An In-depth Look Into SDN Topology Discovery Mechanisms: Novel Attacks and Practical Countermeasures

Eduard Marin
University of Birmingham, UK
imec-COSIC, KU Leuven, Belgium
e.marin@cs.bham.ac.uk

Nicola Buccioli
University of Padua, Italy
nicola.buccioli@icloud.com

Mauro Conti
University of Padua, Italy
conti@math.unipd.it

ABSTRACT

Software-Defined Networking (SDN) is a novel network approach that has revolutionised existent network architectures by decoupling the control plane from the data plane. Researchers have shown that SDN networks are highly vulnerable to security attacks. For instance, adversaries can tamper with the controller's network topology view to hijack the hosts' location or create fake inter-switch links. These attacks can be launched for various purposes, ranging from impersonating hosts to bypassing middleboxes or intercepting network traffic. Several countermeasures have been proposed to mitigate topology attacks but to date there has been no comprehensive analysis of the level of security they offer. A critical analysis is thus an important step towards better understanding the possible limitations of the existing solutions and building stronger defences against topology attacks.

In this paper, we evaluate the actual security of the existing mechanisms for network topology discovery in SDN. Our analysis reveals 6 vulnerabilities in the state-of-the-art countermeasures against topology attacks: *TopoGuard*, *TopoGuard+*, *SPV* and *SecureBinder*. We show that these vulnerabilities can be exploited in practice to manipulate the network topology view at the controller. Furthermore, we present 2 novel topology attacks, called *Topology Freezing* and *Reverse Loop*, that exploit vulnerabilities in the widely used Floodlight controller. We responsibly disclosed these vulnerabilities to Floodlight. While we show that it is difficult to fully eradicate these attacks, we propose fixes to mitigate them. In response to our findings, we conclude the paper by detailing practical ways of further improving the existing countermeasures.

KEYWORDS

Software Defined Networking; Security; Topology Attacks and Countermeasures.

ACM Reference Format:

Eduard Marin, Nicola Buccioli, and Mauro Conti. 2019. An In-depth Look Into SDN Topology Discovery Mechanisms: Novel Attacks and Practical Countermeasures. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*, November 11–15, 2019, London, United Kingdom. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3319535.3354194>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions.acm.org).

CCS '19, November 11–15, 2019, London, United Kingdom

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-6747-9/19/11...\$15.00

<https://doi.org/10.1145/3319535.3354194>

1 INTRODUCTION

Software-Defined Networking (SDN) is a new networking paradigm that is gaining momentum as a technology for developing more dynamic, agile and programmable networks in data centre and enterprise environments [25]. SDN proposes to decouple the control logic (i.e., control plane) from the data forwarding functionality of networking devices, i.e., data plane. One of the key benefits of SDN is that it offers centralised control. The SDN paradigm postulates that the network's intelligence resides in a logically centralised controller, allowing the underlying network infrastructure to become simple forwarding devices [52]. The controller maintains several core services that are responsible for conducting critical functions in the network, e.g., routing or network topology discovery. Another important aspect of SDN is its programmability. Through the use of applications, running on top of the controller, and open standard application programming interfaces, the controller can easily reprogram and collect statistics from networking devices. SDN applications are software-based programs designed to perform high-level tasks in the network, such as balancing load, defining access control list rules or monitoring traffic. The controller typically has a northbound and a southbound interface to communicate with the application and data planes, respectively. The OpenFlow communication protocol, standardised by the Open Networking Foundation (ONF) in 2011, is the de facto open-source standard for the southbound interface [28]. The OpenFlow protocol has received significant attention by the research community [23] and is currently being used in real-world SDN networks, such as Google's B4 network [17].

Using OpenFlow, the controller installs flow rules to instruct switches on how to handle packets. Switches contain several flow tables, each with its own set of flow rules. A flow rule consists of three fields: (i) matching criteria, (ii) action (e.g., drop the packet) and (iii) priority. Every time a switch receives a packet whose headers do not match any of its flow rules, the switch encapsulates the packet in an OpenFlow packet_in and forwards it to the controller. The latter then installs a flow rule in the switch through an OpenFlow packet_out. From this point onwards, if the switch receives a packet with identical headers, it uses the flow rule previously cached for as long as the flow rule remains valid.

Despite its benefits, SDN broadens the attack surface and introduces new security challenges. Several researchers have shown that it is possible to launch security attacks at the application, control and data planes [18, 24, 27, 41, 44, 45, 50, 54], while other work has proposed countermeasures for improving the security of SDN networks [5, 35, 37, 38, 47, 51, 53]. Among the proposed attacks, *topology attacks* that aim at poisoning the network topology are one of the most dangerous types. Although topology attacks are

well-known by the network security community [22, 42], the consequences of such attacks in SDN networks can be more severe than in traditional networks [2]. In traditional networks, adversaries can only tamper with the topology of a small fraction of the network by convincing a set of switches/routers of a specific (fake) topology event. Instead, SDN relies on the use of a logically centralised controller with full network visibility. As the controller contains all the network topology information, adversaries can influence any part of the network regardless of their location within the network. To further complicate matters, SDN-enabled switches lack sufficient logic and capabilities to implement traditional countermeasures such as dynamic Address Resolution Protocol (ARP) inspection.

Maintaining a genuine network topology view at the controller is of utmost importance. SDN core services and applications require real-time and accurate topology information to perform their tasks correctly. If adversaries compromise the network topology, they can redirect traffic through compromised machines. This allows bypassing middleboxes or conduct Man-in-The-Middle (MiTM) or Denial-of-Service (DoS) attacks. Furthermore, adversaries can impersonate hosts to receive their traffic. This is especially dangerous in the case of a server that handles a large amount of traffic.

Our contribution

This paper demonstrates that securing the SDN topology discovery mechanisms implies not only to design secure topology defences but also to implement the topology services at the controller correctly. Concretely, the contributions of this work are the following:

- We conduct a systematic security analysis of the state-of-the-art defences against topology attacks (see footnote¹). This resulted in the identification of 6 vulnerabilities in *TopoGuard*, *TopoGuard+*, *Stealthy Probing-Based Verification (SPV)* and *SecureBinder*. We propose and implement attacks against *TopoGuard/TopoGuard+* and provide clear evidence of other attacks against *SPV* and *SecureBinder* (Section 5).
- We discover important security vulnerabilities within the topology services in *Floodlight*, one of the major SDN controllers. Following the principle of responsible disclosure, we notified *Floodlight* about the vulnerabilities we identified. Then we introduce and practically demonstrate two novel attacks, called *Topology Freezing* and *Reverse Loop*, that can severely damage the controller’s view of the network. As fully eliminating these attacks would require major changes in the *Floodlight* controller, we propose practical ways of mitigating such attacks (Section 6).
- Based on our findings, we also discuss possible ways of further hardening the existing topology countermeasures to defend against link fabrication and host location hijacking attacks (Section 7).

¹We contacted the authors of *TopoGuard*, *TopoGuard+*, *SPV* and *SecureBinder* to request the source code of their solutions. They all replied to our emails and answered our questions. Unfortunately, the authors of *SPV* and *SecureBinder* were unable to share their source code with us.

Organisation. The remainder of this paper is organised as follows: Section 2 gives an overview of related work. Section 3 reviews the SDN topology discovery mechanisms and briefly summarises the topology attacks and countermeasures proposed by other researchers. Section 4 shows the laboratory setup we used for our experiments. In Section 5, we analyse the security of *TopoGuard*, *TopoGuard+*, *SPV* and *SecureBinder* and exploit weaknesses in each of them to uncover new topology attacks. In Section 6, we introduce *Topology Freezing* and *Reverse Loop*, two novel topology attacks that leverage weaknesses in the way the *Floodlight* controller implements its topology services. Along with each of these attacks, we propose fixes to mitigate them. Section 7 elaborates on practical ways of further enhancing the existing countermeasures. Section 8 provides concluding remarks.

2 RELATED WORK

Hong et al. [15] and Dhawan et al. [11] were the first to show how adversaries can poison the network topology view at the controller to create fake links between switches (i.e., link fabrication attacks) or impersonate a victim host (i.e., host location hijacking attacks). In response to these attacks, Hong et al. [15] and Dhawan et al. [11] devised *TopoGuard* and *SPHINX*, respectively. *TopoGuard* prevents these attacks by (i) adding an integrity check to topology packets (ii) labelling switch ports to avoid hosts propagating topology packets to the network and (iii) checking pre- and post-conditions (i.e., verifying that a host left the previous network location before moving to the new one). *SPHINX* proposes a general framework for detecting the occurrence of attacks (not only topology attacks) by validating all network updates. To achieve its goal, *SPHINX* constructs a flow graph of observed traffic between each pair of endpoints and compares it with past graphs in order to find anomalies. However, neither *TopoGuard* nor *SPHINX* can thwart sophisticated topology attacks. For example, adversaries can still perform host location hijacking attacks either by spoofing the victim host’s MAC address or by exploiting the time that hosts are in transit, i.e., moving from one network location to another one.

Skowrya et al. found two topology attacks against *TopoGuard* called port amnesia and port probing [40]. They also proposed *Topoguard+*, an extended version of *Topoguard* that additionally checks for suspicious port reset events and tracks the latency of inter-switch links. The latter allows detection of link fabrication attacks by adversaries who relay topology packets using an out-of-band channel. Another approach to detect relay-based link fabrication attacks was proposed by Alimohammadifar et al. [4]. The authors developed a security solution called *SPV* that periodically injects probing packets to the network to find fake inter-switch links. Jero et al. introduced *SecureBinder* [19], a security solution that uses a modified legacy version of the 802.1x authentication protocol to bind together all host network identifiers. However, the previous two solutions can only protect against certain attacks; *SPV* is only suitable for finding fake links whereas *SecureBinder* focuses only on preventing host location hijacking attacks.

This paper extensively analyses the security of *TopoGuard*, *TopoGuard+*, *SPV* and *SecureBinder* and discusses possible research directions to further improve them.

3 BACKGROUND

This section provides the background to understand the current topology discovery mechanisms in SDN. Afterwards, we briefly review the state-of-the-art topology attacks and defences that have been proposed by other researchers.

3.1 Topology discovery mechanisms in SDN

Below, we describe the process for discovering the network topology in SDN. Subsequently, we introduce the Host Tracking Service (HTS) and the Link Discovery Service (LDS), the two main controller core services involved in network topology discovery.

Network topology discovery is the process by which the controller learns about: (i) the network devices (e.g., switches), (ii) the links between switches and (iii) the location of the hosts within the network. The first is achieved when switches establish a TCP connection – ideally with TLS/SSL – and perform the OpenFlow handshake with the controller. All switches have a unique identifier known as DataPath ID (DPID). It is important to note that switches themselves do not support any mechanism to discover links or track hosts. For inter-switch link discovery, the controller relies on the Link Discovery Service (LDS), whereas the Host Tracking Service (HTS) is used for tracking the host’s location. Next, we describe the HTS and the LDS in more detail.

The **Host Tracking Service (HTS)** maintains information about hosts (e.g., MAC and IP addresses) and their location within the network, i.e., the DPID and port number of the switch where the host is connected. To discover the host location, the HTS leverages on OpenFlow packet_in packets triggered when a host sends a packet for which the switch does not have any flow-rule installed. This causes the HTS to create an entry in the host profile table, binding the host identifiers to its current network location. Whenever a host migrates to a new network location, the host profile table is updated following the procedure previously described. Similarly, if a host disconnects from a switch, the latter notifies the controller by sending an OpenFlow packet containing a port-down event. In such a case, the HTS immediately proceeds to remove the corresponding entry in the host profile table.

The **Link Discovery Service (LDS)** discovers and keeps track of the links between switches. In most SDN controllers, the LDS is based on the OpenFlow Discovery Protocol (OFDP) [3]. Figure 1 illustrates the process by which the controller can discover a unidirectional link between two switches (denoted by S1 and S2). First, the controller encapsulates a *Link Layer Discovery Protocol (LLDP)* packet inside an OpenFlow packet_out packet and sends it to S1, which in turn forwards the LLDP packet to S2. Subsequently, S2 encapsulates the LLDP packet in an OpenFlow packet_in and sends it to the controller, allowing the LDS to discover a unidirectional link from S1 to S2. Following the same approach, the controller can also infer whether a reverse link exists, i.e., from S2 to S1. This procedure is performed regularly in order to account for the dynamics of SDN networks. Existing inter-switch links are removed (i) if a switch detects a port disconnection or (ii) if no LLDP packets are received during a certain amount of time.

Figure 1: Procedure to discover a unidirectional link from S1 to S2 using the OpenFlow Discovery Protocol (OFDP).

3.2 Existing topology attacks and defences

In this section, we summarise the existing topology attacks and the defences that have been proposed to mitigate them.

Topology attacks typically assume that adversaries have some knowledge of the network topology and can learn the network identifiers of the victim host(s). These are realistic assumptions. The network topology information can be recovered using standard path tracing tools (e.g., traceroute) or applying reconnaissance techniques (e.g., [41]). Similarly, as SDN networks use the same (insecure) protocols as traditional networks (e.g., ARP or DHCP), it is relatively simple for adversaries to obtain the network identifiers of the victim host(s).

3.2.1 TopoGuard (NDSS’15). Hong et al. identified two topology attacks called *host location hijacking* and *link fabrication* which have received significant attention in the last few years [15]. Hong et al. also introduced TopoGuard, a security solution that impedes adversaries from executing both attacks. TopoGuard considers adversaries who can control one or more hosts, i.e., the controller and the switches are fully trusted. TopoGuard also assumes the use of SSL/TLS to protect the control channel between the controller and each of the switches.

Attacks. In a *host location hijacking* attack, adversaries aim to convince the controller that a victim’s host moved to another network location. For this purpose, adversaries who control one or more hosts can send packets using the network identifiers (e.g., the IP and/or MAC address) of the victim host. This causes the HTS to update the network location information of the victim’s host. This attack can be successfully launched for as long as the victim host remains idle. On the other hand, in a *link fabrication* attack the goal of adversaries is to create fake links between switches. Figure 2 shows several ways for adversaries to create such fake links. For example, adversaries can modify legitimate LLDP packets or even craft valid ones. Another approach consists of relaying LLDP packets between two network locations using either an in-band or an out-of-band channel. In all these cases, the adversary manages to trick the controller into believing that there is a new inter-switch link when the link does not actually exist. All packets that traverse this link will be dropped or intercepted by the adversary.

(a) Link fabrication attack where H2 relays LLDP packets to H1 using an out-of-band channel.

(b) Link fabrication attack where H1 crafts and sends LLDP packets identical to those that originate from S3 port 1.

Figure 2: Link fabrication attack performed using two different methodologies: (2a) relaying LLDP packets over an out-of-band channel (2b) forging LLDP packets. In both cases, the controller believes that there is a unidirectional link from S3 port 1 to S1 port 1.

Countermeasures. Regarding the *host location hijacking* attack, Hong et al. proposed to check pre- and post-conditions before accepting a host migration as valid. The intuition behind this approach is that all genuine host migrations produce a series of events that need to occur sequentially. Essentially, this implies that a host needs to first leave its current network location before being able to connect to a new switch. More specifically, the controller first waits to receive a port-down from the switch where the host is initially connected, and then it checks whether the host is still reachable at the initial network location. Only if these conditions are satisfied, the controller accepts the migration as valid and the host can send packets from its new network location. Despite this being a very simple countermeasure, it enables detection of attacks where a host appears to be in two locations simultaneously.

To prevent *link fabrication* attacks, Hong et al. proposed to (i) protect the integrity of LLDP packets and (ii) avoid hosts' participation in the LLDP propagation process. To ensure LLDP integrity, a controller-signed field – computed over the DPID and the port number of the source switch – is added within LLDP packets. This prevents adversaries from crafting fake LLDP packets. To guarantee that LLDP packets are sent only to switches, a port-labelling strategy was designed to identify which type of device is connected to each switch port. This approach considers three possible states: (i) HOST, (ii) SWITCH or (iii) ANY. HOST means that there is a host connected to the switch port, SWITCH refers to the case where a switch is connected, whereas ANY is used if no device is connected. Initially, all switch ports are labelled as ANY. The port label is updated based on the first type of traffic received by the switch on each port. However, it is also important for their approach to be able to “forget” the port type. Recall that SDN networks are expected to be used in dynamic environments where a host can be unplugged and replaced by a switch (or vice versa). This requirement can be satisfied by resetting the port type to ANY every time a port-down event is detected, i.e., when the host disconnects from the switch. This port-labelling strategy prevents adversaries who control more than one host from relaying LLDP packets, since these packets are only sent to switch ports.

3.2.2 TopoGuard+ (DSN'18). Skowrya et al. presented two new topology attacks called *port amnesia* and *port probing* that can be successfully conducted even in the presence of TopoGuard [40]. Furthermore, Skowrya et al. designed and implemented an extension of TopoGuard, called TopoGuard+, which not only prevents *port amnesia* attacks but also detects link fabrication attacks based on relaying LLDP packets through an out-of-band channel. TopoGuard+ defends against adversaries who control one or several hosts. The controller and the switches are assumed to be trusted.

Attacks. In the *port amnesia* attack, the goal of the adversary is to bypass the port-labelling technique proposed in TopoGuard. Adversaries can disconnect and reconnect the network interfaces of their hosts to reset the switch ports to ANY. This can let the hosts emulate the behaviour of switches to transmit (fake) LLDP packets to the controller.

In the *port probing* attack, the adversary circumvents the mechanisms used in TopoGuard to thwart *host location hijacking* attacks by exploiting the time it takes for a victim's host to migrate to a new network location. This attack leverages the fact that the host's identifiers are not bound to any network location while hosts are in transit. A technique was proposed to stealthily and accurately detect the moment that the victim's host leaves its network location. Even more, the authors demonstrated that a host migration can be maliciously triggered remotely.

Countermeasures. TopoGuard+ extends TopoGuard by including two new modules: (i) the Control Message Monitor (CMM) and (ii) the Link Latency Inspector (LLI). The CMM enables the controller to identify suspicious port-type resets during LLDP propagation. For this purpose, the controller monitors the traffic and raises an alert if port-up or port-down are received while a LLDP packet is in progress. This makes TopoGuard+ resistant to port amnesia attacks. Nevertheless, the CMM module cannot detect link

fabrication attacks that rely on the use of an out-of-band channel. To defend against such attacks, the LLI module is used. This module detects fake links by keeping track of the latencies of the genuine links between switches.

3.2.3 Stealthy Probing-Based Verification (ESORICS'18).

Alimohammadifar et al. presented SPV, a stealthy probing-based verification approach for detecting any type of link fabrication attack [4]. Similarly to most existing work, SPV assumes that the SDN controller is trusted and that the control channels between the SDN controller and the switches are protected. In contrast to other works, SPV considers adversaries who can control not only hosts but also a few switches within the network. The authors assume that adversaries can use a *low-bandwidth* out-of-band channel to create fake inter-switch links. However, the authors acknowledge the fact that SPV cannot defend against adversaries who forward all traffic through the out-of-band channel, since this would create a link that actually resembles genuine links in the network.

Countermeasures. To verify the legitimacy of inter-switch links, SPV relies on the use of probing packets that are indistinguishable from normal traffic. For this, SPV listens to the network traffic and maintains a list of reference packets sent by hosts. This also includes the DPIDs of the switches from where these packets were sent. For validating a link, SPV chooses a reference packet at random from the ones previously stored in the list. To guarantee the security of SPV, the reference packet cannot be a packet that has previously been used by any of the two switches involved in this link. SPV proposes using probing packets where some fields match those of normal traffic and some fields are randomised. Specifically, the probing packet takes the *Ethernet_type* and *Payload length* from the randomly-chosen reference packet while the source and destination MAC/IP addresses are chosen at random. Similarly to the OFDP protocol, SPV sends the probing packet to the sender switch which in turn forwards it to the destination switch. Upon receiving the probing packet, the destination switch sends it back to the controller. The core idea of their approach is simple yet effective; if the probing packet returns to the controller, there exists a link between these switches. Otherwise, SPV concludes that the link is fake and removes it from the network topology view at the controller.

Additionally, a mechanism was designed to handle lost probing packets (e.g., due to link failures). In such a case, SPV generates a new probing packet by fetching the first probing packet and using the LineSweep algorithm [21, 32]. The new probing packet, which is just slightly different from the first one, is then sent to the sender switch following the procedure previously described.

3.2.4 SecureBinder (USENIX'17). Jero et al. discovered an attack called *Persona Hijacking* that takes advantage of the inherent weaknesses in the identifier binding mechanisms in SDN [19]. Besides proposing a very effective and dangerous attack, Jero et al. introduced SecureBinder, a defence mechanism that can be used to defeat host location hijacking attacks, including the port probing attack introduced in TopoGuard+. SecureBinder assumes that adversaries can control one or more hosts. The rest of the network components are considered to be trusted.

Attacks. *Persona Hijacking* comprises two phases: (i) *IP takeover* and (ii) *flow poisoning*. In the *IP takeover* phase, the goal of the adversary is to break the binding between the IP address and the MAC address of the victim's host. The adversary can successfully launch this attack if it convinces the DHCP server to release the victim's IP address so that it can bind its own MAC address to it. The *flow poisoning* phase is needed only when the DHCP server checks if the IP address is in use before assigning it to a new host. This phase consists of all the necessary steps to break the binding between the victim's MAC address and its network location. Essentially, the adversary exploits a flow rule inconsistency on a switch to redirect traffic to itself. This attack can let adversaries fully takeover and become the owner of the victim's identifiers.

Countermeasures. SecureBinder binds together all hosts' identifiers using a modified legacy version of the 802.1x authentication protocol [34] that additionally checks if the hosts MAC addresses are valid, i.e., within the list of authorised hosts. The controller takes the role of the authenticator, allowing the host (i.e., the supplicant) to access the network after authenticating successfully. The authenticator server, which is connected to the controller, contains a database that binds each host's MAC address with its certificate.

In addition, SecureBinder leverages the SDN architecture to ensure that all binding control traffic is sent directly to the controller (instead of being broadcasted to the network). This prevents adversaries from sniffing the control packets exchanged to establish those bindings and allows the controller to perform several cross-layer checks for validating the bindings when they are updated.

4 LABORATORY SETUP

Figure 3: Our hardware SDN network is composed of three switches (i.e., Raspberry Pi 3 Model B) connected with each other through a linear topology, a controller (i.e., Apple MacBook Pro) and several hosts (not shown in the image) connected to the end of the Ethernet cables. All inter-switch and controller-switch links are 100 Mb/s.

In the next sections, we analyse the security of the state-of-the-art topology defences and the topology core services in Floodlight. For this, we have performed experiments in an emulated environment using Mininet 2.3.0 [43] and in a hardware SDN network. Our hardware SDN network, shown in Figure 3, comprises three Raspberry Pi 3 Model B [12] acting as OpenFlow switches, a controller running on an Apple computer and several hosts that are implemented either in other Raspberries or in a fixed Desktop PC using Linux. To allow multiple SDN-enabled switches to communicate with the controller, we used a traditional L2 Ethernet switch whose only function is to forward the OpenFlow packets from the controller to the switches (and vice versa). We chose to use the Open vSwitch [13] version 2.5.5 LTS as a switch², while our controller is based on Floodlight [30]. The choice of Floodlight was motivated by the fact that most existing topology defences are implemented on it. The controller was installed on a 64-bit Ubuntu 14.04 VM with two cores of 2,8 GHz Intel Core i7 and 8GB of RAM.

5 SECURITY ANALYSIS OF THE TOPOLOGY DEFENCES

In this section, we evaluate the security and propose new attacks against TopoGuard, TopoGuard+, SPV and SecureBinder.

5.1 TopoGuard/TopoGuard+

As TopoGuard+ integrates all the security mechanisms used by TopoGuard, we refer to the joint solution as TopoGuard+. One of the main design goals of TopoGuard+ is to preclude any link fabrication attack regardless of its nature. Despite TopoGuard+ mitigating relay-based link fabrication attacks to a large extent, we identified two new vulnerabilities in the *mechanisms to track link latencies* as well as in the *LLDP packet generation*.

To exploit the weaknesses in the mechanisms to track link latencies, we need to overload switches to increase the latency of the inter-switch links. Intuitively, this could be a possible limitation of our attacks since SDN-enabled hardware switches could incorporate mechanisms to defend against overloading by malicious hosts. However, we want to stress that our findings and attacks can be extrapolated to real-world SDN networks for several reasons. First, SDN-enabled hardware switches contain simple CPUs, which restrict their capabilities for parsing and processing packets [10, 46]. Second, SDN-enabled hardware switches have a small flow table space that can only accommodate from hundreds to a few thousand flow rules [10, 20, 26]. For example, a widely used SDN-enabled hardware switch like *Pica8* can only support 8192 flow entries [1]. Likewise, the rate at which flow tables can be updated is limited. As a result, SDN-enabled hardware switches can only handle 100-200 flow rule updates per second [9, 14, 20, 41, 47, 48]. The previous two limitations arise from the fact that SDN-enabled hardware switches achieve wire-speed packet processing using Ternary Content Addressable Memory (TCAM), which is costly and power hungry. Finally, Zhang et al. demonstrated that hosts do not need to directly send packets to switches to overload them [54]. Instead, hosts can trigger the controller into sending a sufficient number of packets to overload switches more effectively.

²Open vSwitch supports OpenFlow 1.4 protocol (and earlier).

5.1.1 Insecure mechanisms to track link latencies. As previously described, TopoGuard+ relies on the LLI module for measuring the latency of the inter-switch links.

In TopoGuard+, LLDP packets contain a fresh encrypted timestamp so that the controller can measure the overall time between sending and receiving an LLDP packet, (i.e., T_{LLDP}). For computing the latency of an inter-switch link (e.g., T_{S1-S2}), the controller subtracts the latencies of the control links (i.e., T_{S1} and T_{S2}) from T_{LLDP} (see Figure 4). The LLI module then compares T_{S1-S2} with a threshold that is determined using an interquartile range of the list of valid latencies (see Algorithm 1). If T_{S1-S2} is within the valid range of latencies, the LLDP packet is processed correctly and T_{S1-S2} is added to the list of valid latencies. Otherwise, TopoGuard+ raises an alert and removes the link if the failure persists over time.

Figure 4: Link Latency Inspector (LLI) module. The latency of the link between S1 and S2, T_{S1-S2} , is obtained as follows: $T_{S1-S2} = T_{LLDP} - T_{S1} - T_{S2}$. T_{LLDP} is the time between sending and receiving an LLDP packet at the controller. T_{S1} and T_{S2} are the control link latencies of S1 and S2, respectively.

Algorithm 1 Procedure to compute the threshold

- 1: **if** delay != 0 **then**
 - 2: $q1 \leftarrow \text{quartile}(\text{latency_list}, 25)$
 - 3: $q3 \leftarrow \text{quartile}(\text{latency_list}, 75)$
 - 4: $\text{interquartile_range} \leftarrow q3 - q1$
 - 5: $\text{threshold} \leftarrow q3 + 3 * \text{interquartile_range}$
 - 6: **if** delay > threshold **then**
 - 7: stop processing LLDP packet
 - 8: **else**
 - 9: add delay to latency_list
-

Below, we present two attacks against the LLI module where adversaries influence the link latencies to remove genuine links or to create fake ones.

Attack 1. We discovered a new attack against TopoGuard+ that allowed us to remove genuine links between switches. Our attack leverages the fact that the controller removes existing links if their latency is above the threshold in a few LLDP rounds.

Without loss of generality, let us describe the proposed attack using the network topology shown in Figure 5a. During normal execution, we observed that the latency of the links between switches was approximately 4 ms (see third line in Figure 5b). Nevertheless, we found a way to increase the latency of the links between switches using H1.

Initially, we conducted a series of experiments with different numbers of packets and time between packet bursts then measured how each strategy affected the latency of the inter-switch links. Based on our experiments, we chose the smallest number of packets that can sufficiently overload S2 to drop its links to other switches. Specifically, we injected bursts of 100 packets every 1 s with spoofed source MAC addresses and the destination MAC address of H2. This forced S2 to constantly request new flow rules to the controller, consuming a significant amount of its resources. Figure 5 shows that TopoGuard+ started to report errors due to high latencies (around 140 ms) and shortly after it began to remove the affected links. Even if TopoGuard+ re-discovers these links after our attack, we observed that this causes the controller to lose all the previous information about them.

Attack 2. Recently, Shrivastava et al. were able to perform a relay-based link fabrication attack against TopoGuard [39]. Yet, this is no surprise since TopoGuard was *not* designed to preclude those attacks. In this paper, we demonstrate that TopoGuard+ is also vulnerable despite the effort of their authors to protect against such attacks. In order to execute our attack, adversaries need to increase the latency threshold until it becomes comparable to the latency of their out-of-band-channel. However, this is not an easy task since the latency threshold (i) depends only on the latencies of the links that were previously marked as valid and (ii) is computed using an interquartile range, which helps finding outliers.

Without loss of generality, let us describe our attack using the network topology shown in Figure 5a. An adversary who controls H1 can send a large number of packets to S2, resulting in an abrupt increase in its resource consumption. Consequently, S2 either processes the LLDP packets very slowly or drops them. Using this method, the adversary cannot increase the latency threshold at her will since the latency of the LLDP packets traversing S2 will fall outside the range of valid latencies. Instead, the adversary should opt for carefully overloading S2 over a longer period of time, increasing the time it takes for S2 to process the LLDP packets only slightly each time. By repeatedly doing so, adversaries can delay LLDP packets in such a way that the latency threshold is gradually increased. As TopoGuard+ uses a single latency threshold for the network, adversaries can mount this attack regardless of their location within the network.

5.1.2 Lack of freshness in LLDP packets. In TopoGuard+, the controller appends a MAC tag – computed over the DPID and the port number of the source switch – to all LLDP packets. While the authors of TopoGuard+ stated that it is essential to protect the integrity of LLDP packets, their approach lacks freshness. This makes it possible to reuse MAC tags to create valid LLDP packets.

(a) Malicious H1 overloads S2.

```

10:31:38.915 INFO LINK delay between sw 3 and 2 is okay. delay:1ms, threshold:9ms
10:31:38.917 INFO LinkLatencyQueue.size: 395
10:31:38.918 INFO LINK delay between sw 1 and 2 is okay. delay:4ms, threshold:9ms
10:31:38.918 INFO LinkLatencyQueue.size: 396
10:31:38.918 INFO LINK delay between sw 2 and 1 is okay. delay:5ms, threshold:9ms
10:31:38.918 INFO LinkLatencyQueue.size: 397
10:31:38.918 INFO LINK delay between sw 2 and 3 is okay. delay:4ms, threshold:9ms
10:31:53.923 INFO LinkLatencyQueue.size: 398
...
10:33:24.303 ERROR Detected suspicious link discovery: abnormal delay during LLDP propagation
10:33:24.303 ERROR LINK sw 2 and 3
10:33:24.303 ERROR Link delay is abnormal. delay:82ms, threshold:9ms
10:33:24.303 ERROR Detected suspicious link discovery: abnormal delay during LLDP propagation
10:33:24.303 ERROR LINK sw 3 and 2
10:33:24.303 ERROR Link delay is abnormal. delay:142ms, threshold:9ms
10:33:24.303 ERROR Detected suspicious link discovery: abnormal delay during LLDP propagation
10:33:24.303 ERROR LINK sw 2 and 1
10:33:24.303 ERROR Link delay is abnormal. delay:145ms, threshold:9ms
10:33:24.304 ERROR Detected suspicious link discovery: abnormal delay during LLDP propagation
10:33:24.304 ERROR LINK sw 1 and 2
10:33:24.304 ERROR Link delay is abnormal. delay:143ms, threshold:9ms
...
10:33:54.081 INFO Inter-switch link detected: Link [src=00:00:00:00:00:02 outPort=4,
dst=00:00:00:00:00:00:01, inPort=1]
10:33:54.081 INFO LinkLatencyQueue.size: 403
10:33:54.081 INFO Inter-switch link detected: Link [src=00:00:00:00:00:00:01 outPort=1,
dst=00:00:00:00:00:00:02, inPort=4]
10:33:54.081 INFO LinkLatencyQueue.size: 404
10:33:54.081 INFO Inter-switch link detected: Link [src=00:00:00:00:00:00:03 outPort=1,
dst=00:00:00:00:00:00:02, inPort=3]
10:33:54.082 INFO LinkLatencyQueue.size: 405
10:33:54.082 INFO Inter-switch link detected: Link [src=00:00:00:00:00:00:02 outPort=3,
dst=00:00:00:00:00:00:03, inPort=1]
10:33:54.088 INFO LinkLatencyQueue.size: 406

```

(b) Floodlight log console. Initially, the latencies of the inter-switch links (i.e., S1-S2 and S2-S3) are valid and hence they are added to the latency list. Subsequently, LLI detects a number of links with higher latencies and stops processing the corresponding LLDP packets. After a few LLDP iterations (approximately 20 s later), the controller removes the links between S1-S2 and S2-S3 and then re-discovers the links as new links.

Figure 5: Attack against the Link Latency Inspector (LLI) module proposed by TopoGuard+. (5a) Network scenario. (5b) Log console.

Attack. We tested the feasibility of this attack using our hardware SDN network comprising two hosts, three switches and a controller running TopoGuard+³ (see Figure 6). Our goal was to create a fake unidirectional link from S3 to S1 using two malicious hosts (H1 and H2) that can communicate over an out-of-band channel. Note that we assume that the ports of the switches where H1 and H2 are connected are initially set to *HOST*.

(a) H2 sends the MAC tag of S3-Port1 inside the LLDP packet to H1 such that the latter can forge valid LLDP packets as if they originated from S3 port 1.

(b) H1 modifies the LLDP packet that it receives from S1. It keeps all the fields excluding the Chassis ID and the DPID, which are easily derivable, and learns the valid MAC tag from H2.

Figure 6: Link fabrication attack against TopoGuard+ where adversaries manage to forge valid LLDP packets.

As in the port amnesia attack proposed in TopoGuard+, the first step of our attack is to disconnect and reconnect the network interfaces of both hosts (e.g., by unplugging the cables). This causes the controller to reset their port type to *ANY*. For our attack to succeed, the resetting needs to be done before an LLDP round starts so that the CMM module does not flag these events as suspicious. Then, H2 waits to receive an LLDP from the controller, which contains a valid MAC for the corresponding DPID and port number of S3, i.e., the source switch. At this point, H2 sends the valid MAC tag to H1 over the out-of-band channel. Due to the lack of freshness, the MAC tag needs to be exchanged only once. Upon learning the MAC, H1 can successfully send LLDP packets as if

they were originating from S3 port 1. This triggers the controller into believing that there is a real unidirectional link from S3 to S1. We want to highlight that, as the process of creating this link does not involve relaying LLDP packets, the LLI module is not capable of detecting our attack.

5.2 Stealthy Probing-Based Verification (SPV)

By conducting a thorough analysis of SPV, we found two weaknesses in the way the probing packets are generated and sent to the switches. These vulnerabilities mainly stem from the fact that network traffic is not completely random in practice. Therefore, any defence that uses probing packets is likely to have similar issues to the ones we identified in SPV.

5.2.1 Insufficient obfuscation in the probing packet. SPV periodically verifies the legitimacy of all inter-switch links by sending probing packets that resemble normal traffic. Yet, probing packet generation is a very fragile task since ideally it requires (i) having a large amount of network traffic sent by hosts and (ii) obfuscating the probing packets at different levels.

Attack. To validate a link while concealing the probing packet from adversaries, SPV should ideally send the probing packets at random times. However, as SPV uses newly discovered links before verifying them, it is likely that one of the first packets sent to this link is a probing packet. Clearly, it is undesirable to use an unverified link for a long time. This observation reduces the randomness of the process and increases the chances that adversaries can identify the probing packets. To further improve their chances, adversaries who control a large set of malicious hosts can inject enough packets in the network to influence the generation of probing packets. These packets can contain Ethernet_type and Payload_size fields that are unique in the network. In both cases, whenever adversaries receive a packet that is likely to be a probing packet, they can simply forward it over the out-of-band channel. As discovered links are used before SPV verifies them, even if adversaries fail to infer some probing packets, they can still have a sufficiently large time window to perform their attacks.

The previous two observations show that it is very difficult to create probing packets that resemble normal traffic regardless of how these are created.

5.2.2 Use of the LineSweep algorithm. Another important weakness of SPV originates from the use of the *LineSweep algorithm* for generating new probing packets when the initial probing packet fails to arrive to the destination. The LineSweep algorithm takes the initial probing packet and slightly modifies it such that the new probing packet is not completely different than the initial one. Concretely, it chooses new IP and MAC addresses that are adjacent to the ones used in the first probing packet, e.g., in the same subnet.

Attack. The use of the LineSweep algorithm can expose useful information to infer what packets are used for validating a link. In a large network with many hosts, the probability of two hosts connected to the same switch having adjacent IP and MAC addresses is very low. Following this reasoning, adversaries can initially drop the traffic they receive – including the first probing packet – and wait until the second probing packet is sent. As the subsequent

³We downloaded the source code of TopoGuard+ from its public repository in Git [49] on Oct 2nd, 2018.

probing packets will have almost identical IP and MAC addresses, adversaries can identify and relay these probing packets using their out-of-band channel. This way, adversaries can extend the time window the controller uses this link for routing purposes before it is removed from the network.

5.3 SecureBinder

SecureBinder thwarts host location hijacking attacks using a slightly different version of a legacy 802.1x authentication protocol. We acknowledge that SecureBinder clearly enhances the security of the actual binding mechanisms in SDN. Nevertheless, we identified two potential weaknesses in the way the modified legacy version of the 802.1x authentication protocol is used.

5.3.1 Disconnect the good and connect the bad. SecureBinder is an extension of a standard legacy 802.1x authentication protocol that allows hosts to authenticate to the network. Before a host is authenticated, the switch port where the host is connected accepts only “authentication traffic” (i.e., 802.1x frames). In the 802.1x protocol used by SecureBinder, hosts are authenticated only once each time they change their location in the network (instead of in every packet). The choice of extending a legacy 802.1x protocol was motivated by the fact that per-message authentication would incur a large overhead in the controller. However, we observed that this decision comes with important security implications.

Our hypothesis was that, if adversaries can connect a (malicious) host to the network location where the victim’s host is located without triggering a port-down, they can bypass SecureBinder and join the network without needing to authenticate themselves to the authentication server. To force the victim’s host to move to another network location, adversaries could follow an approach similar to the one used in the port amnesia attack [40]. Crucial to the proposed attack is that during the authentication procedure the host does not establish any cryptographic session key with either the controller or the switch. As a result, the packets transmitted by hosts after completing the authentication protocol are neither encrypted nor authenticated.

According to SecureBinder, it should be impossible to disconnect and connect a host without triggering a port-down and a port-up event even when the adversary has *specialised equipment* and *physical access* to the devices. We envision a scenario where the physical hosts can contain several Virtual Machines (VMs) and a virtual switch for routing the packets to/from each VM. In such a case, the virtual switch (inside the physical host) should always inform the corresponding physical switch when a VM is (dis)connected so that the VM can authenticate to the controller using SecureBinder. The way this is done depends on how the virtual switch is designed and programmed. However, it is important to note that all the events generated by the VMs should be treated as if they were originating from a physical host connected to a physical switch.

Attack. We conducted several experiments to investigate if it is possible to disconnect a (victim) host from its switch/port and connect a (malicious) host fast enough such that the switch does not notice about this disconnection. According to the IEEE 802.3 standard, if twisted pairs Ethernet connections are used between

switches and hosts, a signalling protocol is used where a link integrity pulse is sent by these devices every 16 ± 8 ms [16]. In other words, a switch infers that a host is no longer connected if it does not receive such a pulse in 24 ms. If adversaries can remotely detect or trigger a (victim) host disconnection and immediately connect a (malicious) host to the switch, they could benefit from the valid ongoing communication session initiated by the victim’s host.

We started our experiments by running the command `ifconfig eth0 down && ifconfig eth0 up` on a Mininet host to measure the average time between disconnecting and re-connecting a network interface. This test resulted in a delay of only 8 ms, which indicates that it could be possible to circumvent SecureBinder in order to add a malicious host in the network. (Recall that switches can only detect that a host is no longer connected after 24 ms). However, we observed that Mininet switches always detect the port disconnection and send a port-down to the controller. In contrast to real OpenFlow switches, Mininet switches do not implement any signalling protocol with the host and thus they do not check the port liveness before notifying the controller about the port-down. In other words, Mininet emulates the disconnection of a host from a switch without considering the delays introduced by the signalling communication protocol between the host and the switch.

Subsequently, we used Wireshark to measure the average time between a port-down and a port-up on the control plane. Similarly to the previous test, this experiment also resulted in a delay of 8 ms. Motivated by the results obtained in Mininet, we tested our hypothesis using our hardware SDN network with two Raspberry Pi 3 acting as a switch and host, respectively. However, in this case the previous commands took 68 ms, whereas the average time between the port-down and port-up was around 1630 ms on average.

While our preliminary results indicate that SecureBinder is capable of detecting our attack, we suggest that topology defences should not base their security solely on the fact that port-down and port-up events are all genuine (i.e., they are generated if and only if a host really (dis)connects from/to a switch). As a future work, we plan to further investigate how to accelerate the process of disconnecting and reconnecting a network interface.

5.3.2 Insecure low level bindings. SecureBinder provides a weak binding between the hosts’ MAC address and their network location. More specifically, the main limitation of SecureBinder is that it does not bind the authentication traffic to the switch/port where the host is connected.

Attack. Adversaries can intercept the authentication traffic from/to a victim host and replay it in a different network location. If adversaries replay the victim’s authentication traffic to a switch whose path to the controller is faster, they can convince the controller that the victim’s host is at their location. This attack can be possible (i) if the adversary manages to insert an Ethernet hub in the network or (ii) when the victim host and the adversary have two VMs running on the same physical host.

Figure 7 shows a realistic network scenario where this attack could be mounted. This network configuration is known as *in-band SDN* and is widely used in practice [8, 36]. Unlike out-of-band SDN configurations, where all switches can directly communicate to the controller, only a few switches interact with the controller in an in-band SDN configuration.

In an in-band SDN configuration, adversaries can intercept the (plain) authentication traffic from/to H1 (e.g., using an Ethernet hub) and replay it to S2 using H2. If the link between S1 and S2 is slower than the path between S2 and the controller, the authentication traffic sent by the adversary using H2 is received first. This causes the controller to believe that H1 is connected to S2. To improve the effectiveness of this attack, the adversary could overload S1 to deliberately increase the latency of the link between S1 and S2. The main advantage of our attack is that, if H2 succeeds in convincing the controller that H1 is connected to S2, H2 not only receives all the traffic to H1 but also prevents H1 from receiving its traffic. Even if the genuine authentication traffic sent by H1 is eventually received and validated by the controller, the previously installed flow rules can lead to inconsistencies in the data plane.

Figure 7: Possible attack scenario against SecureBinder using an in-band SDN network.

6 IMPLEMENTATION ATTACKS

In 2018, Nehra et al. showed that most SDN controllers lack security mechanisms to protect the network topology information [29]. Motivated by their study, we manually inspected the source code of the Floodlight controller to find potential ways of tampering with the network topology view at the controller⁴. Our analysis resulted in the identification of two new attacks called *Reverse Loop* and *Topology Freezing*. These attacks do not require the controller or the switches to be compromised, and assume that the control channel between switches and the controller is protected using TLS/SSL.

Although these attacks are specific to the Floodlight controller, these (or similar) attacks are also likely to exist in other major SDN controllers. Most controllers rely on the standard OFDP protocol, each with minor message field variations [29]. In addition, the OFDP protocol is not well defined and lacks security mechanisms to protect packet integrity and confidentiality. This results in ad hoc insecure implementations.

6.1 Reverse Loop

We unveil a new attack – which we call *Reverse Loop* – that exploits a weakness in the way the LDS handles the *LINK-TYPE* field inside the LLDP packets. Before describing our attack, let us first briefly explain what the purpose of the *LINK-TYPE* field is. Suppose that S1 and S2 are connected to each other and the controller does not know yet about the existence of a link between them. Initially, the controller sends an LLDP packet with the *LINK-TYPE* field set to ‘0x01’ to S1 which in turn sends it to S2. Once the controller infers a unidirectional link from S1 to S2, it immediately checks if the reverse link (i.e., from S2 to S1) exists by sending an LLDP packet to S2. However, in this case the *LINK-TYPE* field is set to ‘0x02’ (instead of ‘0x01’). Note that the controller regularly repeats this procedure with all switches to collect information about existing or new inter-switch links.

The core idea behind the *Reverse Loop* attack is to extend the duration of an LLDP round as much as possible to exhaust the controller resources, potentially leading to crashes. To illustrate how the *Reverse Loop* attack works, let us give an example using the network topology shown in Figure 2b. Essentially, the adversary proceeds in the same way as when a fake link is created from S3 to S1. Specifically, the adversary (i.e., H1) starts by sending a maliciously crafted LLDP packet containing the headers associated with S3. The controller then proceeds as expected and sends an LLDP packet with the *LINK-TYPE* field set to ‘0x02’ to S1 to validate if a reverse link exists from S1 to S3. Subsequently, the adversary transmits the LLDP packet with the *LINK TYPE* field set to ‘0x01’. Crucially, we found that re-sending the initial LLDP packet (with the *LINK TYPE* field set to ‘0x01’) triggers the controller into checking if the reverse link exists indefinitely. We then discovered an even more powerful variant of our attack that induces the controller into continuously computing the topology instance. This variant leverages the fact that the LDS re-computes the topology instance every time the latency of a link changes. (Note that from Floodlight version 1.2 onwards, LLDP packets contain a time-stamp that is used to determine the link latency).

To test the practicality and severity of the *Reverse Loop* attack, we conducted a series of experiments where we instructed H1 to send LLDP packets with a slightly modified time-stamp to force the controller into recomputing the topology instance. We tested our attack in Mininet using a tree topology with depth 4 and fanout 3, comprising 81 hosts and 40 switches. Figure 8 illustrates the resource consumption of the Floodlight process inside the controller before and during the attack. To measure the CPU performance, we used the `top` command on the Linux shell. This test demonstrates that computing the topology instance is a very demanding task that can cause the controller to crash. While we did not evaluate it, the *Reverse Loop* also results in more LLDP packets being sent by the controller, which can negatively impact the network bandwidth.

We highlight that the *Reverse Loop* attack can facilitate the creation of fake links using an out-of-band channel when TopoGuard+ is deployed. As the controller always sends an LLDP packet to check for the reverse link whenever it receives a *valid* LLDP packet, our attack can indicate whether the LLI module accepted the delayed LLDP packet. Recall that the controller checks for the reverse link only if the latency of the LLDP packet is within the valid range.

⁴We downloaded the Floodlight controller version 1.2 from the official Git repository on Sept 5, 2018 [31].

Figure 8: CPU usage of the Floodlight controller during the Reverse Loop Attack. In this example, the attack starts after approximately 60 s.

The fundamental reason this attack is possible is because LLDP packets lack integrity protection that guarantees that they have not been tampered with, and originate from the sender switch. Protecting the integrity of LLDP packets would mitigate such attacks to a large extent, but this would require major changes in the Floodlight controller. Additionally, we argue that the use of an anomaly detection system could also help to keep track of the number of times a certain LLDP packet is sent.

6.2 Topology Freezing

Our second attack – which we call *Topology Freezing* – affects the module responsible for computing the topology instance. It can be launched to “freeze” the current topology instance, preventing the controller from updating part of the network topology view.

Essentially, our attack is based on the following observation: When two links are created that originate from the same “origin” switch/port (e.g., S1 port 1) but end in two different network locations (e.g., S4 port 1 and S5 port 1), the LDS accepts both links and treats the “multi-link” port as a broadcast domain port (see Figure 9). As a result, the LDS removes the “origin” port and its links from the topology graph construction. Yet, we made the crucial observation that these links are still being used by the controller to compute the shortest path (using an outdated topology instance), causing systematic runtime exceptions.

To show the consequences of performing our attack, we implemented it in Mininet using the network topology shown in Figure 9. As we configured the network so that the shortest path between two hosts is determined based on the number of hops, H3 and H6 initially communicate through Link A. (Note that our attack can be executed regardless of the metric used to compute the shortest path between a pair of hosts). To execute our attack, we used H4 and H5 to create two fake links between (i) S1-Port1 and S4-Port1 and (ii) S1-Port1 and S5-Port1. From this moment onwards and for as long as S1-Port1 contained two links, the controller was unable to fully update the topology instance and the forwarding module, as shown in Figure 10.

Figure 9: Topology Freezing Attack. H4 and H5 can successfully create two fake links originating from S1 Port1.

```
ERROR [n.ft.TopologyManager] Error in topology instance task thread
java.lang.NullPointerException: null
    at net.fl[...].topology.TopologyInstance.dijkstra(TopologyInstance.java:624)
    at net.fl[...].topology.TopologyInstance.yens(TopologyInstance.java:1068)
    at net.fl[...].topology.TopologyInstance.computeOrderedPaths(TopologyInstance.java:790)
    at net.fl[...].topology.TopologyInstance.compute(TopologyInstance.java:168)
    at net.fl[...].topology.TopologyManager.createNewInstance(TopologyManager.java:1015)
    at net.fl[...].topology.TopologyManager.updateTopology(TopologyManager.java:208)
    at net.fl[...].topology.TopologyManagerUpdateTopologyWorker.run(TopologyManager.java:179)
    at net.fl[...].core.util.SingletonTaskSingletonTaskWorker.run(SingletonTask.java:69)
    at java.util.concurrent.ExecutorsRunnableAdapter.call(Executors.java:511)
    at java.util.concurrent.FutureTask.run(FutureTask.java:266)
```

Figure 10: Log file in the Floodlight controller when freezing the network topology.

Subsequently, we removed Link A from the network topology and observed that H6 can no longer receive packets from H3. This is because the controller keeps using an old topology instance and hence uses Link A (which no longer exists) to carry the packets between H3 and H6. It is interesting to see that even if the controller learns about the existence of a new legitimate link between two switches, it cannot update the topology instance and thus cannot use the link. Therefore, adversaries can execute this attack to freeze the network topology and act maliciously without being noticed by the controller. We want to stress that this attack can complement and increase the impact of existing SDN attacks. To maximise its outcome, adversaries need to carefully choose the switch/port from where they launch the attack. Updating the topology instance is done sequentially based on the switches’ DPID, starting from the switch with the lowest DPID. If the multi-link port is created on the switch with the lowest DPID, the rest of the network is no longer updated. The best strategy is therefore to perform the attack from the switch with the lowest DPID. As the Floodlight controller assigns the DPID to each switch based on its MAC address, finding the switch with the lowest DPID is straightforward.

Fixing this vulnerability is not easy. The most intuitive solution to defend against it would be to check if the link exists (e.g., by pinging the other host) before using the link to send real traffic.

7 PRACTICAL COUNTERMEASURES

To defend against topology attacks, SDN networks could simply use a static network configuration similar to traditional networks (e.g., limiting the number of MAC addresses in a switch port) [2]. However, this is a tedious and error-prone approach that does not account for the dynamics of SDN networks. Re-designing the existing topology discovery mechanisms would provide better security and scalability guarantees [6, 33]. Yet, this is not a viable option due to its difficult adoption. In this section, we propose simple yet effective ways of extending the existing countermeasures to defend against the attacks we found in Section 5.

Link fabrication attacks by forging LLDP packets. Protecting the integrity of LLDP packets is crucial to prevent adversaries from sending forged LLDP packets. As shown in Section 5, some of the defences that were proposed to solve this problem lack freshness, allowing replay attacks. For example, TopoGuard proposed to compute a MAC tag over the DPID and port of the source switch using a cryptographic key that is only known to the controller. To provide integrity protection with freshness, Alharbi et al. presented a solution where the cryptographic key is updated in every LLDP round [3]. The main limitation of their approach is that it requires the controller to keep track of the keys used in each LLDP round. To overcome the limitations of the previous solutions, we suggest computing the MAC tag over the DPID, the port and a time-stamp using a single cryptographic key (instead of using a different cryptographic key in each LLDP round)⁵. This can easily be implemented since LLDP packets already contain a time-stamp field for calculating the link latency. Our modification protects against adversaries who (i) tamper with the DPID or the port to conduct link fabrication attacks or (ii) alter the time-stamp field inside the LLDP packets (e.g., to execute a *Reverse Loop* attack).

Link fabrication attacks using out-of-band channels. We note that the security mechanisms implemented in the LLI module can be enhanced. One possible improvement would be for the LLI module to distinguish between LLDP packets received from existing links and those from new links. We suggest that the LLI module allows for higher tolerances in the latencies of the existing links in order to avoid attacks where adversaries aim at removing genuine links. However, even with these improvements, the LLI module can only mitigate these attacks. This is because the LLI module cannot defend against adversaries who use sophisticated hardware to relay packets, as acknowledged by Skowrya et al. [49]. The only possible way to effectively preclude relay attacks is the use of distance bounding protocols [7]. Distance bounding is a security technique through which it is possible to determine an upper-bound on the physical distance between two parties, i.e., the prover and the verifier. By combining physical properties of the communication channel with a cryptographic challenge-response protocol, distance bounding protocols can detect adversaries who use specialised

hardware to relay packets over an out-of-band channel. As a future work, we plan to investigate how to apply the ideas behind distance bounding protocols in SDN networks.

Host location hijacking attacks. Unfortunately, the HTS only relies on unencrypted and unauthenticated OpenFlow packet_in packets to determine the hosts' location in the network. The lack of strong identifier bindings makes it possible to hijack the location of a victim's host. SecureBinder was introduced to protect SDN networks against host location hijacking attacks through a modified version of the 802.1x authentication protocol. While SecureBinder mitigates host location hijacking attacks to a large extent, it is based on a legacy 802.1x protocol that authenticates hosts only once each time they connect to a new switch. This can be sufficient if the controller was always capable of securely inferring a port disconnection from a host. As a future work, we plan to investigate the possibility of developing a middle-ground solution where the controller authenticates a host with a certain probability every time it sends a packet to the network (instead of only once). Several articles have already shown that it might not be suitable for SDN networks to perform security operations in all packets due to scalability issues. Instead, these articles propose to apply security mechanisms only to (i) a set of packets selected at random [5] or (ii) per flow [55].

8 CONCLUSIONS

This paper presented an in depth evaluation of the security of the topology discovery mechanisms in SDN. We conducted a security analysis of the state-of-the-art topology defences which included TopoGuard, TopoGuard+, SPV and SecureBinder. Our analysis revealed that, even if TopoGuard/TopoGuard+ is used, adversaries can still create fake links or remove genuine links. In addition, we provided clear evidence that SPV and SecureBinder are both likely to be vulnerable to attacks. Our work also uncovered weaknesses on the Floodlight controller which enabled us to identify and demonstrate two new topology attacks called *Reverse Loop* and *Topology Freezing*. These attacks can be mounted for various purposes ranging from performing DoS attacks to causing misleading behaviours as a first step before launching a more sophisticated attack. Finally, we elaborated on possible ways of further improving the existing countermeasures to defend against the new attacks we discovered.

ACKNOWLEDGEMENTS

The authors would like to thank our shepherd, Seungwon Shin, and the anonymous reviewers for their constructive and valuable comments that helped us to improve our paper. We also want to thank Allarna Janson for her help and support. During his research stay at the University of Padua, Eduard Marin was supported by a Travel Grant of the Research Foundation - Flanders (FWO file number V413318N). This work was supported in part by the Research Council KU Leuven (C16/15/058), the FWO (SBO project SPITE) and the European Commission (LOCARD project, Grant Agreement no. 832735)

⁵The cryptographic key should be updated regularly to prevent certain types of crypt-analytic attacks.

REFERENCES

- [1] Pica8: Flow scalability per broadcom chipset. <https://docs.pica8.com/display/picos2102cg/Flow+Scalability+per+Broadcom+Chipset> [Online; accessed 7-Dec-2018].
- [2] A. Abdou, P. C. van Oorschot, and T. Wan. Comparative Analysis of Control Plane Security of SDN and Conventional Networks. *IEEE Communications Surveys Tutorials*, pages 3542–3559, 2018.
- [3] T. Alharbi, M. Portmann, and F. Pakzad. The (in)security of Topology Discovery in Software Defined Networks. In *Local Computer Networks (LCN)*, pages 502–505, 2015.
- [4] A. Alimohammadifar, S. Majumdar, T. Madi, Y. Jarraya, M. Pourzandi, L. Wang, and M. Debbabi. Stealthy Probing-Based Verification (SPV): An Active Approach to Defending Software Defined Networks Against Topology Poisoning Attacks. In *European Symposium on Research in Computer Security (ESORICS)*, pages 463–484, 2018.
- [5] M. Ambrosin, M. Conti, F. De Gaspari, and R. Poovendran. Lineswitch: Tackling control plane saturation attacks in software-defined networking. *IEEE/ACM Transactions on Networking (TON)* 25, 2, pages 1206–1219, 2017.
- [6] A. Azzouni, R. Boutaba, T. Mai Trang Nguyen, and G. Pujolle. sOFTDP: Secure and Efficient Topology Discovery Protocol for SDN. In *CoRR*, Vol. abs/1705.04527. arXiv:1705.04527 <http://arxiv.org/abs/1705.04527>
- [7] S. Brands and D. Chaum. Distance-bounding Protocols. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 344–359, 1994.
- [8] J. Cao, Q. Li, R. Xie, K. Sun, G. Gu, M. Xu, and Y. Yang. The CrossPath Attack: Disrupting the SDN Control Channel via Shared Links. In *USENIX Security Symposium*, pages 19–36, 2019.
- [9] H. Chen and T. Benson. The Case for Making Tight Control Plane Latency Guarantees in SDN Switches. In *Symposium on SDN Research (SOSR)*, pages 150–156, 2017.
- [10] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. DevoFlow: Scaling Flow Management for High-performance Networks. In *ACM SIGCOMM Conference*, pages 254–265, 2011.
- [11] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann. SPHINX: Detecting Security Attacks in Software-Defined Networks. In *Network and Distributed System Security Symposium (NDSS)*, pages 8–11, 2015.
- [12] Raspberry Pi Foundation. 2018. Raspberry Pi 3 Model B. <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/> [Online; accessed 7-Dec-2018].
- [13] Linux Foundation. Open vSwitch version 2.5.5 LTS. <http://openvswitch.org/releases/openvswitch-2.5.5.tar.gz> [Online; accessed 7-Dec-2018].
- [14] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. Erran Li, and M. Thottan. Measuring Control Plane Latency in SDN-enabled Switches. In *Symposium on SDN Research (SOSR)*, pages 25:1–25:6, 2015.
- [15] S. Hong, L. Xu, H. Wang, and G. Gu. Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures. In *Network and Distributed System Security Symposium (NDSS)*, pages 8–11, 2015.
- [16] IEEE. 2016. IEEE Standard for Ethernet. *IEEE Std 802.3-2015 (Revision of IEEE Std 802.3-2012)*, 1–4017. <https://doi.org/10.1109/IEEESTD.2016.7428776>
- [17] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Winderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a Globally-deployed Software Defined Wan. In *ACM SIGCOMM Conference*, pages 3–14, 2013.
- [18] S. Jero, X. Bu, C. Nita-Rotaru, H. Okhravi, R. Skowrya, and S. Fahmy. BEADS: Automated Attack Discovery in OpenFlow-Based SDN Systems. In *Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*, pages 311–333, 2017.
- [19] S. Jero, W. Koch, R. Skowrya, H. Okhravi, C. Nita-Rotaru, and D. Bigelow. Identifier Binding Attacks and Defenses in Software-Defined Networks. In *USENIX Security Symposium*, pages 415–432, 2017.
- [20] N. Katta, O. Alipourfard, J. Rexford, and D. Walker. CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks. In *Symposium on SDN Research (SOSR)*, pages 6:1–6:12, 2016.
- [21] H. Kim and H. Ju. Efficient method for inferring a firewall policy. In *Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 1–8, 2011.
- [22] J. King and K. Lauerma. 2014. ARP poisoning attack and mitigation techniques. https://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/white_paper_c11_603839.html [Online; accessed 7-Dec-2018].
- [23] R. Kloti, V. Kotronis, and P. Smith. 2013. OpenFlow: A security analysis. In *IEEE International Conference on Network Protocols (ICNP)*, pages 1–6.
- [24] D. Kreutz, F. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. In *ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 55–60, 2013.
- [25] D. Kreutz, F. Ramos, P. Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. Software-Defined Networking: A Comprehensive Survey. In *ArXiv e-prints*, 2014. <https://doi.org/10.1109/JPROC.2014.2371999>
- [26] A. Lazaris, D. Tahara, X. Huang, E. Li, A. Voellmy, Y. R. Yang, and M. Yu. Tango: Simplifying SDN Control with Automatic Switch Property Inference, Abstraction, and Optimization. In *ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 199–212, 2014.
- [27] S. Lee, C. Yoon, C. Lee, S. Shin, V. Yegneswaran, and P. A. Porras. DELTA: A Security Assessment Framework for Software-Defined Networks. In *Network and Distributed System Security Symposium (NDSS)*, 2017.
- [28] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, pages 69–74, 2008.
- [29] A. Nehra, M. Tripathi, M. Singh Gaur, R. Babu Battula, and C. Lal. TILAK: A token-based prevention approach for topology discovery threats in SDN. *International Journal of Communication Systems*, pages e3781, 2018.
- [30] Big Switch Networks. 2018. Floodlight. <http://www.projectfloodlight.org/floodlight/> [Online; accessed 7-Dec-2018].
- [31] Big Switch Networks. 2018. Floodlight Git repository. <https://github.com/floodlight/floodlight> [Online; accessed 7-Dec-2018].
- [32] P. P. Lin, P. Li, and V. L. Nguyen. Inferring OpenFlow rules by active probing in software-defined networks. In *International Conference on Advanced Communication Technology (ICACT)*, pages 415–420, 2017.
- [33] F. Pakzad, M. Portmann, W. Lum Tan, and J. Indulska. Efficient topology discovery in openflow-based software defined networks. *Computer Communications* 77, pages 52–61, 2016.
- [34] C. Rigney, S. Willens, A. Rubens, and W. Simpson. Remote Authentication Dial In User Service (RADIUS), 2000.
- [35] T. Sasaki, C. Pappas, T. Lee, T. Hoefler, and A. Perrig. SDNsec: Forwarding Accountability for the SDN Data Plane. In *International Conference on Computer Communication and Networks (ICCCN)*, pages 1–10, 2016.
- [36] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester. Automatic bootstrapping of OpenFlow networks. In *IEEE Workshop on Local Metropolitan Area Networks (LANMAN)*, pages 1–6, 2013.
- [37] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson. FRESCO: Modular Composable Security Services for Software-Defined Networks. In *Annual Network and Distributed System Security Symposium (NDSS)*, 2013.
- [38] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *ACM SIGSAC conference on Computer and communications security (CCS)*, pages 413–424, 2013.
- [39] P. Shrivastava, A. Agarwal, and K. Kataoka. Detection of Topology Poisoning by Silent Relay Attacker in SDN. In *Annual International Conference on Mobile Computing and Networking (MobiCom)*, pages 792–794, 2018.
- [40] R. Skowrya, L. Xu, G. Gu, V. Dedhia, T. Hobson, H. Okhravi, and J. Landry. Effective topology tampering attacks and defenses in software-defined networks. In *International Conference on Dependable Systems and Networks (DSN)*, pages 374–385, 2018.
- [41] J. Sonchack, A. Dubey, A. J. Aviv, J. M. Smith, and E. Keller. Timing-based Reconnaissance and Defense in Software-defined Networks. In *Annual Conference on Computer Security Applications (ACSAC)*, pages 89–100, 2016.
- [42] A. Steinhoff, A. Wiesmaier, and R. Araújo. The State of the Art in DNS Spoofing. In *International Conference on Applied Cryptography and Network Security (ACNS)*, 2006.
- [43] Mininet Team. 2018. Mininet. <http://mininet.org> [Online; accessed 7-Dec-2018].
- [44] K. Thimmaraju, L. Schiff, and S. Schmid. Outsmarting network security with SDN teleoperation. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 563–578, 2017.
- [45] B. E. Ujich, S. Jero, A. Edmundson, Q. Wang, R. Skowrya, J. Landry, A. Bates, W. H. Sanders, C. Nita-Rotaru, and H. Okhravi. Cross-App Poisoning in Software-Defined Networking. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 648–663, 2018.
- [46] A. Wang, Y. Guo, F. Hao, T.V. Lakshman, and S. Chen. Scotch: Elastically Scaling Up SDN Control-Plane Using vSwitch Based Overlay. In *ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 403–414, 2014.
- [47] H. Wang, G. Yang, P. Chinprutthiwong, L. Xu, Y. Zhang, and G. Gu. Towards Fine-grained Network Security Forensics and Diagnosis in the SDN Era. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 3–16, 2018.
- [48] H. Xu, Z. Yu, X. Yang Li, C. Qian, L. Huang, and T. Jung. Real-time update with joint optimization of route selection and update scheduling for SDNs. In *International Conference on Network Protocols (ICNP)*, pages 1–10, 2016.
- [49] Lei Xu. TopoGuard+ git repository. https://github.com/xuraylei/TopoGuard_Plus [Online; accessed 7-Dec-2018].
- [50] L. Xu, J. Huang, S. Hong, J. Zhang, and G. Gu. Attacking the brain: Races in the SDN control plane. In *USENIX Security Symposium*, pages 451–468, 2017.
- [51] L. Xue, X. Ma, X. Luo, E. W.W. Chan, T. T.N. Miu, and G. Gu. LinkScope: Towards Detecting Target Link Flooding Attacks. *IEEE Transactions on Information Forensics and Security (TIFS)*, 2018.
- [52] S. Hassas Yeganeh, A. Tootoonchian, and Y. Ganjali. On scalability of software-defined networking. In *IEEE Communications Magazine*, pages 136–141, 2013.
- [53] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable Flow-based Networking with DIFANE. In *ACM SIGCOMM Conference*, pages 351–362, 2010.

- [54] M. Zhang, G. Li, L. Xu, J. Bi, G. Gu, and J. Bai. Control Plane Reflection Attacks in SDNs: New Attacks and Countermeasures. In *Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2018.
- [55] P. Zhang. Towards rule enforcement verification for software defined networks. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1–9, 2017.