Effective Topology Tampering Attacks and Defenses in Software-Defined Networks

Richard Skowyra*, Lei Xu†, Guofei Gu†, Veer Dedhia*, Thomas Hobson*,
Hamed Okhravi*, James Landry*
*MIT Lincoln Laboratory, Lexington, MA
Email: {richard.skowyra, veer.dedhia, thomas.hobson, hamed.okhravi, jwlandry}@ll.mit.edu
†Texas A&M University, College Station, TX
Email: x.rayyle@gmail.com, guofei@cse.tamu.edu

***Abstract*—As Software-Defined Networking has gained increasing prominence, new attacks have been demonstrated which can corrupt the SDN controller's view of network topology. These topology poisoning attacks, most notably host-location hijacking and link fabrication attacks, enable adversaries to impersonate end-hosts or inter-switch links in order to monitor, corrupt, or drop network flows. In response, defenses have been developed to detect such attacks and raise an alert. In this paper, we analyze two such defenses, TopoGuard and Sphinx, and present two new attacks, Port Probing and Port Amnesia, that can successfully bypass them. We then develop and present extensions to TopoGuard to make it resilient to such attacks.**

I. Introduction

Software-Defined Networking (SDN) is a networking paradigm that facilitates network management and administration by providing an interface to control network infrastructure devices (*e.g.,* switches). In this paradigm, the system responsible for making traffic path decisions (the control plane) is separated from the switches responsible for delivering the traffic to the destination (the data plane). The SDN controller is the centralized system that manages the switches, installs forwarding rules, and present an abstract view of the network to the SDN applications.

SDN provides flexibility, manageability, and programmability for network administrators. Complex network management tasks can be implemented using high-level SDN controller abstractions and APIs without the need to deal with low-level network functionalities. Moreover, the centralized model can facilitate the reconstruction of important holistic network properties, such as the network topology, without the need for more sophisticated, distributed algorithms. These advantages have resulted in the relatively rapid adoption of SDN. OpenFlow [28] is one realization of SDN which has been widely implemented in commercial devices [14].

Unfortunately, such flexibility presents new security challenges. The centralized model of SDN makes SDN controllers an attractive target for attackers. Security concerns considered to date include rule consistency [38], operator-error in configuring SDN devices [6], and denial-of-service attacks [41]. Several proposed defenses have focused on mitigating such weaknesses [43], [23], [25]. In addition, many new languages have been proposed to specify SDN rules and configurations, which can facilitate formal verification of such rules [42], [13], [22], [32]. The focus of these language efforts has primarily been on rule consistency or implementation bottlenecks that can be abused by an attacker.

Another class of attacks against SDN are high-level protocol attacks that seek to abuse SDN services to poison the controller's abstraction of the network and its properties. Topology tampering attacks are a prominent example of high-level SDN protocol attacks. In a topology tampering attack, an attacker seeks to poison the controller's view of the network topology, convincing it to believe a false topology is present instead of the actual physical topology. Through link-fabrication attacks, a malicious actor can redirect traffic over forged network links passing through compromised machines, enabling man-in-the-middle or denial-of-service attacks. With host-location hijacking attacks, an attacker can impersonate an end-host and cause traffic bound for the victim to be re-directed to the attacker [16]. By impersonating an important server, for example, an attacker can hijack new and ongoing client sessions.

Two recent defenses, TopoGuard [16] and SPHINX [8], attempt to detect these topology tampering attacks via monitoring of switch-based sensors and packets sent to the SDN controller. TopoGuard relies on behavioral profiling and invariant-checking to detect false network links and spoofed end-hosts, respectively. SPHINX uses an anomaly-detection approach, relying on the inconsistencies in network state at different sensors to detect attacks.

In this paper, we systematically evaluate the effectiveness of these defenses and demonstrate that they can be bypassed generically. We present two novel attacks which we call *Port Amnesia* and *Port Probing*. They can be used as precursors to link fabrication and host location hijacking by attackers to avoid detection. Through port amnesia, an attacker can force a reset in the port type which is used by the defenses to detect anomalous link advertisements. After mounting port amnesia, link fabrication can succeed without being detected by TopoGuard or SPHINX. Through port probing, an attacker sends a fake message bringing a port up at a wrong location after the port goes down for routine migrations or maintenance. We build generic port amnesia and port probing attacks followed by link fabrication and host-location hijacking attacks that succeed undetected even when both TopoGuard and SPHINX are present, without requiring per-defense customization. Through

analysis of these attacks, we argue that not only are these defenses insufficient to prevent topology tampering, but that approaches which rely solely on passive monitoring of network events could be vulnerable to the same attacks. Furthermore, even when passive monitoring can detect certain attacks, we argue that it is often hard for the controller to distinguish between the attacker and the victim. This provides attackers the opportunity to use the defense system itself as a mechanism for denial-of-service attacks.

Using the insights gained from these attacks, we then develop and implement TOPOGUARD+, an extension to TopoGuard built on top of its open-source version [45], which prevents in-band LLDP port amnesia attacks through monitoring of characteristic control plane message patterns generated as part of the attack. We also develop a defense against out-of-band port amneisa attacks. These attacks rely on an attacker having access to a secret channel used to relay LLDP packets outside of the network. The defense takes advantage of this, and detects unavoidable latency additions introduced by processing packets over the external channel. Both defenses are evaluated on a testbed network, and are found to introduce negligible overhead on dataplane flows.

The contributions of this paper are as follows:

- We construct two new topology tampering attacks, port amnesia and port probing, that can bypass state-of-the-art SDN defenses, TopoGuard and SPHINX. We show that these attacks can successfully poison the controller's view of the network topology even when TopoGuard and SPHINX are both deployed, without requiring per-defense customization.
- We implement and evaluate our attacks on an SDN network and measure their parameters and properties.
- We discuss the generality of our attacks and their applicability to passive monitoring defenses.
- We design, implement, and evaluate countermeasures against all forms of Port Amnesia, and argue that active, dynamic defenses will be necessary to mitigate topology tampering attacks in SDN networks.

The remainder of the paper is laid out as follows. In Section II we provide an overview of Software-Defined Networking and the OpenFlow architecture. Classes of topology tampering attacks, TopoGuard, and SPHINX are discussed in Section III. We provide the details of our attacks in Section IV. Analysis of their effectiveness and implementation details are presented in Section V. TOPOGUARD+ is presented in Section VI and evaluated in Section VII. Related work is discussed in Section IX and we conclude in Section X.

II. SDN OVERVIEW

Software-Defined Networking (SDN) is a networking paradigm that separates the control plane from the data plane and provides a logically centralized controller, facilitating faster and easier network monitoring and management [11]. The control plane (*i.e.,* the controller) decides how packets should be handled while the data plane (*i.e.,* the switches) is responsible for the forwarding of packets in accordance with those decisions.

The OpenFlow standard [33] an architecture that relies on a logically centralized, software-based controller, which communicates over a secure control plane to OpenFlow-enabled network switches. An OpenFlow switch routes network dataplane packets based on flow tables, which are ordered lists of rules where each rule consists of a guard, a set of actions to trigger, and a time to expiration. The actions are activated and the packet processed only if the packet's header pattern-matches successfully against the guard for that rule. If a packet does not match any rules in a flow table, it is forwarded to the OpenFlow controller as a `Packet-In` event.

Common open-source controllers include NOX/POX [15], [36], Beacon [10], Floodlight [12], and Ryu [39]. Throughout this paper, without loss of generality, we refer to the Floodlight controller in our discussions of attack details defenses, but our discussions are equally applicable to other controllers as well.

III. EXISTING ATTACKS AND DEFENSES

In this section, we discuss existing attacks and defenses designed to corrupt an OpenFlow controller's internal representation of end-host locations and network topology. These are protocol-based attacks which do not require an attacker to have control-plane access or knowledge of any software vulnerability in the controller or switches. Both attacks were first discussed by Hond, *et al.* [16] and Dhawan, *et al.* [8], who also proposed the TopoGuard and SPHINX defenses, respectively. We summarize each defense below, but refer the readers to the respective papers for a full overview.

A. Attacks

1) Link Fabrication: Modern SDN controllers provide a Link Discovery Service which infers the existence of links between switches. While the specific implementation varies by controller, link discovery in general consists of three phases. The controller first emits crafted Link Layer Discovery Protocol (LLDP) packets to switches via `Packet-Out` events. Next, each switch broadcasts the LLDP packet over all dataplane ports. Finally, all switches that receive an LLDP packet forward it to the controller via a `Packet-In` event, containing the switch identifier and port on which the packet was received. Thus, the controller can infer the existence of a link between two switch ports by observing that an LLDP packet sent by the controller to one switch was sent to the controller by the other.

Link Fabrication attacks corrupt the controller's view of network topology, allowing the attacker to act as a virtual link between two switches. In these attacks, the host captures a legitimate LLDP packet broadcast from a switch and relays it to another point in the network. When the packet is reintroduced to the network, the controller infers the existence of a link from the switch the packet was captured on, to the switch on which the relayed packet was reintroduced. This effectively allows an attacker to act as a man-in-the-middle for all traffic flowing over the virtual link, as well as allowing

denial of service attacks via the creation of network blackholes or forwarding loops.

2) Host Location Hijacking: Many OpenFlow controllers also maintain a Host Tracking Service (HTS) that maps each host's addressing information (*e.g.,* IP and MAC addresses) to a network location defined by the switch and port to which the host is connected. The HTS is kept up-to-date by OpenFlow's default flow rule handling: when a packet is received whose header does not match an existing flow rule, it is forwarded to the controller via a `Packet-In` event. The HTS logs the source address data contained in the packet header and binds (or, in the case of movement, updates) it to the switch's identifier and port at which the packet was originated.

Host Location Hijacking (HLH) attacks rely on corrupting the HTS by spoofing the victim's addressing information from an attacker-controlled network location. This causes the HTS to register a migration from the victim's actual location to the attacker's location, prompting the installation of flow rules that will redirect traffic to the attacker. HLH has some similarities to conventional to ARP spoofing, but differs in two key aspects. First, HLH attacks the MAC-to-Port binding while ARP spoofing attacks the IP-to-MAC binding. Second, HLH uses arbitrary packets while ARP spoofing targets ARP specifically. This makes defenses to ARP attacks ineffective against HLH.

B. TopoGuard

TopoGuard [16] is a recently proposed SDN security component implemented for the Floodlight controller. It provides detection capabilities against both *Host-Location Hijacking* and *Link Fabrication* attacks. TopoGuard consists of two broad components: a behavioral profiler and a policy enforcer. The former infers the type of device connected to a switch port. Devices may be classified as a HOST, a SWITCH, or ANY. All devices begin as type ANY. If the controller receives dataplane traffic whose source address has not been seen before from a port, it is marked as a HOST. If the controller instead receives LLDP packets from a port, it is marked as a SWITCH. On detection of a `Port-Down` event, the type is reset to ANY.

The latter enforces a policy designed to detect each attack. TopoGuard addresses Host Location Hijacking via a *Host Migration Verification* policy that checks pre- and post-conditions whenever migration is detected. The pre-condition is that a host has disconnected from its original location via a `Port-Down` event. The post-condition is that a host must be unreachable at its previous location. This is checked by a ping from the controller. If either is violated, an alert is raised.

TopoGuard addresses Link Fabrication attacks using two techniques: authenticated LLDP packets and Port Property verification. Authenticated LLDP packets are digitally signed by the controller, preventing forgery or corruption by an adversary. Port property verification uses behavioral profiling to raise an alarm when either an LLDP packet is received from a HOST port, or first-hop traffic is received from a SWITCH port.

C. SPHINX

While TopoGuard is designed to detect specific protocol violations, SPHINX is a more generic anomaly detector for OpenFlow networks [8].

SPHINX uses flow graphs to detect anomalous dataplane behavior. Flow graphs track the current and past routes taken between two end-hosts, and are annotated with meta-data (*e.g.,* flow volume) gleaned from switch counters and packet headers.

SPHINX does not explicitly check for either host location hijacking or link fabrication. Rather, it attempts to detect divergences between what the SDN controller intends network state to be, a set of accepted invariants, and what the network state actually is. Specifically, SPHINX assumes that `Flow-Mod` messages emitted by the controller are trustworthy, as are the majority of switches. It compares per-flow counter data maintained by each switch with the expected values gleaned from `Flow-Mod` messages and sanity invariants (*e.g.,* ingress and egress bytes per flow should be equal).

IV. Topology Tampering Attacks

In this section, we present two new attacks, *port amnesia* and *port probing*, that enable traditional link fabrication and host-location hijacking attacks to succeed undetected even when TopoGuard or SPHINX are deployed. Our attacks were conducted within virtual machines running 64-bit Ubuntu 14.04, and are implemented via Bash scripts. We obtained the TopoGuard [16] prototype system from the public Git repository. We were unable to obtain a prototype for SPHINX from the developers. As a surrogate, we implemented checks for all of the invariants specified in Table 3 and Table 4 of the SPHINX paper, as well as all automatically generated flow-specific topological and forwarding constraints. While it is possible that the authors' version of SPHINX includes additional checks, the attacks presented here do not exploit flaws in SPHINX' coverage. They instead rely on implicitly trusted dataplane messages and unavoidable race conditions to cause inconsistent network topologies.

A. Port Amnesia

Modern SDN defenses, such as TopoGuard and SPHINX, attempt to prevent LLDP relaying via behavioral profiling. TopoGuard, for example, uses a simple classifier based on first-seen traffic. A node starts as ANY, and is either marked as a SWITCH if an LLDP packet is seen or as HOST if other first-hop traffic is generated.

However, LLDP exists because network topology may be dynamic: a host could be unplugged and replaced by a switch, and the network should be able to rapidly adapt accordingly. This requires a behavioral profile to be able to be 'forgotten' in response to such a change, in order to avoid false positives. Without access to out-of-band information on network state, these changes in port usage must be inferred from OpenFlow events logged by the controller.

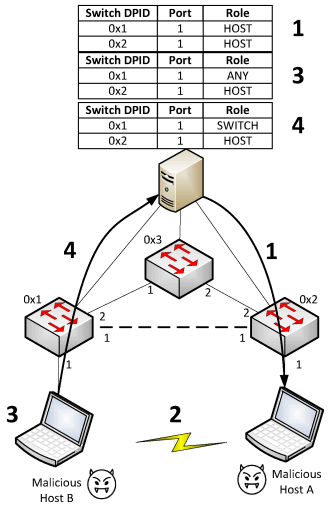

Fig. 1: Port Amnesia Attack

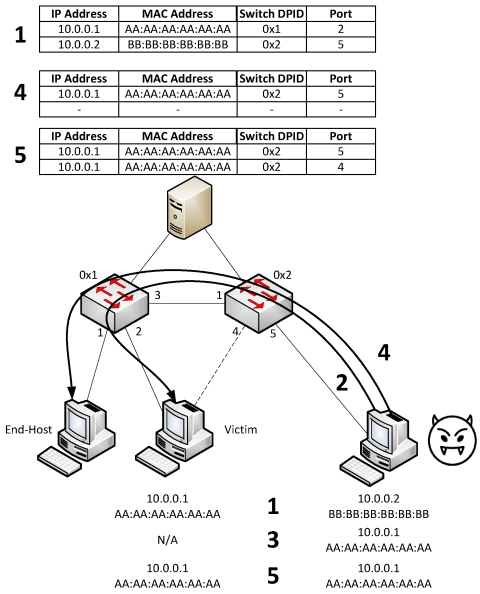

Fig. 2: Port Probing Attack

TopoGuard relies on `Port-Down` events to infer a potential change in port usage. These messages are generated by OpenFlow switches whenever a port is disabled (*e.g.,* by unplugging a cable or disabling a network interface). When the controller receives a `Port-Down`, TopoGuard resets the classification of that port to ANY.

The *port amnesia* attack relies on the fact that behavioral profiling can be cleared by attacker-controllable OpenFlow messages. In TopoGuard an attacker is able to cause their classification to be reset at will by briefly disabling its network port, generating a `Port-Down` event. We name this technique *port amnesia*, since the attacker is able to make the controller forget the previous classification of that port. Note that while `Port-Down` messages are specific to TopoGuard, any defense which both uses a per-port profile, and clears that profile based on OpenFlow messages generated by the dataplane, is vulnerable to the port amnesia attack.

Port amnesia can be used to enable traditional link fabrication attacks while TopoGuard is deployed, as we show using two different scenarios. In Figure 1, two malicious machines are connected to an OpenFlow network with TopoGuard deployed. These hosts communicate out-of-band with one another via a wireless link. (Alternatively, a multi-homed single host could be used.) The behavioral profiles maintained for the attacker's network ports are depicted above the controller.

The attacker first waits until an LLDP packet is sent by the controller to switch 0x2 and received by host A (1). At this point, both links are classified as HOSTS. The attacker sends this packet to B over their shared wireless link (2). It then conducts a port amnesia attack by bringing down the on interface host B, resetting its state to ANY (3). Finally, the interface is brought back up and the LLDP packet is forwarded over host B's link (4). This causes TopoGuard to classify port 1 on 0x2 a SWITCH, and causes the controller to infer a link from port 1 of 0x1 to port 1 of 0x2. The same attack can be performed in order to classify interface A as a SWITCH by waiting for an LLDP packet sent to switch 0x1. Once both interfaces are considered to be switches, traffic can be forwarded through the forged link without causing any alerts to be raised.

Even if an out-of-band communications channel is unavailable, port amnesia can be also used to construct a weaker in-band channel by 'context-switching' between being classified as a HOST and a SWITCH. The attack proceeds as in the out-of-band case, but with an added complication. Once the controller infers a link between switch 0x1 and 0x2, the colluding hosts must be seen as switches while originating packets sent over the inferred link, but also be seen as hosts while sending packets over their secure channel. Otherwise, ports flagged SWITCH would appear to originate first-hop traffic every time the secure channel is used. This context-switching introduces added latency to the fabricated link (as discussed in Section V) and is detectable at the controller (but does not currently raise any alerts) given the number of profile resets required.

B. Port Probing

In addition to LLDP relaying, modern SDN defenses aim to prevent host-location hijacking attempts. TopoGuard relies on checking a pre- and post-condition surrounding end-host movement from port A to B. Prior to movement, a `Port-Down` event must have been received from A. Following movement, the host should no longer be reachable on A. SPHINX does not implement specific checks, but relies on detecting anomalies such as the same identifier (*e.g.,* IP address) being bound to multiple network ports. Both approaches prevent attackers from imitating hosts that are online and have not moved.

Host-location hijacking is still effective against hosts which are *in transit* between locations, however. An unavoidable race condition is introduced after a victim host leaves the network,

but before it has rejoined at another location. While in transit, the victim's identifiers are not bound to any network location, allowing the first host which transmits with those identifiers to 'complete' the move from the controller's point of view.

We introduce *port probing* as a technique that enables attackers to learn, with high precision, when victims begin movement and become susceptible despite the presence of TopoGuard or SPHINX. Depending on the level of stealth desired by an attacker and the amount of precision needed in estimating when a host goes offline, port probing uses a variety of liveness probes to periodically query a victim host. Once those probes indicate that the victim has gone offline, the host-location hijacking attack can be triggered.

Note that we focus on scenarios where the victim has a legitimate movement, *i.e.,* dynamic VM migration in a data-center or planned server maintenance. Thus, we will wait for that vulnerable period to 'complete' the movement. However, a more sophisticated attacker may *induce* such movement, creating a window of opportunity to attack. One such method could be to utilize automatic VM migration based on resource usage. Many hypervisors (*e.g.,* VMware) offer services to automatically migrate VMs between servers when CPU or memory resources become saturated. An attacker could co-locate a host with the target VM and mount a denial-of-service attack against those resources (*e.g.,* cache page dirtying or heavy disk I/O) until the victim was moved by the hypervisor.

Figure 2 schematically depicts a port probing attack to optimally time host-location hijacking against an SDN controller running TopoGuard. The top of the figure displays an abbreviated view of the Host Tracking Service database that binds network identifiers to network locations. The center of the figure depicts an example OpenFlow network. The dotted line between the victim and switch `0x2` is the location to which the victim intends to move. At the bottom, the network identifiers used by the attacker and victim are displayed.

To instantiate the attack, a malicious user joins the network with its own IP and MAC addresses (1). At this point the Host Tracking Service maintains an entry for both the attacker and victim, correctly mapping their identifiers to network location. In order to hijack the victim, the attacker must first acquire its MAC address. We used `arping`, which is a tool to send and receive `arp` packets. From a valid response, we then extract the victim's MAC address.

The attacker then begins the port probing phase. It periodically tests the victim's reachability, waiting until the victim is unreachable to continue the attack (2). The reachability tests we've analyzed, and the precision/stealth tradeoffs between them, are examined in Section IV-B1. Note that at this point the victim has generated a `Port-Down` event in the process of disconnecting, and it is no longer bound to port 2 on switch `0x1`. Thus, TopoGuard has already validated the unfinished move.

Once the victim is known to be offline, the attacker assumes their identity via standard host-location hijacking (3).While packet spoofing is sufficient, we observed that `ifconfig` can reset a network interface card's MAC and IP address rapidly enough (4.1 milliseconds, onaverage) that spoofing via packet header rewriting is unnecessary.

At this point the attacker has assumed the victim's identity, but the Host Tracking Service remains unaware of the change. In order to generate a `Packet-In` event and complete the victim's movement, the attacker must first originate traffic (4). Any dataplane (*e.g.,* ICMP, HTTP, DNS, *etc.*) traffic will suffice. At this point the attacker has successfully hijacked its victim, and new flows with the victim as their destination will instead be directed to the attacker.

Until the victim rejoins the network, the attacker may impersonate the victim with impunity. During this time frame no anomalous behavior is detected by SPHINX, and no TopoGuard policies are violated: by winning the race condition the attacker's hijacking is indistinguishable (to the controller) from a successful move by the victim. At some point, however, the victim will complete its intended move to port 4 of switch `0x2` and rejoin the network (5). Once it originates traffic, the Host Tracking Service will (depending on controller) either have multiple switch ports assigned to the same identifier or will begin oscillating between switch ports based on the last seen packet. This will trigger SPHINX and other anomaly detectors, and may break routing correctness as flows are re-directed between victim and attacker, causing a denial-of-service

Alert Floods The detection of the attack actually provides further opportunities for the attacker to manipulate the network by exploiting the action taken by the controller upon detection. Both SPHINX and TopoGuard raise an alert to prompt intervention by a network operator whenever an anomaly or policy violation is detected, respectively. Note, however, that determination of which end-host is the attacker and which is the victim is left to the operator, and may require follow-up investigation (especially in the context of VM migration, where the future location of the victim may require correlating network and hypervisor logs). Furthermore, this alert does *not* alter network state in any way, and thus does not block the ongoing attack. It merely informs network operators that a suspicious event has occurred.

Attackers can take advantage of this to flood operators with spurious alerts by spoofing arbitrary end-host identifiers from one or more nodes, thus, distracting them from a smaller number of real victims on which the attackers want to maintain persistence.

Conceivably, TopoGuard and SPHINX could be modified to automatically isolate end-hosts after detecting a hijacking attack. In this case, however, the system must infer which of the two hosts is the attacker and which one is legitimate. Without out-of-band data sources, any attempt to distinguish is likely to be imperfect, thus can be leveraged by an attacker to mount denial-of-service attacks by causing the controller to isolate arbitrary victims for some period of time. Even if both hosts are isolated the attacker can still mount a denial-of-service attack, albeit at the cost of losing its own network access. In some network environments, however (*e.g.,* IaaS-type cloud computing), acquiring a new end-host or virtual

TABLE I: Liveness Probe Options

Type	Stealth	Requirements	Timing (ms)
ICMP Ping	Low	None	0.91 ± 0.04
TCP SYN	Medium	Port Known	492.3 ± 1.4
ARP ping	High	Same subnet	133.5 ± 1.6
TCP Idle Scan	Very High	Suitable zombie	1.8 ± 0.1

machine has low cost for the attacker.

1) Liveness Probing: In order to mount a Host Location Hijacking attack without triggering either TopoGuard or SPHINX, an attacker cannot impersonate a victim until that victim has disconnected from the network. To determine when this occurs the attacker must periodically test the victim's liveness via a network probe. These probes, however, may themselves trigger an alert by a network monitoring systems if they are distinct from normal network traffic. To this end, we investigated several liveness probe options, summarized in Table I. All of these are available using `nmap`, a standard network mapping and reconnaissance tool. In Table I, the Stealth column refers to the estimated likelihood of standard Intrusion Detection Systems flagging probe traffic as suspicious, based on recommended rules for the Snort open-source Intrusion Detection System [34]. Note that a number of factors outside of scan type also contribute to the stealthiness of an actual scan, including the scan rate and various evasion techniques like packet fragmentation. The Timing column indicates the time resolution of each probe type by showing, in milliseconds the mean and Standard Deviation of scan time from 1000 scans on our testbed, not including the round-trip time between attacker and victim (which would be invariant over all scan types).

ICMP ping is a standard test for reachability and liveness, but is commonly blocked by firewalls. Even when an attacker and victim are on the same subnet, frequent ICMP Pings are an obvious indicator of network reconnaissance and are likely to be flagged by IDS.

TCP SYN scans detect host reachability by initiating a TCP handshake on a specified port. If the port is open or closed (thus indicating an active host), the scanner will receive a SYN-ACK or RST packet, respectively. If the request times out, the host is assumed to be unreachable. Although TCP traffic itself is ubiquitous and not subject to suspicion, SYN scans are unique in that no data is exchanged over the TCP session. Snort rules tracking zero-data flows may detect this scanning technique. `nmap` can be used to evade such rules, however, by adding decoy data and fake follow-up packets to the established TCP session. TCP SYN scans are also very slow. As can be seen in the table, a single scan takes almost half a second to complete. As shown in Section V-B, this is comparable to the time taken to launch the entire attack.

ARP ping scans broadcast an ARP Request for the target. If the target responds with an ARP Reply, it is assumed to be online. Standard ARP scan detection techniques assume that the scan is being used for network-wide host discovery. They look for large floods of ARP requests for non-existent IPs. Targeted attacks against a known IP address are much harder to detect (with a low false positive rate) due to the ubiquity of ARP requests on Ethernet. In fact, the majority of network IDS, including both Snort and Bro, do not support ARP ping detection [46]. This stealthiness comes at a cost, however. ARP scans are two order of magnitude slower than ICMP pings.

TCP idle scans [9] use a side channel in the TCP implementation to scan a target indirectly. Instead of sending TCP SYN normally, an intermediate 'zombie' host is used. This zombie appears in traffic or IDS logs as the originator of the scan, rather than the actual attacker. This technique is extremely stealthy, but has pre-requisites which are not always available. The attacker must be able to spoof a packet from the zombie, and the zombie must be running a susceptible versions of TCP.

Given these tradeoffs, we chose to use ARP pings in our host location hijacking attacks as attacker and victim already share the same network (by virtue of being administered by the same SDN controller).

2) Downtime Window Duration: When running TopoGuard or SPHINX, Host Location Hijacking attacks that do not rely on alert floods are limited to the downtime window of an offline or migrating host. Specifically, it is limited to the period of time from when the attacker realizes the host is down (by the failure of a liveness probe) to the period of time when TopoGuard or SPHINX raise an alert that multiple network locations are using the same identifier.

For some scenarios, such as target hosts which go offline for patching or maintenance, attackers have a window of minutes to hours within which to impersonate the target. In these cases factors like scan rate (probes per minute) and network round-trip times (10s to 100s of ms for enterprise networks [30]) are minor factors which do not appreciably impact the usable downtime window.

Other scenarios, such as VM live migration, have tighter time constraints. Live migration is a technique used by hypervisors to relocate a VM from one physical machine (and thus network location) to another, while minimizing the disruption of ongoing network sessions. Xen and VMWare, two of the most commonly used hypervisors, have been consistently shown to produce downtime windows on the order of seconds [47], [40], [27], [44]. Round-trip times between VMs sharing a cloud infrastructure are only on the order of hundreds of microseconds [18]. However, in order to detect a migration in progress an attacker needs a high probe rate with minimal timeout before declaring the host unreachable. This may reduce stealth, if a cloud-based IDS is present and configured to monitor for network scans.

V. Results

In this section we present the results of mounting our attacks. We also discuss possible extensions indicated by the evaluation.

A. Port Amnesia Attack

Colluding hosts are set up in the topology presented in Figure 1. We successfully register links with the TopoGuard

controller by relaying LLDP packets over a second 802.11 wireless network . The controller infers a route through our malicious link, allowing arbitrary man-in-the-middle attacks. We leverage the `bridge-util` tools to create Linux bridges between the SDN-connected interface and wireless interface of each malicious host.

TopoGuard will *not* raise an alert when we create our false link, as LLDP traffic is faithfully received and transmitted while first-hop traffic is not generated. The SPHINX system implicitly trusts new links, and only raises an alert when existing links are changed. Furthermore, since all packets sent to the link are faithfully transited, switch-based flow counters do not register a discrepancy in inbound/outbound traffic volume.

The in-band attack scenario requires that the attacker takes a more active role. A primary consideration (given the need to context-switch) is the rate at which LLDP packets are sent by the controller. This is dependent upon the specific SDN controller; Floodlight (and TopoGuard) send LLDP probes out every 15 seconds (this was confirmed using Wireshark). To defeat TopoGuard's protections, the attacker must perform a HOST/SWITCH context switch between sending host-like traffic and sending LLDP packets. More generally, the port amnesia attack must be performed at each context switch.

Using ifconfig to change a network interface (down and then up with IP and MAC addresses) takes 3.25 milliseconds, on average. However, the physical networking layer defines how and when a switch will detect `port-down` messages, which correspond to physical detachment of an interface from a switch. For Ethernet over twisted-pair, the IEEE 802.3 standard defines a link integrity pulse time of 16 ± 8 milliseconds [1]. If no link pulses are received for that interval, a device is considered to be disconnected from the switch. An attacker changing network identifiers faster than 16 milliseconds will *not* trigger a `port-down/up` in the switch. Thus, in order to context switch between HOST and SWITCH, an attacker must wait at least 16 milliseconds between bringing the network interface down and back up. In the worst case, this adds a 16 ms latency to each packet, but allows the attacker to interleave their HOST and SWITCH traffic arbitrarily.

B. Port Probing Attack

The port probing timeline of events is detailed in Figure 3. Green text indicates victim actions, red text indicates attacker actions, and black text indicates SDN controller events. During our attack, we take measurements at the points specified in the timeline. We use `date` to provide the current system time, with microsecond precision. Note that due to the limited window of time in which the attacker can act with impunity, actions taken after the victim down event are most critical to optimize.

The primary goal of port probing is to quickly ascertain that the victim is offline. This affects the total amount of time that the attacker can impersonate the victim without detection. Thus, the attacker wants to know as quickly as possible when the victim disconnects by sending frequent probes with a low timeout value. High probe rates, however, increase the probability of detection by an IDS. Similarly, low timeout values may generate false positives and trigger an alert if the victim has not begun migration.

To determine how much delay an attacker must suffer to remain stealthy, we performed two sets of experiments detailed in Sections V-B1 and V-B2. In the first, we investigated the factors that impact the optimal timeout threshold for attacker probes. In the second, we investigated what scan rates for ARP and TCP SYN scans are sufficient to generate an alert in the Snort IDS using standard, best-practice detection rules.

Once the victim is known (or believed) to be offline, the attacker launches a conventional host-location hijacking attack by changing their network identifiers to those of the victim. The time to do this using `ifconfig` is shown in Figure 4. On average, this takes 9.94ms. Note that the distribution is heavy-tailed, however, with some trials taking as long as 160ms.

Finally, the attacker uses the victim's ID to send and receive traffic. We measure the time that it takes (from the victim going down) for the attacker to reach this success state. The measurement is taken in two places: first, we measure when our interface comes up as the victim. This is shown in Figure 5, and on average takes 478ms. The majority of this time is spent waiting for a probe timeout after the victim has gone offline, as can be seen in Figure 8. Once this step is complete, the attacker successfully originates packets with the victim's identity. Additionally, we measure when the controller acknowledges the attacker as the victim. This is shown in Figure 6. Once complete, traffic sent to the victim is routed to the attacker.

Our attack, from victim down to controller recognition of ID, takes an average of 549ms. As discussed in Section IV-B2, live VM migration downtime windows are on the order of seconds. This leaves the majority of the victim migration window open for attacker actions. For human-mediated movement scenarios (*e.g.,* a server going offline for patching), the attack is virtually instantaneous with respect to the downtime window.

Furthermore, the majority of this time is spent conducting the final reachability probe to the target, and ensuring it has timed out. In ideal network conditions (*i.e.,* minimal variance in RTT), probe timeout values could be reduced and this attack could be launched in tens of milliseconds.

1) Probe Timeouts: Unlike port scans (which return packets to the attacker), liveness scans can only detect a host going offline by the absence of returned packets. The duration to wait for a packet return before deciding the host if offline is referred to as the probe timeout value, and often dominates the scan duration (*e.g.,* the standard ICMP ping defines timeout values in seconds, while a scan may take only a few milliseconds). The attacker has obvious incentive to minimize timeout values, but in doing so risks false positives (*i.e.,* believing the host has gone offline when it has not) due to delayed packet arrivals. Therefore, the attacker must estimate the distribution characterizing the round-trip time (RTT) of packets. As long as the attacker can measure the RTT of packets to the

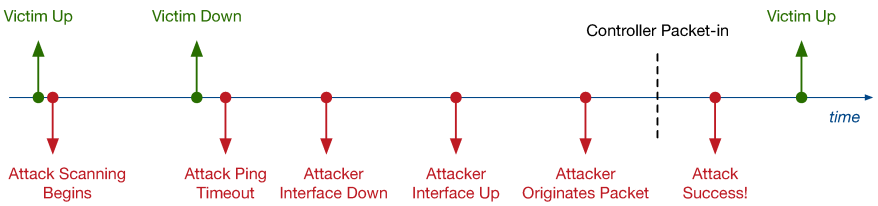

Fig. 3: Host location hijacking timeline (not drawn to scale)

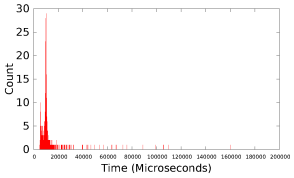

Fig. 4: Distribution of time taken to change network identifiers using ifconfig.

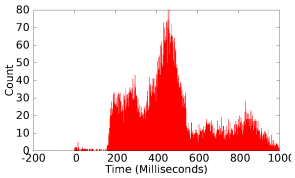

Fig. 5: Distribution of times from Victim Down to Attacker Interface Up. At this point the attacker has claimed the victim's network identity.

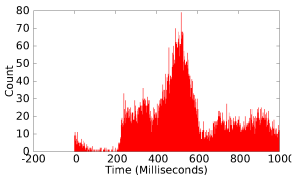

Fig. 6: Distribution of times from Victim Down to Controller Packet-in. At this point packets sent to the victim will be routed to the attacker

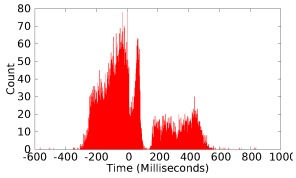

Fig. 7: Distribution of times from Victim Down to start of attacker's final ping.

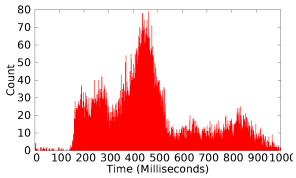

Fig. 8: Distribution of times from Victim Down to Attack Ping Timeout. This is the earliest that the attacker knows the victim has left the network.

target (or a representative surrogate), the probe timeout value given a desired false-positive rate can be easily derived by computing the quantile distribution function for the observed measurements.

For our experiments we modeled network delay as a normal distribution with $\mu = 20ms$ and $\sigma = 5ms$. Due to the timescale of our attack, higher network latencies mask the attack completely and dominate the time taken to mount it. With these network conditions and a desired false positive rate of, 1% we chose a probe timeout of $35ms$. In other environments (*e.g.,* datacenters) a different delay profile will hold, but it remains straightforward for the attacker to calculate this on-the-fly.

Using this timeout value, Figures 7 and 8 show the times to start and end the last ping (which will timeout), relative to the victim going offline. The start of the last ping depends on when the attacker began scanning, the RTT of each packet, and when the victim goes offline. This ping begins, on average, within half a millisecond of the victim going offline, as seen in Figure 7. The end of that ping measures when the attacker believes the victim is offline, relative to when the victim actually goes offline. This includes the timeout value discussed above. On average, the attacker realizes that the victim is offline 12 milliseconds after the event. This is seen in Figure 8.

2) Scan Detection: In addition to choosing a timeout value, the attacker must also choose a scan rate. Higher rates afford the attacker a better resolution, but may also trigger networking intrusion detection systems that are sensitive to port scanning. To determine the highest scan rate that is still relatively covert, we ran TCP SYN and ARP scans at varying rates over a network link monitored by the open-source Snort IDS [34]. Snort's default detection rules, however, do not include rules for either type of scan. We augmented these with a set of rules from Proofpoint, an IDS vendor which publishes best-practice Snort rulesets [37]. The Proofpoint rules detected TCP SYN scans above 2 scans per second.

ARP scans, however, remained undetected. We have been unable to find rules for either Snort or the stateful Bro IDS which can reliably detect ARP scanning. Some literature, in fact, indicates that such scans are not considered malicious scanning at all, as they reveal nothing about the status of

individual ports on the target machine [46]. Given the paucity of options for detecting ARP scans, we elected to send ARP liveness probes at a rate of 54 KBps (1 packet every 50ms).

VI. Defenses

A. Port Probing

Recall that port probing relies on exploiting a fundamental race condition associated with host migration: the first end-host claiming to be the target will be treated as such by the controller. This attack crucially relies on the lack of authentication surrounding network identifiers (*e.g.,* MAC and IP Address). More generally, it relies on the ability to spoof network identifiers and the bindings between them. Conventional network access control such as IEEE 802.1x [7] uses a certificate or other cryptographic credential to verify that a device is authorized before enabled traffic to transit the network port used by the device. Unfortunately, 802.1x does not cryptographically bind network identifiers (*e.g.,* MAC address) to user credentials, and thus is insufficient to prevent port probing attacks. However, recent work on secure identifier binding in SDNs [19] extends the coverage afforded by 802.1x through the entire identifier stack. This would effectively prevent port probing attacks, as the attacker can no longer misleadingly claim to be the victim device without triggering alerts.

B. Port Amnesia

Since recent work has demonstrated an effective defense against port probing attacks, we focus our efforts on preventing port amnesia attacks. To do so, we extend the open-source version of TopoGuard [45] to detect characteristic anomalous interactions corresponding to in-band and out-of-band port amnesia attacks. TopoGuard+ adds two additional modules to TopoGuard: a Control Message Monitor (CMM) module that detects anomalous control plane interactions during LLDP propagation, and a Link Latency Inspector (LLI) module that detects abnormal latencies during LLDP propagation between switches. CMM and LLI prevent in-band and out-of-band port amnesia attacks respectively.

C. Control Message Monitor (CMM)

Recall that in-band port amnesia attacks rely on periodically resetting TopoGuard's behavioral classifier during LLDP propagation in order for end-hosts to appear as switches. The CMM implements a checking procedure to detect this. When an LLDP probe is in progress, receipt by the controller of any of the following message types from a port involved in the LLDP probe (either sender or receiver) will raise an alert. Since the receiver may not be known in advance, the check is retroactively applied to the receiving port for the time between LLDP packet generation and receipt by logging the relevant messages in the controller:

- `Port-Up` or `Port-Down` — This indicates a behavioral profile reset used by in-band port amnesia. Each attacker port must change its status from HOST to SWITCH repeatedly, in order to both relay LLDP traffic and originate data-plane traffic over their secure channel. This necessitates bringing the interface down and up again, which will raise an alert.

The CMM effectively stops in-band port amnesia attacks by installing checks for behavior uniquely characteristic of, and critical to, the attacks. Out-of-band port amnesia attacks, however, do not have such a signature that can be easily monitored. Instead, we address these attacks by focusing on the fact that any out-of-band relaying will necessarily add latency not present in a switch-switch connection.

D. Link Latency Inspector (LLI)

If an attacker has access to an out-of-band channel (*e.g.,* a wireless link) over which packets can be relayed, they do not need to switch between HOST and SWITCH behavior profiles, and thus can evade the CMM. However, utilizing this channel incurs delays due to both signal propagation over the back channel, and encoding/decoding of packets (*e.g.,* converting from Ethernet to 802.1n). [1]

Based on this insight, we implemented a Link Latency Inspector module in TopoGuard+ to measure the latency of switch-internal links during all LLDP propagations, and flag anomalies that may indicate a fabricated link. In order to resolve the latency between two target switches (e.g., sw_1 and sw_2), we measure the overall LLDP propagation time (T_{LLDP}) between them and the delays of control links (T_{SW1} and T_{SW2}). Then, the switch link latency can be estimated as $T_{LLDP} - T_{SW1} - T_{SW2}$.

LLDP Propagation Delay (T_{LLDP}). In order to measure the LLDP propagation delay, we add an extra timestamping function for each LLDP packet during link discovery procedure. In particular, we extend LLDP packets with optional Type-Length-Value (TLV) field that contains the encrypted value of their departure times (by using controller-owned keys). Once the SDN controller receives LLDP packets, it may decrypt the timestamps and compute LLDP propagation delays.

Control Link Latency (T_{SW}). In addition, we adopt echo messages to measure round-trip delays between an SDN controller and a switch. The idea is to utilize packet-out messages to send out a probe message (ICMP ping) to the target switches and set its next-hop (output action) to the controller. As long as the controller receives the probe message, it measures the one-time round-trip delays by computing the elapsed time from sending the probe message to receiving it. Moreover, we take the average of the latest three latency measurements of the control links in order to minimize variance.

Verification of Link Update. The Link Latency Inspector verifies link updates by considering its latency. The insight lies in that the switch link latency may abnormally increase if there exist extra devices or channels to relay LLDP packets. To

[1] This work assumes the attacker is using compromised end-hosts to conduct LLDP relay attacks, which do not have specialized packet-forwarding hardware or extremely high-bandwidth links. For example, a purely hardware-based device which uses point-to-point laser communications is out of scope of this work.

achieve the goal, the LLI utilizes a straightforward application of interquartile range (IQR), which is widely used to find outliers in a set of data. The LLI maintains a fixed size data store for values of the latencies of switch internal links measured from verified LLDP packets and computes lower quartile (Q1), upper quartile (Q3), and interquartile range (IQR, Q3-Q1) upon the data store. When a new LLDP packet arrives in the SDN controller, the LLI inspects the computed latency value with the threshold (Q3 + 3IQR). If any suspicious latency of a switch internal link is found, the Link Latency Inspector raises an alarm to the network administrator and may optionally block the topology update.

VII. Evaluation

In this section, we present a performance and security evaluation of TopoGuard+. We implemented a prototype of TopoGuard+ over the TopoGuard system in the Floodlight controller. In particular, we extended the *LinkManager* application to inspect control messages during LLDP propagation and measure LLDP propagation delays. In addition, we implemented a new application to track real-time controller-switch latencies that are reported *LinkManager*. To evaluate TopoGuard+, we utilized Mininet to created an emulated SDN testbed as shown in Figure 9. All data plane links are configured with 5 milliseconds latency and an out-of-band link between two attacker-compromised hosts with 10-millisecond latency.

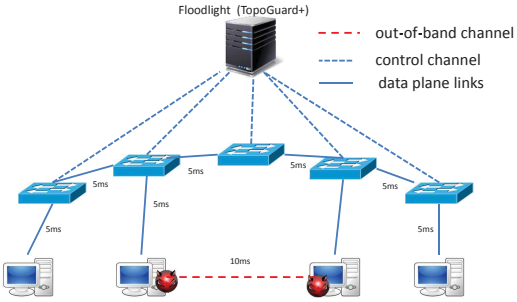

Fig. 9: The evaluation testbed

A. Security Evaluation

In order to calibrate the Link Latency Inspector, we first measured the latency of all four links in Figure 9. Figure 10 records 100 latency measurements of switch link latencies from TopoGuard+. Overall, the average probed latencies for all switch links is around 5 milliseconds, which is consistent with our setup in the mock network environment. The only potential consequences of these lower-latency measurements is a slight decrease of the threshold value for detection of anomalous links, which makes it easier to detect fake links. In addition, the maximum latencies for those links exhibit micro-burst characteristics (*e.g.,*, 12 milliseconds) which may introduce false positives for TopoGuard+. We consider that such jitter can be tolerated in the SDN controllers as we discussed in Section VIII-A.

In order to evaluate the effectiveness of the LLI against out-of-band port amnesia attacks, we also measure the latency threshold distribution for anomalous link discovery, as shown in Figure 11. From the startup of Floodlight controller, we record both measured link latencies and computed threshold for anomalous link detection. Moreover, we control two compromised hosts to build up fake links by utilizing a side channel, as shown in Figure 9, one minute after the bootstrap of the Floodlight controller. The result showcases that TopoGuard+ can effectively locate all fake links. Figure 13 depicts the alert raised for such a link. Note that the plot begins with a set of burst values that dramatically raise the detection threshold. This is due to the bootstrapping of the Floodlight controller, which adds significant extra latency for measurement of link latencies. Once it has reached a steady state, the threshold values converge, which also showcases that our approach can tolerate a small number of anomalous inputs of link latencies.

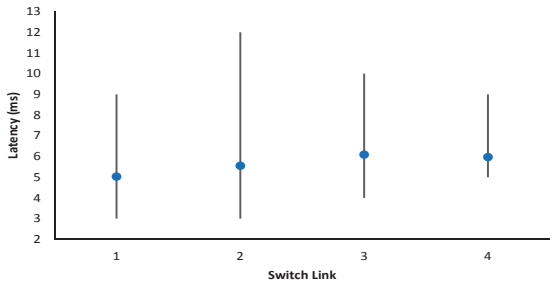

Fig. 10: The latency of switch internal links

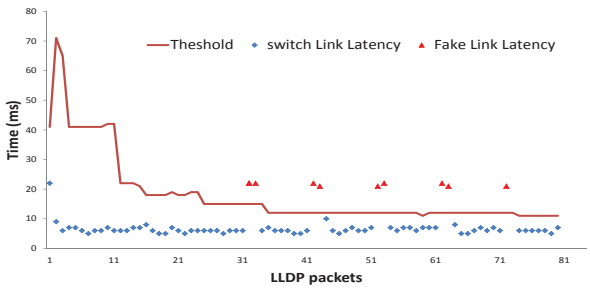

Fig. 11: The threshold distribution with link latencies

In addition, we measured the effectiveness of TopoGuard+ against in-band port amnesia attacks. We launched the aforementioned topology tampering attacks in the testbed environment and confirmed that every port amnesia attack was detected and an alert raised. As shown in Figure 12, TopoGuard+ can successfully detect in-band port amnesia attacks since such attacks must by necessity cause characteristic control plane messages to be generated (*e.g.,*`Port-Down` messages) during LLDP propagation.

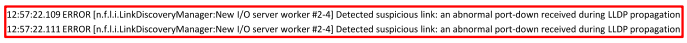

Fig. 12: Alerts from TopoGuard+ for anomalous control messages during LLDP propagation

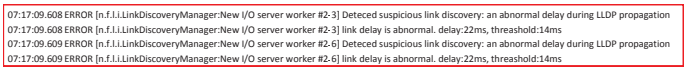

Fig. 13: Alerts from TopoGuard+ for anomalous link latencies from link tampering attacks

B. Performance Evaluation

We also evaluated the performance overhead introduced by TopoGuard+. In this experiment, we leveraged the Java `System.nanoTime` API to measure time stamps of running a program with the precision of 1 nanosecond. The major performance overhead of TopoGuard+ lies in the extra security inspections during processing of LLDP packets (*i.e.,* monitoring control messages and link latencies), and not on any dataplane operations (*e.g.,* packet forwarding). The table II shows TopoGuard+ adds an average of 0.299 milliseconds to the LLDP processing logic in the Floodlight controller. Moreover, TopoGuard+ also introduces 0.134 milliseconds overhead to LLDP packet construction through the addition of an extra encrypted timestamp TLV. The overall results showcase TopoGuard+ add negligible overhead to the Floodlight controller, none of which impact dataplane flows.

TABLE II: TopoGuard+'s Performance Overhead

Function	Overhead introduced by TopoGuard+
LLDP Construction	0.134ms
LLDP Processing	0.299ms

VIII. Discussion

A. False Alerts from the Bursts of Latency

TopoGuard+ may raise false alerts for micro-bursts in link latencies as shown in Section VII. The consequence of the false positives will falsely remove benign switch links from topology view maintained by SDN controller, which may further cause re-computation of routing paths and other topology dependent services. However, we consider that the SDN controller can withstand such fluctuating cases, as the default link timeout value exceeds the LLDP probing interval by a factor of 2-3, as shown in Table III. Thus, a benign switch link will be removed from topology view of the Floodlight controller only if there are multiple bursts of link latencies over 10-35s, and not in response to an isolated event. Moreover, we can also increase the timeout value to greatly decrease the possibility of removal of benign switche links by TopoGuard+.

IX. Related Work

In addition to the attacks discussed in this paper, two other primary types of SDN-specific attacks have been presented in the literature to date: saturation attacks [41], [43] and controller-switch communication [5] attacks.

TABLE III: Link timeout and discovery intervals in various SDN controllers

Controller	Link Discovery Interval	Link Timeout
Floodlight	15s	35s
POX	5s	10s
OpenDaylight	5s	15s

Analogous attacks have been demonstrated in traditional networking protocols. ARP poisoning is the most similar attack to host location hijacking for traditional networks. Similar link fabrication attacks have previously been demonstrated in traditional networking protocols including Open Shortest Path First [31], [20], Optimized Link State Routing Protocol [17], and Spanning Tree Protocol [35].

A number of approaches have been developed that verify that flow rules do not violate a set of invariants or that an intended configuration state is maintained [2], [3], [24]. For instance, NetPlumber [23] and VeriFlow [25] observe OpenFlow messages between the controller and switches and detect if rules would be installed that violate an invariant or pre-defined policy. Such verification approaches have focused on logic errors in rules as opposed to malicious topological manipulation and thus none of the approaches to date detect the TopoMirage attacks.

Other related efforts include the work by Mekky, *et al.* [29] to allow efficient inspection and filtering of higher network layers in SDNs. Kotani and Okabe [26] filter `Packet-In` messages according to some predefined rules to protect the controller. LineSwitch [4] mitigates control plane saturation DoS attacks by applying probabilistic black-listing. Spiffy [21] detects link-flooding DDoS attacks in SDNs by applying rate changes to saturated links. These defenses, although effective against other attacks, do not detect the TopoMirage attacks.

X. Conclusion

We examined recently proposed defenses, TopoGuard and SPHINX, that aim to prevent host location hijacking and link fabrication attacks in SDNs. We presented two new attacks, port amnesia and port probing, that can bypass state-of-the-art topology tampering defenses, analyzed the parameters and properties of these attacks, and demonstrate them against real-world SDN systems. Furthermore, we designed, implemented and evaluated countermeasures against these attacks.

XI. Acknowledgements

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.
This material is based upon work supported by the Department of Defense under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of Defense.

This material is based upon work supported in part by the National Science Foundation (NSF) under Grant no. 1617985, 1642129, 1700544, and 1740791. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

REFERENCES

[1] Ieee 802.3 ethernet standard, 2012.

[2] AL-SHAER, E., AND AL-HAJ, S. Flowchecker: Configuration analysis and verification of federated openflow infrastructures. In *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration* (2010), ACM, pp. 37–44.

[3] AL-SHAER, E., MARRERO, W., EL-ATAWY, A., AND ELBADAWI, K. Network configuration in a box: Towards end-to-end verification of network reachability and security. In *Network Protocols, 2009. ICNP 2009. 17th IEEE International Conference on* (2009), IEEE, pp. 123–132.

[4] AMBROSIN, M., CONTI, M., DE GASPARI, F., AND POOVENDRAN, R. Lineswitch: Efficiently managing switch flow in software-defined networking while effectively tackling dos attacks. In *Proceedings of the 10th AsiaCCS* (2015), ACM, pp. 639–644.

[5] BENTON, K., CAMP, L. J., AND SMALL, C. Openflow vulnerability assessment. In *Proceedings of HotSDN* (2013), ACM, pp. 151–152.

[6] CANINI, M., VENZANO, D., PERESINI, P., KOSTIC, D., REXFORD, J., ET AL. A nice way to test openflow applications. In *NSDI* (2012), vol. 12, pp. 127–140.

[7] CONGDON, P., ABOBA, B., SMITH, A., ZORN, G., AND ROESE, J. IEEE 802.1X remote authentication dial in user service (RADIUS) usage guidelines. RFC 3580 (Informational), Sept. 2003.

[8] DHAWAN, M., PODDAR, R., MAHAJAN, K., AND MANN, V. Sphinx: Detecting security attacks in software-defined networks. In *Proceedings of Network and Distributed Systems Security (NDSS)* (2015).

[9] ENSAFI, R., PARK, J. C., KAPUR, D., AND CRANDALL, J. R. Idle port scanning and non-interference analysis of network protocol stacks using model checking. In *USENIX Security Symposium* (2010), pp. 257–272.

[10] ERICKSON, D. The beacon openflow controller. In *Proceedings of HotSDN* (2013), ACM, pp. 13–18.

[11] FEAMSTER, N., REXFORD, J., AND ZEGURA, E. The road to sdn. *Queue 11*, 12 (2013), 20.

[12] FLOODLIGHT. http://www.projectfloodlight.org/projects/.

[13] FOSTER, N., HARRISON, R., FREEDMAN, M. J., MONSANTO, C., REXFORD, J., STORY, A., AND WALKER, D. Frenetic: A network programming language. In *ACM SIGPLAN Notices* (2011), vol. 46, ACM, pp. 279–291.

[14] FOUNDATION, O. N. Openflow certified product list.

[15] GUDE, N., KOPONEN, T., PETTIT, J., PFAFF, B., CASADO, M., MCKEOWN, N., AND SHENKER, S. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review 38*, 3 (2008), 105–110.

[16] HONG, S., XU, L., WANG, H., AND GU, G. Poisoning network visibility in software-defined networks: New attacks and countermeasures. NDSS.

[17] HU, Y.-C., PERRIG, A., AND JOHNSON, D. B. Wormhole attacks in wireless networks. *Selected Areas in Communications, IEEE Journal on 24*, 2 (2006), 370–380.

[18] JACKSON, K. R., RAMAKRISHNAN, L., MURIKI, K., CANON, S., CHOLIA, S., SHALF, J., WASSERMAN, H. J., AND WRIGHT, N. J. Performance analysis of high performance computing applications on the amazon web services cloud. In *IEEE CloudCom* (2010), IEEE, pp. 159–168.

[19] JERO, S., KOCH, W., SKOWYRA, R., OKHRAVI, H., NITA-ROTARU, C., AND BIGELOW, D. Identifier binding attacks and defenses in software-defined networks. In *26th USENIX Security Symposium* (Vancouver, BC, 2017), pp. 415–432.

[20] JONES, E., AND MOIGNE, O. Ospf security vulnerabilities analysis. *Work in Progress* (2006).

[21] KANG, M. S., GLIGOR, V. D., AND SEKAR, V. Spiffy: Inducing cost-detectability tradeoffs for persistent link-flooding attacks. NDSS.

[22] KATTA, N. P., REXFORD, J., AND WALKER, D. Logic programming for software-defined networks. In *Workshop on Cross-Model Design and Validation (XLDI)* (2012), vol. 412.

[23] KAZEMIAN, P., CHAN, M., ZENG, H., VARGHESE, G., MCKEOWN, N., AND WHYTE, S. Real time network policy checking using header space analysis. In *NSDI* (2013), pp. 99–111.

[24] KAZEMIAN, P., VARGHESE, G., AND MCKEOWN, N. Header space analysis: Static checking for networks. In *NSDI* (2012), pp. 113–126.

[25] KHURSHID, A., ZOU, X., ZHOU, W., CAESAR, M., AND GODFREY, P. B. Veriflow: Verifying network-wide invariants in real time. In *Proceedings of NDSI* (Lombard, IL, 2013), pp. 15–27.

[26] KOTANI, D., AND OKABE, Y. A packet-in message filtering mechanism for protection of control plane in openflow networks. In *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems* (2014), ANCS '14, pp. 29–40.

[27] LIU, H., JIN, H., XU, C.-Z., AND LIAO, X. Performance and energy modeling for live migration of virtual machines. *Cluster computing 16*, 2 (2013), 249–264.

[28] MCKEOWN, N., ANDERSON, T., BALAKRISHNAN, H., PARULKAR, G., PETERSON, L., REXFORD, J., SHENKER, S., AND TURNER, J. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review 38*, 2 (2008), 69–74.

[29] MEKKY, H., HAO, F., MUKHERJEE, S., ZHANG, Z.-L., AND LAKSHMAN, T. Application-aware data plane processing in sdn. In *Proceedings of HotSDN* (2014), pp. 13–18.

[30] MURRAY, D., AND KOZINIEC, T. The state of enterprise network traffic in 2012. In *Communications (APCC), 2012 18th Asia-Pacific Conference on* (2012), IEEE, pp. 179–184.

[31] NAKIBLY, G., KIRSHON, A., GONIKMAN, D., AND BONEH, D. Persistent ospf attacks. In *NDSS* (2012).

[32] NELSON, T., FERGUSON, A. D., SCHEER, M. J., AND KRISHNAMURTHI, S. Tierless programming and reasoning for software-defined networks. *NSDI, Apr* (2014).

[33] OPEN NETWORKING FOUNDATION. *OpenFlow Switch Specification*, 1.4.0 ed., July 2015.

[34] OREBAUGH, A., BILES, S., AND BABBIN, J. *Snort cookbook*. " O'Reilly Media, Inc.", 2005.

[35] ORNAGHI, A., AND VALLERI, M. Man in the middle attacks. In *Blackhat Conference Europe* (2003).

[36] POX. http://www.noxrepo.org/pox/about-pox/.

[37] PROOFPOINT. https://rules.emergingthreats.net/open/snort-2.9.0/rules/emerging-scan.rules, 2016.

[38] REITBLATT, M., FOSTER, N., REXFORD, J., SCHLESINGER, C., AND WALKER, D. Abstractions for network update. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication* (2012), ACM, pp. 323–334.

[39] RYU. http://osrg.github.io/ryu/.

[40] SALFNER, F., TRÖGER, P., AND POLZE, A. Downtime analysis of virtual machine live migration. In *The Fourth International Conference on Dependability (DEPEND 2011). IARIA* (2011), pp. 100–105.

[41] SHIN, S., AND GU, G. Attacking software-defined networks: A first feasibility study. In *Proceedings of HotSDN* (2013), ACM, pp. 165–166.

[42] SHIN, S., PORRAS, P., YEGNESWARAN, V., AND GU, G. A framework for integrating security services into software-defined networks. *Proceedings of ONS 13* (2013).

[43] SHIN, S., YEGNESWARAN, V., PORRAS, P., AND GU, G. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of ACM CCS* (2013), ACM, pp. 413–424.

[44] SVÄRD, P., HUDZIA, B., WALSH, S., TORDSSON, J., AND ELMROTH, E. Principles and performance characteristics of algorithms for live vm migration. *ACM SIGOPS Operating Systems Review 49*, 1 (2015), 142–155.

[45] TOPOGUARD. https://github.com/xuraylei/floodlight_with_topoguard.

[46] TRABELSI, Z., HAYAWI, K., AL BRAIKI, A., AND MATHEW, S. S. *Network attacks and defenses: A hands-on approach*. CRC Press, 2012.

[47] VOORSLUYS, W., BROBERG, J., VENUGOPAL, S., AND BUYYA, R. Cost of virtual machine live migration in clouds: A performance evaluation. In *Cloud Computing*. Springer, 2009, pp. 254–265.