks, TopoGuard[16] appends a static key-hash message authentication code

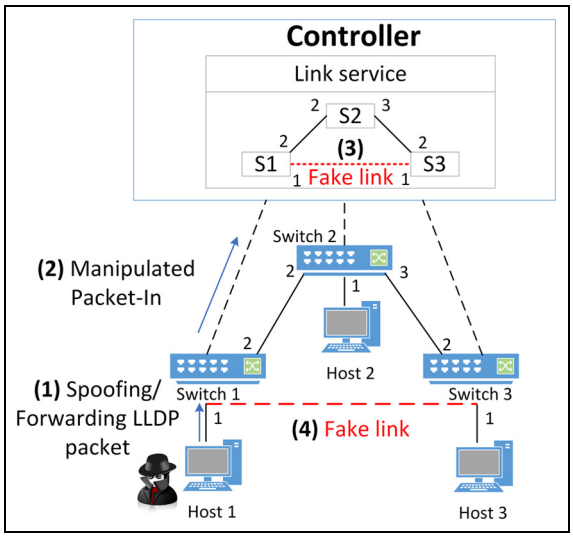

Figure 3. An example of a link spoofing attack between switch 1 and switch 3.

(HMAC)-based authenticator type, length, value (TLV) into an LLDP packet and verifies it when getting the LLDP packet in the controller. TopoGuard also assigns the switch ports as SWITCH and HOST based on the types of received packets and blocks the LLDP packets from the HOST ports. Besides that, SPHINX[17] maintains a flow graph, which provides a clean mechanism to aid in the detection of various violations for the network topology, but it is ineffective in large-scale and dynamic networks. Furthermore, we find that when the attacker suppresses all host-generated traffic and only forwards LLDP packets to perform the link spoofing attack, these schemes do not provide any notification of the violation.

An attacker can create a fake link between two switches by link spoofing attacks due to the lack of LLDP packets verification schemes in the link service. Once the topology information provided by this link service is poisoned by the fake link, all dependent operations such as routing or forwarding applications are malfunctioning. This can lead to denial of service due to link disruption and eavesdropping attack. The problem of link disruption has been known and solved by the aforementioned methods. However, the eavesdropping by LLDP Forwarding attack has not been mentioned in the literature, and thus, there is no existing solution for it. The detail about the novel eavesdropping attack and the proposed countermeasure will be explained in the following sections.

Novel eavesdropping attack

As far as we know, none of the current controllers contain a security mechanism in the link service to

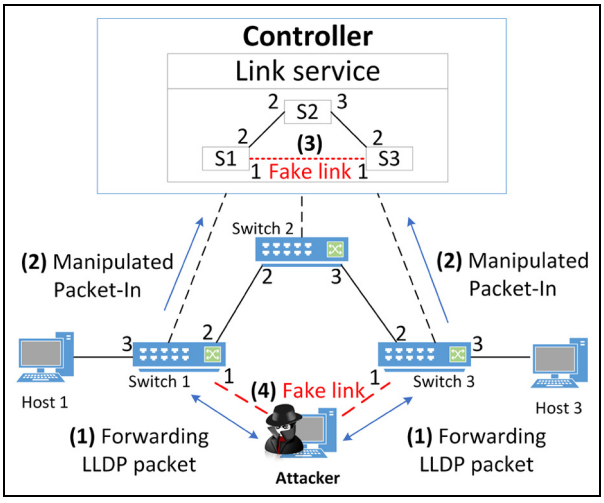

Figure 4. A novel eavesdropping attack.

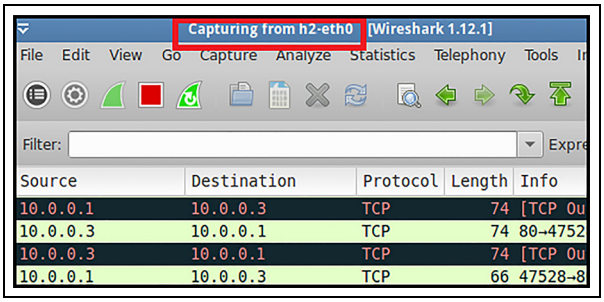

Figure 5. A successful eavesdropping attack.

comprehensively prevent the LLDP Forwarding attack. Even the current mainstream controllers, such as Floodlight, OpenDaylight, and POX, still have this security issue. The existing link spoofing prevention methods implemented in these controllers such as TopoGuard and SPHINX cannot completely prevent the LLDP Forwarding attack.

In this article, it is found that the LLDP Forwarding attack can lead to a new kind of eavesdropping. The novel eavesdropping attack is based on the fact that the existing prevention methods are useless if the attacker suppresses all host-generated traffic (e.g. address resolution protocol (ARP), domain name system (DNS)) and only performs the LLDP Forwarding attack to make a fake link in the network. The eavesdropping attack scenario is depicted in Figure 4 with a linear network topology. The middle attacker host (the attacker in the figure) requires two network interface cards (NICs) and sets up physical links (e.g. wired or wireless) between two switches, that is, switch 1 and switch 3, through these two NICs. To circumvent the detection of existing protection schemes, the attacker needs to disable all host-generated traffic when forwarding LLDP packets. As part of the normal LLDP broadcast, the attacker host receives a LLDP packet from switch 1 by a NIC and then forwards it to switch 3 via another NIC. In this step, the attacker can use a interactive packet crafting tool (e.g. tcpbridge[18]) to transfer the LLDP packet from a NIC to another NIC in the attacker's host. When switch 3 receives the LLDP packet, it then transfers the received LLDP packet to the controller. The controller thinks that switch 1 connects to switch 3 and then updates the link database. An opposite link is created in a similar manner. In other words, the attacker makes a fake link by controlling two NICs connected to two target switches to transfer the LLDP packets between them. In addition, the attacker can make a fake link using two hosts connected to two switches to transfer the LLDP packets between them via a private connection, for example, a tunnel or a direct connection.

After installing a fake link, the attacker can launch an eavesdropping attack between host 1 and host 3 because the fake link can interrupt the operation of the shortest-path routing service in the controller, which means this service will set up a direct traffic flow from switch 1 to switch 3 via the fake link. Hence, when host 1 communicates with host 3, the attacker can capture all network traffic, as shown in Figure 5, where an attacker captures the traffic by h2-eth0 NIC between host 1 (10.0.0.1) and host 3 (10.0.0.3).

In general network topology, the attacker can still perform the link spoofing attacks using the similar method if they can connect to two switches through two NICs. For example, as shown in Figure 6, host 1 requires five hops (switches) (s1, s2, s3, s5, s6 or s1, s2, s4, s5, s6) to reach host 5 or host 6. However, during a link spoofing attack, host 2 (under attacker control) will forward the LLDP packets to the partner host 4 (under attacker control) via a private connection (e.g. a tunnel) to inform the controller that there is a direct link between port number 1 of switch 2 and port number 1 of switch 5, and vice versa. The controller updates its topology information and the fake link between switch 2 and switch 5 is created. The traffic from host 1 can be sent to host 5 or host 6 through a new shortest path included the fake link (s1, s2, s5, s6). As a result, the attacker can eavesdrop or modify the traffic coming to attackers' NICs before it reaches the destination. Thus, even for general network topology, the eavesdropping attack can occur in the same way as in simple topology shown in Figure 4.

Hybrid prevention method

In this section, we propose a novel hybrid prevention method to protect the link service against attacks. Furthermore, an experiment is carried out to analyze the effectiveness of the proposed method.

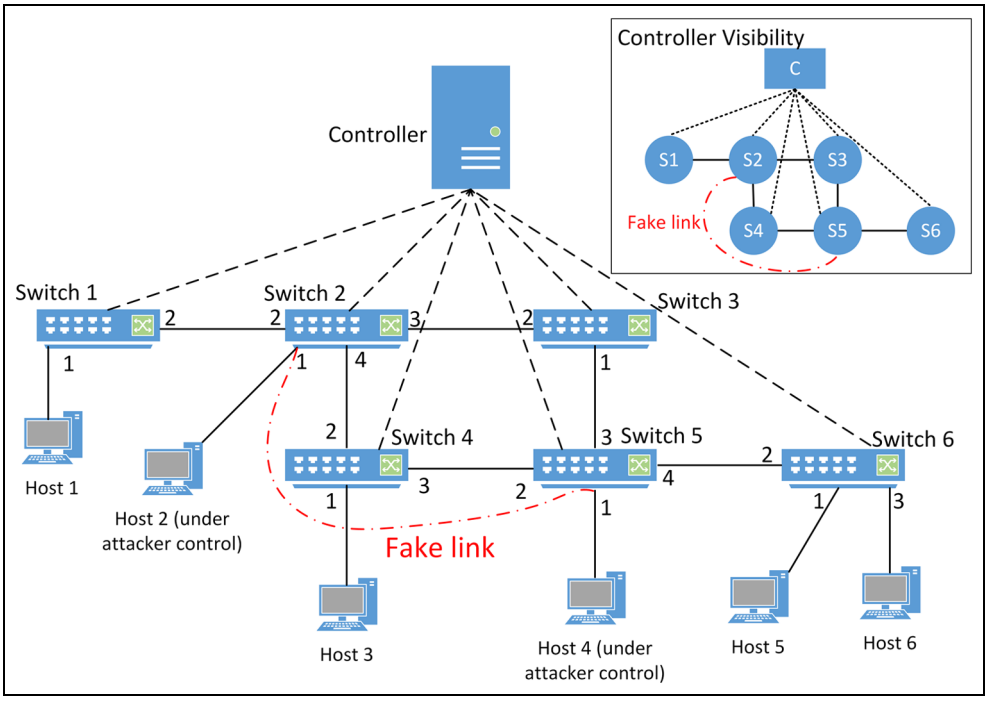

Figure 6. An example of general network topology.

Proposed method

Although the existing link spoofing prevention methods can guarantee the integrity of LLDP packets and ensure the switch ports work correctly, the link service is still vulnerable to eavesdropping attack caused by a fake link as mentioned earlier. To overcome the drawbacks of current solutions, we propose a hybrid countermeasure that not only ensures the integrity of the LLDP packets, but also detects and stops the fake link launched by a powerful attacker through the combination of the following two schemes.

Semi-dynamic signature of the controller. As we mentioned before, the link discovery procedure is periodically performed. For each link discovery cycle, the secret key is changed in the controller's signature of the LLDP packets in proposed method. Hence, it is securer than using a static key, and it is lighter than the dynamic solution for which every LLDP packet has a unique secret key. This effectively prevents an LLDP spoofing attack.

LLDP switch-to-switch (s2s) time. To detect a fake link launched by the LLDP Forwarding attack, the countermeasure also considers the LLDP switch-to-switch (s2s) time, which is the transmission time of LLDP packet between two neighbor switches. The LLDP s2s time can be determined by subtracting the southbound latency of the two switches from the LLDP latency. The southbound latency of a switch is the time for the LLDP to be transmitted between that switch and the controller, while the LLDP latency is the round trip time of the LLDP packet in a link discovery procedure.

In the case of normal links, the LLDP s2s time equals to one-hop transmission time between two switches. In the case of fake links, the LLDP s2s time varies depending on the case of attack. In the first attack case, the attacker uses a single host with two NICs connected to two target switches to transfer the LLDP packets between them. With this case, the LLDP s2s time consists of the transmission time of the LLDP packet from a switch to the attacker, the processing time of the LLDP packet in the attacker's host, and the transmission time of the LLDP packet from the attacker to the other switch. In the second attack case, the attacker uses two hosts connected to two switches to transfer the LLDP packets via a private connection (e.g. a tunnel or a direct connection). With this case, the LLDP s2s time consists of the processing time of the packet in two attackers' hosts and the transmission time of the packet via at least three hops. In either attack case, the LLDP packet needs to be transmitted via more hops, which increases the LLDP s2s time. However, it is the processing time of the LLDP packet in the attackers' host that significantly increases the LLDP s2s time in case of fake links. Although the processing time might vary depending on the manipulation tools used by the attacker as shown in the experiment, the processing time is always much longer than the transmission time of the LLDP packet. Therefore,

Table 1. Pseudocode of the hybrid countermeasure for link spoofing attacks.

Algorithm: Semi-dynamic signature and LLDP s2s time-based Fake Links Detection

```
Input: LLDP packets with signature of controller in a Link Discovery Procedure, LLDP s2s time threshold;
Output: The list of fake links;
FakeLinks = {};
while in Link Discovery Procedure do
   for each LLDP packet
      if CheckSignature() == true:
         continue;
      else: skip this invalid LLDP packet;
   UpdateLinks(); // Update all links of valid LLDP packets
   ChangeSecretKey(); // Change the secret key of controller signature in each Link Discovery Procedure
   for each link in Link Database
      LLDPLatency = GetLLDPLatency();
      SouthBoundLatency = GetSouthBoundLatency();
      LLDP s2s = LLDPLatency - SouthBoundLatency;
      if LLDP s2s > threshold:
         FakeLinks + = {switch source, switch destination} of the link
         TerminateFakeLink(); // Stop this fake link
end;
return FakeLinks;
```

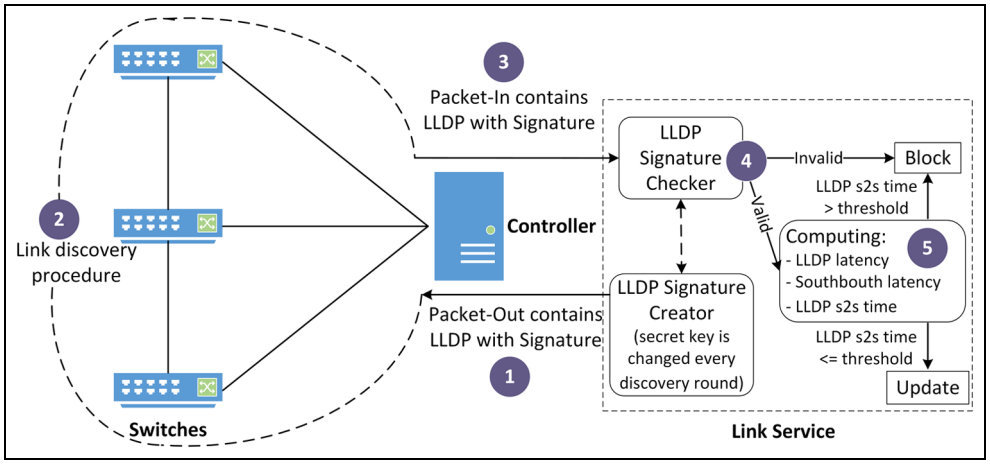

Figure 7. A hybrid countermeasure for link spoofing attacks.

the LLDP s2s time in case of fake links is always much longer than that of normal links, which makes it possible to identify the fake links based on the LLDP s2s time.

Figure 7 illustrates the entire operating process for a new link service and Table 1 shows the algorithm of proposed countermeasure for link spoofing attacks. The controller adds an HMAC-based signature to LLDP packets with a secret key, which is changed in every link discovery round, and verifies the signature when receiving the LLDP packets from switches. If the signature is invalid, the controller simply blocks it. Otherwise, LLDP s2s time is calculated by subtracting the total southbound latency from the LLDP latency. To find the LLDP latency, controller inserts a timestamp in TLV of LLDP packet. Similarly, the controller adds a timestamp in an EchoRequest and then sends to switch. When the controller receives an EchoResponse from the switch, the southbound latency of this switch is calculated. As a result, if the LLDP s2s time exceeds the threshold, we can regard the link as a fake, and block it for a period of time. The threshold can be set to a value that is greater than the maximum LLDP s2s time of the normal links. To get the maximum LLDP s2s time, the network administrator can run the link discovery cycle in a number of times and get the maximum value of LLDP s2s time of the normal links.

As we mentioned earlier, the eavesdropping attack caused by the fake link can also be performed in the

Table 2. LLDP latency of the links.

Link (switch, port to switch, port)	LLDP latency (ms)
sw1, 2 –> sw2, 2	42
sw2, 2 –> sw1, 2	27
sw2, 3 –> sw3, 2	30
sw3, 2 –> sw2, 3	21
sw1, 1 –> sw3, 1 (2 hosts)	4227
sw3, 1 –> sw1, 1 (2 hosts)	3340
sw1, 1 –> sw3, 1 (1 host)	977
sw3, 1 –> sw1, 1 (1 host)	975

LLDP: Link Layer Discovery Protocol.
The bold terms emphasizes the difference of latency between the normal links and the fake links.

general network topology. However, with the proposed countermeasure, the fake link can still be detected in both attack cases even in the general network topology.

Evaluation

Our experiment is carried out using a Mininet emulator tool,[19] which can accurately emulate an SDN and SDIoT network. The proposed countermeasure is implemented on the Floodlight v1.2[11] controller by editing the LinkDiscoveryManager module. In this module, in order to implement the semi-dynamic signature of the controller as a TLV in LLDP packets, we use SHA-256, that is a key-hash message authentication (HMAC) method, along with the secret key that changes in each link discovery procedure. The LLDP s2s time-based fake link detection mechanism is also executed in this module. We also use Open vSwitch,[20] a virtual switch with support for the OpenFlow protocol. The experiment is run on a PC with an Intel Xeon CPU E3-1230 V2 3.30 GHz with 16 GB RAM. For simplicity, we use the linear network topology shown in Figure 4. We perform a link spoofing attack and get the LLDP latency to calculate LLDP s2s time from the response of the Floodlight controller by reviewing its console output. There are two attack cases. In the first case, the attacker uses two hosts connected to two switches to perform the attack through three steps. First, the attacker uses tcpdump[21] to collect the LLDP packet from a target switch through an attacker host. Then, this packet is sent to the other attacker's host. Finally, the attacker uses tcpreplay[22] to resend the LLDP packet to another target switch. In the second case, the attacker performs the attack via an attacker's host with two NICs connected to two switches. The attacker uses tcpbridge[18] to forward the packet between two NICs, and thus, the packet can be transferred between two target switches.

Figure 8 presents the successful launch of the link spoofing attack in a Floodlight controller. The solid frames indicate that non-existent bidirectional links between port number 1 of switch 1 (DPID = 00:00:00:00:00:00:00:01, Port = 1) and port number 1 of switch 3 (DPID = 00:00:00:00:00:00:00:03, Port = 1) are accepted by the controller, and thus, the link spoofing attack is successful. To simplify the demonstration, we assume that the southbound latency of every switch is equal. Therefore, instead of using the LLDP s2s time to detect the fake link, we directly use the LLDP latency. To determine the threshold of LLDP latency, we have run the link discovery procedure 50 times and determined that the maximum LLDP latency of a normal link does not exceed 100 ms, which is the threshold in our experiments. As shown in Table 2, the LLDP latency of the fake links (i.e. 4227, 3340, 977, 975 ms) are much longer than that of the normal links (i.e. 42, 27, 30, 21 ms). Specifically, let us compare the LLDP latency in two cases of attack. The LLDP latency of the fake link in the first case (attack using two hosts) is much longer than that of the second case (attack using a single host). This is because the LLDP latency of the fake links depends on both the number of attacking hosts and the processing time of the manipulating tools used by the attacker. In the first case, the LLDP packet was processed in two hosts, which generally doubles the processing time of the LLDP packet as compared to the second case where the packet was processed in only one host. Furthermore, in the first case, the manipulating tools used by the attacker are tcpdump and tcpreplay while tcpbridge is used in the second case. The tcpdump and tcpreplay take longer processing time than tcpbridge. Consequently, the LLDP latency of the fake link in the first case is at least three times longer than that of the second case. Therefore, the LLDP s2s time in either case of attack is always much longer than that of normal links, which makes it possible to detect the fake links based on the LLDP s2s time. Figure 8 also demonstrates the successful malicious link identification based on LLDP latency with a threshold of 100 ms. The dotted frames indicate that the fake links between port number 1 of switch 1 and port number 1 of switch 3 in the network are detected right after they have been discovered. The eavesdropping attack can be blocked thanks to the fake link detection mechanism.

There are many packet crafting tools that are available publicly. However, in terms of packet capture and replay, the tcpdump and tcpreplay are one of the most common and powerful tools used by the attacker in Linux-based environments. In the first attack case, where the attacker captures the packets into a file from a host and then replays it on the other host, the attacker can use the alternative tools including Wireshark (wireshark.org), hping (hping.org), and Scapy (secdev.org/projects/scapy/). However, like tcpdump and tcpreplay, these tools leverage libpcap library (sourceforge.net/

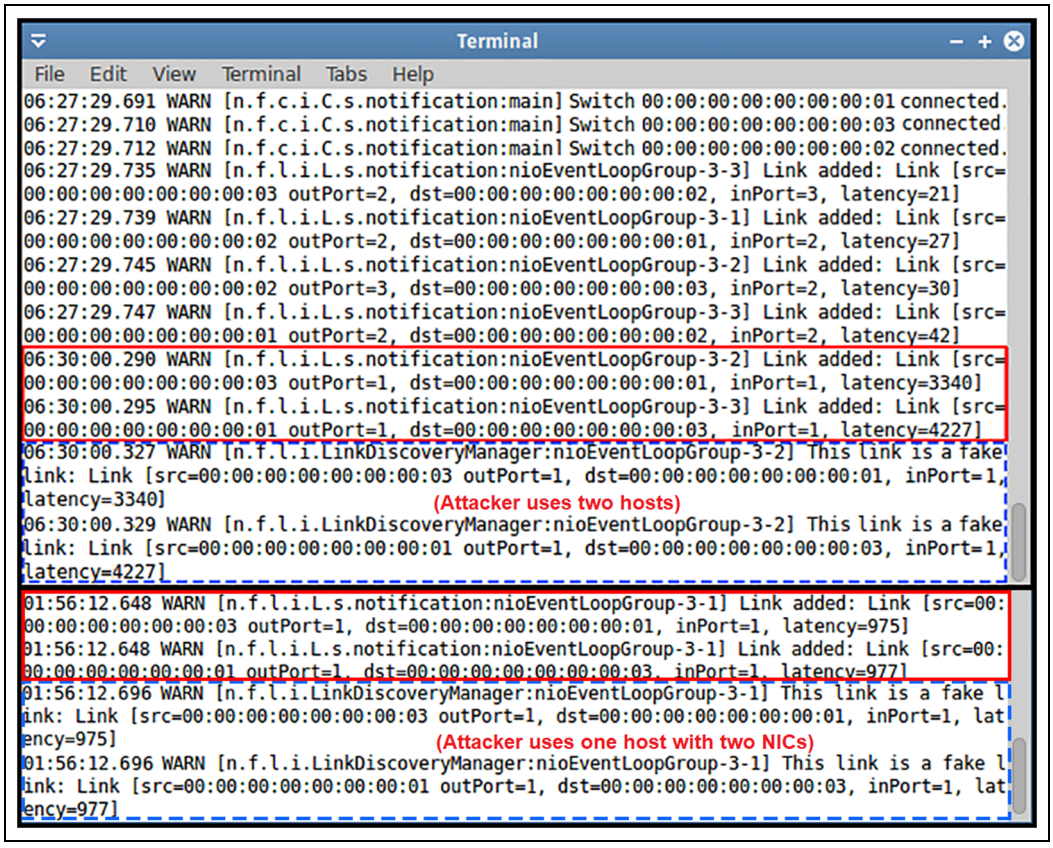

Figure 8. Successful fake link attack by the forwarding method (solid frame) and successful fake link detection (dotted frame).

projects/libpcap/) to capture and replay network traffic. Hence, their performance and packet processing time are almost the same with tcpdump and tcpreplay. In the second attack case, where the attacker needs to bridge network traffic across two NICs in an attacker's host, to the best of our knowledge, tcpbridge is the most suitable tool for this task. Furthermore, the additional steps in packet processing of the tools always add the latency in LLDP packets. Therefore, the proposed countermeasure can distinguish between the fake links and the normal links based on the LLDP latency regardless of the attack tools.

Conclusion

Link service is one of the most important services in SDIoT controller since it provides the information about the links among switches. Without it, the hosts cannot communicate to each other in the network. However, it is found that the link service is also vulnerable to the link spoofing attacks including LLDP spoofing and LLDP Forwarding attacks due to the fact that anyone can manipulate the LLDP packets used in the link service to create fake links. While fake links created in LLDP spoofing attack leads to denial of service due to link disruption, the fake links created in LLDP Forwarding attack can lead to a novel eavesdropping. This work proposes a hybrid countermeasure to prevent link spoofing attacks in SDIoT controller through two schemes. The first scheme relies on adding the semi-dynamic signature of the controller to LLDP packets to guarantee the integrity of the LLDP packets. The second scheme detects and blocks fake links by comparing the LLDP s2s time to a predefined threshold. The experiment demonstrates that the proposed hybrid countermeasure is feasible and efficient since it can prevent all link spoofing attacks in real time.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2017-2017-0-01633) supervised by the IITP (Institute for Information & Communications Technology Promotion).

References

1. Evans D. *The Internet of Things: how the next evolution of the Internet is changing everything*. White Paper, April 2011. San Jose, CA: Cisco.
2. Gubbi J, Buyya R, Marusic S, et al. Internet of Things (IoT): a vision, architectural elements, and future directions. *Future Gener Comput Syst* 2013; 29(7): 1645–1660.
3. Al-Fuqaha A, Guizani M, Mohammadi M, et al. Internet of things: a survey on enabling technologies, protocols, and applications. *IEEE Commun Surv Tuts* 2015; 17(4): 2347–2376.
4. McKeown N. Software-defined networking. *INFOCOM Keynote Talk* 2009; 17(2): 30–32.
5. Kreutz D, Ramos FMV, Verissimo PE, et al. Software-defined networking: a comprehensive survey. *P IEEE* 2015; 103(1): 14–76.
6. Qin Z, Denker G, Giannelli C, et al. A software defined networking architecture for the internet-of-things. In: *Proceedings of IEEE/IFIP network operations and management symposium*, Krakow, 5–9 May 2014, pp.1–9. New York: IEEE.
7. Jararweh Y, Al-Ayyoub M, Darabseh A, et al. SDIoT: a software defined based internet of things framework. *J Ambient Intell Humaniz Comput* 2015; 6(4): 453–461.
8. Liu J, Li Y, Chen M, et al. Software-defined internet of things for smart urban sensing. *IEEE Communications Magazine* 2015; 53(9): 55–63.
9. Wu D, Arkhipov DI, Asmare E, et al. UbiFlow: mobility management in urban-scale software defined IoT. In: *Proceedings of IEEE conference on computer communications*, Kowloon, Hong Kong, 26 April–1 May *2015*, pp.208–216. New York: IEEE.
10. Vilalta R, Mayoral A, Pubill D, et al. End-to-end SDN orchestration of IoT services using an SDN/NFV-enabled edge node. In: *Proceedings of optical fiber communications conference and exhibition*, Anaheim, CA, 20–24 March 2016, pp.1–3. New York: IEEE.
11. Floodlight controller, http://www.projectfloodlight.org/
12. POX controller, https://github.com/noxrepo/pox
13. OpenDaylight controller, https://www.opendaylight.org/
14. McKeown N, Anderson T, Balakrishnan H, et al. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Comp Com* 2008; 38(2): 69–74.
15. Congdon P. Link layer discovery protocol. *RFC* 2922, 2002, https://tools.ietf.org/html/rfc2922
16. Hong S, Xu L, Wang H, et al Poisoning network visibility in software-defined networks: new attacks and countermeasures. In: *Proceedings of network and distributed system security symposium*, San Diego, CA, 8–11 February 2015. Reston, VA: Internet Society.
17. Dhawan M, Poddar R, Mahajan K, et al. SPHINX: detecting security attacks in software-defined networks. In: *Proceedings of network and distributed system security symposium*, San Diego, CA, 8–11 February 2015. Reston, VA: Internet Society.
18. tcpbridge, http://tcpreplay.synfin.net/tcpbridge.html
19. Lantz B, Heller B and McKeown N. A network in a laptop: rapid prototyping for software-defined networks. In: *Proceedings of the 9th ACM SIGCOMM workshop on hot topics in networks*, Monterey, CA, 20–21 October 2010, p.19. New York: ACM.
20. Open vSwitch, http://openvswitch.org/
21. tcpdump, http://www.tcpdump.org
22. tcpreplay, https://github.com/appneta/tcpreplay