Flow Wars: Systemizing the Attack Surface and Defenses in Software-Defined Networks

Changhoon Yoon, Seungsoo Lee, Heedo Kang, Taejune Park, *Student Member, IEEE,* Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu, *Member, IEEE*

***Abstract*—Emerging Software Defined Network (SDN) stacks have introduced an entirely new attack surface that is exploitable from a wide range of launch points. Through an analysis of the various attack strategies reported in prior work, and through our own efforts to enumerate new and variant attack strategies, we have gained two insights. First, we observe that different SDN controller implementations, developed independently by different groups, seem to manifest common sets of pitfalls and design weakness that enable the extensive set of attacks compiled in this paper. Second, through a principled exploration of the underlying design and implementation weaknesses that enables these attacks, we introduce a taxonomy to offer insight into the common pitfalls that enable SDN stacks to be broken or destabilized when fielded within hostile computing environments. This paper first captures our understanding of the SDN attack surface through a comprehensive survey of existing SDN attack studies, which we extend by enumerating 12 new vectors for SDN abuse. We then organize these vulnerabilities within the well-known CIA (confidentiality, integrity, availability) model, assess the severity of these attacks by replicating them in a physical SDN testbed, and evaluate them against three popular SDN controllers. We also evaluate the impact of these attacks against published SDN defense solutions. Finally, we abstract our findings to offer the research and development communities with a deeper understanding of the common design and implementation pitfalls that are enabling the abuse of SDN networks.**

***Index Terms*—Software defined network security, SDN security, network security**

I. Introduction

Software-defined networking has steadfastly emerged as the flagship technology for enabling dynamism and elasticity in next-generation networks. SDNs offer a pathway for cloud network orchestration and many enterprise networks have already deployed or plan to deploy such technology in the near future. While some are motivated by reduced operational cost, others make the switch to improve the flexibility of their network architectures and enable exciting new network functionality (e.g., Google data centers [62], [26]). As the popularity of SDNs increase, it is likely that legacy network elements will be steadily replaced with SDN applications and white-box network elements.

Manuscript received January 22, 2016; revised February 1, 2017; accepted August 15, 2017;

C. Yoon, S. Lee, H. Kang, and T. Park are with KAIST, Daejeon, South Korea (e-mail: {chyoon87, lss365, kangheedo, taejune.park}@kaist.ac.kr).

S. Shin is the corresponding author and he is also with KAIST, Daejeon, South Korea (e-mail: claude@kaist.ac.kr)

V. Yegneswaran and P. Porras are with SRI International, Menlo Park, CA 94025 USA (e-mail: {vinod, porras}@csl.sri.com)

G. Gu is with Texas A&M University, TX 77843 USA (e-mail: guofei@cse.tamu.edu)

However, among the impediments to large-scale migration of legacy production networks are concerns about the security implications of such changes. On the one hand, SDNs offer an agility in software-controlled network flow and topology management that is well suited to address the increasing needs of today's dynamic networks. On the other hand, this same new agility in dynamic flow and topology management also introduces new challenges in maintaining well-defined and consistent network perimeter controls. For example, can one or more network applications inadvertently manipulate flows in a manner that bypasses security devices or security policies? [45] Can one network application produce unexpected interference with the operation of other peer network applications and impact the network's robustness? [53] In contrast to legacy networks, which are typically constrained by tight-knit topologies and restricted operations, the security of SDNs is highly dependent on the trustworthiness of the network applications and the integrity of the forwarding plane.

In this paper, we adopt a more systematic and pragmatic approach to evaluating the vulnerabilities that arise throughout SDN stacks. We first conduct a comprehensive survey of possible methods for abuse or exploit of an example SDN stack that is implemented in OpenFlow. We utilize these examples to develop a taxonomy for such attacks, that is based on the CIA model. Informed by the results of our study, we suggest architectural improvements to detect or mitigate the impacts of such abuses. We begin by first surveying example vectors for abuse or direct attack reported in current SDN literature, including many SDN-related research projects [43], SDN research papers [34], [35], and technical blog articles [12], [11] from both the network and security communities. We extend this work by discovering 12 additional novel abuse or attack cases through a thorough examination of well-known SDN elements and their use cases. For example, we explore various security-related abuses that may arise from network applications running on real-world open-source SDN controllers, such as OpenDayLight [1] and FloodLight [15].

For each identified vulnerability, we conduct a systematic evaluation of the attack surface with real-world experiments. The objective of these experiments is to underscore the feasibility, effectiveness, and significance of such attacks and API misuses in practice. We implement and reproduce the attacks in a realistic SDN testbed consisting of real-world SDN controllers and OpenFlow network devices, including: (i) three different well-known SDN controllers, FloodLight [15], OpenDaylight [1], and POX [46]; and (ii) three OpenFlow network devices supporting SDN functions from two different

vendors, HP and Pica8.

Finally, we distil the lessons learned from our analysis into a set of development and operational best practices that serve to mitigate the effects of such vulnerabilities. We do so while acknowledging the inherent difficulties in designing bug-free applications and the benefits of an open-source community-driven software development process. If SDN developers and administrators carefully design network applications and operate SDN elements with such rigorous assessments, we believe that the attack surface of a deployed SDN can be greatly reduced.

In summary, the key contributions of this paper include the following. First, we conduct a comprehensive examination of SDN attacks and security-relevant abuses, including a presentation of 12 previously unreported SDN attack vectors. We classify each vector analyzed in the form of an SDN attack taxonomy that offers a systematic categorization of the design or implementation weakness that enables each of our reported cases. Second, we implement and test 22 of these attack and abuse strategies using real-world SDN devices to validate their feasibility and prevalence, and to assess their impact. We summarize our test-case findings in Table II, and observe that indeed many of our cases occur across independently-developed control-layer implementations. Third, we provide an in-depth analysis of each attack or abuse strategy, and discuss possible methods to harden an SDN in a manner that prevents or reduces the impact of each test case. In addition, we introduce guidelines for SDN application design and development that is specifically tailored to avoid similar misuse attacks. We hope that this focused assessment and attack taxonomy will bring greater awareness to the design and implementation pitfalls that render today's software-defined networks vulnerable to a wide range of abuse.

II. Background

A. What is Software Defined Networking (SDN)?

A fundamental issue with legacy network devices is their functional rigidity. Specifically, a legacy network device (e.g., network switch) consists of two main components: (i) control plane and (ii) data plane. The control plane conducts complicated network functions such as creating routing tables to determine network flow control policies. The data plane is tightly coupled with the control plane, and it handles hardware-level network packet forwarding based on the policies from the control plane. If one desires to add a new network function or protocol, the legacy device must be removed and a redesigned device must be implemented and re-deployed in its place. This prospect has seriously hindered the ability to rapidly innovate, test, and deploy new or advanced network functions and protocols.

Emerging programmable network architectures, such as *Software Defined Networks (SDNs)*, seek to address this limitation by separating the control plane from the data plane [57] [21] [38]. An SDN simplifies the data plane to provide basic and efficient packet forwarding, and separates the complex and dynamic control plane functions into software applications, which are hosted on commodity hardware platforms, referred to as the controller. This architectural separation of the data and control planes enables one to easily program new network control functions as applications.

The Open Networking Foundation [40] conceptualizes SDN architectures into three-layers, as shown in Figure 1. Here, the *infrastructure layer* denotes the data plane. To add more programmability to the control plane, the control plane is separated into the *control layer* and the *application layer*. The role of the control layer is analogous to that of a traditional operating system running application programs, it manages SDN applications and provides developers with an interface for writing network flow management applications that control the packet handling operations of the data plane.

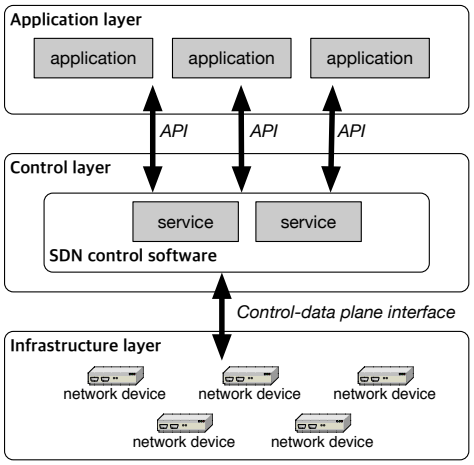


Fig. 1. Conceptual Architecture of Software Defined Networking (SDN) from Open Networking Foundation [41]

B. What is OpenFlow?

The OpenFlow [38] open standard protocol represents one popular embodiment of an SDN. It specifies functions of network devices (e.g., switch) and defines the communication protocol between the data plane and the control plane. OpenFlow-enabled network devices (i.e., data plane) are managed by OpenFlow controllers (i.e., control plane) such as NOX [18], Floodlight [15], and POX [46].

III. Security requirements for SDN

As the first step of our study, we explore the *unique* security requirements that must be fulfilled in order for a software-defined network to provide a reliable and secure networking environment. We identify the critical assets that must be secured for each layer of the SDN architecture, and apply the classic CIA (Confidentiality, Integrity and Availability) triad model to derive the security requirements. Table I summarizes the unique critical assets that must be protected in each layer of SDN, and the assets are classified according to the respective CIA attributes.

A. Control plane assets

One of the most critical assets in the control plane would be the *controller instance* itself. The controller instance, which is usually a single process, should be available at all times.

TABLE I
CRITICAL ASSETS IN SDN

Location	*Potential risk (CIA)*	*Critical assets*
Control Plane	Confidentiality	Controller configuration Network service configurations Global network-view
	Integrity	Controller configuration Network service configurations Active network services Global network-view Network behavior
	Availability	Controller instance Network services Global network-view
Control Channel	Confidentiality	Control messages
	Integrity	Control messages
	Availability	Control-data plane connections
Data Plane	Confidentiality	Network information Flow table (entries)
	Integrity	Flow table (entries)
	Availability	SDN-specific services

Whenever the control plane becomes unavailable, the entire network goes out of control. Furthermore, loss of confidentiality and integrity of *controller configuration* imposes risk. The controller configuration usually contains extremely sensitive information, such as the administrator login credentials, REST API access URLs etc., and adversaries can simply use such information to take over the entire network or manipulate the information to cause configuration errors.

Another critical asset is the *network service* (or SDN application). Network services usually perform useful and important network functions, ranging from simple forwarding to advanced load balancing. Such services should always be *available* to the network; failure to continuously provide the services will affect the network behavior. The integrity of the *active network service list* also must be preserved for the same reason. Moreover, these network services are also configurable, and the *configuration of the services* should never be disclosed or manipulated. For example, the configuration information may include the ACL rules for firewall service, and disclosure or manipulation of such information may put the entire network at risk.

When these network services make network control decisions, a *global network-view* is often referenced. A simple forwarding service, for instance, will calculate the path based on the global network-view and install flow-rules to the devices on the paths. Hence, inaccurate or manipulated global network-view will cause various network problems, and the information should always be available to the network services. In addition, a complete network topology information is sensitive and thus should be kept confidential.

Finally, the network behavior should never be intentionally manipulated by the entities other than the authorized SDN controller and network services activated by the network administrator.

B. Control channel assets

An SDN controller establishes connections with multiple network devices and exchanges *control messages*, mostly to control the network. These control messages contain sensitive network information and important control decisions, and therefore, the confidentiality and integrity of such messages must be preserved. Obviously, the connections between the controller and devices should remain available at all times.

C. Data plane assets

The data plane components (or SDN switches) establish connections with a centralized SDN controller and communicate with the controller via exchanging control messages. An SDN switch should always be able to accept/send control messages and install flow rules as instructed. Such basic SDN-specific services should be available under any consequences.

In an SDN, an SDN controller dynamically programs the network by installing flow rules to the underlying SDN switches, and the flow rules installed to each switch (flow table entries) should never be manipulated by unauthorized entities as any modifications made to the flow table will directly change the network behavior. Moreover, since the flow table entries intuitively and directly express network control policies, the entries must remain confidential. For example, by looking at the flow table entries, it is possible to infer the ACL policy enforced to the network.

Best practices and security requirements (e.g., controlled and secure CLI access, secure SNMP) applied in the management of traditional network switches also apply to SDN switches. In addition, since there exist SDN-specific vulnerabilities that adversaries may exploit to compromise the network, the SDN switches should be indistinguishable from any other network switches.

IV. ATTACK OVERVIEW

In the next few sections, we introduce various attack scenarios and test each scenario against real SDN components to verify that the components satisfy the security requirements discussed in the prior section.

An SDN typically consists of three primary layers: an application layer, a control layer, and a data plane (or infrastructure layer). There is also a control to data plane interface, which may be embodied as a standardized protocol, such as OpenFlow. Here, we normally consider the combination of the application layer and the control layer as the control plane; for our purposes, the terms infrastructure layer or data plane may be used interchangeably. Given this generalized layered architecture, our goal is to enumerate and classify vectors (or opportunities) for misuse or attack across the SDN stack. Our work surveys both vectors that have been identified in prior academic work, and newly identified scenarios, which we then organize into three categories. (1) *Control-plane-specific*, which includes abuse and attack vectors against the control and application layers. (2) *Control-channel-specific* scenarios involve the abuse of OpenFlow protocol. (3) *Data-plane-specific* attacks refer to interface attacks that involve the crafting of flow data that are input to the network devices, where an adversary may live within or outside the SDN managed network. We will refer to each class of attack by the following terms: **CP** for attacks that directly target the control plane, **CC** for attacks that seek to abuse the control channel, and **DP** for attacks targeting the data plane. We will

use the notation **CP-x** to refer to attack scenarios involving the control plane, where *x* is a number starting from 1.

All of these attacks, summarized in Table II, will be presented in the following sections with their working scenarios, real test cases, and in-depth analysis. Specifically, we describe 8 categories of control plane attacks and 16 attack examples. In the case of the control channel specific attacks, we describe two categories, and in each category we present one example case. Finally, we describe four categories of data plane attacks each with one example attack scenario. In Table II, the 10 known examples are marked with the (†) notation (the rest are new). We highlight the target location of each attack on the SDN stack in Figure 2.

Assumption: We acknowledge that there may be several different manifestations of an SDN. For example, an architecture separating the control plane from the data plane may be considered a form of SDN [17], as could a software router (e.g., XORP [21]). However, as OpenFlow is the most widely deployed instance of SDNs [38], we frame our study around such environments. Therefore, all attacks presented in this paper are centered around SDN components using the OpenFlow protocol and our primary focus is on attacks affecting SDN components (e.g., the SDN controller). Some attacks that destroy network configurations (e.g., dynamic flow tunneling and flow rule conflicts [45], [30]) are beyond our scope. However, we believe that they are well represented in previous studies [45], [7], [30]. In addition, several studies have been proposed that attempt to guarantee network correctness in an SDN environment [28], [3], [19], [16], [20]. While they offer promise in mitigating network failures, they are not focused on detecting adversarial components specifically targeting SDN components. Hence, we have not included these approaches in our attack taxonomy.

Test Environment: We have conducted an empirical study of vectors for abuse or direct attack across an SDN stack, specifically OpenFlow, through real experiments against a range of switches and well-known controllers. The environment consists of one controller machine, three physical OpenFlow-enabled switches (HP-3500yl, HP-3800, and Pica8-P3290) and three physical host machines. We have selected three well-known controllers for our test; POX (v. dart), FloodLight (v. 0.9) and OpenDaylight (v. hydrogen).

V. Remote Attacks on Control Plane

In theory, the SDN design principle should ensure the security of the control plane by isolating the control network (e.g. OpenFlow network) from the data network; however, there exist several attack scenarios that *can* harm a software-defined network implementation. In this section, we explore scenarios in which an adversary could *remotely* affect the network availability. We also introduce new attack scenarios, and demonstrate their effectiveness by launching them against three representative SDN controllers: POX, Floodlight, and OpenDaylight. For brevity, we will focus the presentation of our attack demonstrations using the Floodlight controller.

A. Denial-of-Service (DoS) [CP-R-1]

In software-defined networks, successful DoS against SDN controllers is a serious security threat as it may result in the loss of availability or the instability of the victim network. We describe two feasible *remote* DoS attacks against the control plane.

1) Packet-In flooding: An SDN switch notifies an SDN controller of each unseen flow (or for each flow table mismatch) as an event, and such events are transferred to the control plane via the Southbound API (i.e., OpenFlow).

In the case of OpenFlow, each unseen flow generates a *packet_in* event. Here, we consider a denial-of-service attack scenario that floods the SDN control plane with *packet_in* messages.

Such excessive number of *packet_in*s may cause a centralized SDN controller to end up in an unpredictable state [52], [54], [33]. For example, an SDN controller may use up all available system resources to process the flood of *packet_in*s and thus become unreachable.

2) Switch table flooding: According to Dover et al., [12], it is possible to fill up the switch table of Floodlight and cause switch disconnections by persistently sending forged OpenFlow messages. Floodlight adds one switch table entry when it receives OpenFlow's *features_reply* message with new DPID value; therefore, the switch table can be potentially filled up by continuous generation of such messages with different DPID values. This attack causes the switch table to use up all the memory resource available in the controller, and the attack eventually causes the controller to disconnect the linked switches from itself.

3) Switch identification spoofing: This attack, also introduced by Dover [11], modifies the OpenFlow control message to spoof its identity as if it is the target switch. The authors identify that Floodlight checks the DPID and the name of a switch to distinguish between switch devices. Furthermore, if Floodlight receives, from a compromised switch, a connection request in which its DPID and name are modified to have the same values as that of another switch connected to the controller, it disconnects the connection with the existing switch to establish a new connection with the compromised one. Leveraging such findings, Dover showed that it is possible to send forged OpenFlow messages to disconnect legitimate switches from a Floodlight controller.

4) Malformed control message injection: Prior work has also demonstrated attacks on the control plane with malformed OpenFlow control messages. Shalimov et al. [51] modified the message length field of the OpenFlow header to an incorrect value, and sent such malformed messages to various SDN controllers, such as POX and Floodlight. They claimed that POX and Floodlight disconnected the switches that were assumed to have sent such malformed messages.

5) System time manipulation: SDN controllers and applications often refer to system variables for various purposes. For instance, an SDN controller may refer to the system time variables for carrying out time-sensitive tasks, such as calculating packet timeouts. We expect that the behavior of SDN controllers can be affected by manipulating such system variables. Here, we modify the system time variable to confuse the controller.

As shown in Figure 3, Floodlight allows an application to modify the current time as much as desired, and such modification eventually causes the controller to disconnect

TABLE II
AN OVERVIEW OF SDN ATTACKS AND VULNERABILITIES

Attack		*Vulnerability*		*Examples and Description*
[CP-R] Control Plane Remote attacks				
[CP-R-1]	Denial-of-Service	**[V-3]**	Architectural bottleneck	**i) Packet-In flooding†:** The network hosts participating in an software-defined network may intentionally generate a large number of distinct network flows to exploit the bottleneck that exist in the SDN architecture [63], [10], [54], [33], [59], [60].
		[V-2]	Weak authentication	**ii) Switch table flooding†:** Crafted control messages may be continuously injected to the control plane, and it may eventually fill up the switch table maintained on an SDN controller [12].
				iii) Switch identification spoofing†: Weak switch authentication mechanism may be exploited to remotely drop the legitimate switch connections [11].
		[V-6]	Improper exception handling	**iv) Malformed control message injection:** Malformed control messages may be injected to the control plane to drop the legitimate switch connections.
		[V-8]	Dependence on external variable	**v) System time manipulation:** Arbitrary modification of system time may affect the behavior of SDN controllers.
[CP-R-2]	Network-view manipulation	**[V-2]**	Weak authentication	**i) Host location hijacking†:** Weak host authentication mechanism may be exploited to manipulate the host information [22].
				ii) Link fabrication†: Weak link discovery mechanism may be abused to manipulate the link information [22].
[CP-L] Control Plane Local attacks				
[CP-L-1]	Arbitrary system termination †	**[V-1]** **[V-4]**	Lack of authorization Monolithic controller design	An SDN application may execute a system exit command to terminate the controller instance [53].
[CP-L-2]	System resource exhaustion †	**[V-4]** **[V-5]**	Monolithic controller design Lack of resource management	Poorly designed or malicious SDN components(or applications) may use up the system resources, and ultimately affect the network availability [53].
[CP-L-3]	Network service neutralization	**[V-1]**	Lack of authorization	**i) Control message delivery obstruction I:** A control message subscription list may be manipulated to obstruct arbitrary SDN applications from receiving control messages.
		[V-7]	Naïve service chaining mechanism	**ii) Control message delivery obstruction II:** An SDN application may participate in a service chain and drop control messages before the other applications awaiting for them.
				iii) Service chain jamming: An SDN application may participate in a service chain and freeze (or hold) the sequential execution of services.
[CP-L-4]	Unauthorized application management	**[V-1]**	Lack of authorization	Unauthorized SDN controller components(or applications) may arbitrarily manipulate the state of the target application.
[CP-L-5]	Unauthorized network control	**[V-1]**	Lack of authorization	**i) Flow-rule modification:** An SDN application may issue a flow rule to overwrite the existing rule in the flow table of a switch to cause unexpected network behavior.
				ii) Flow table flushing: An SDN application may flush the flow table entries of a switch to disallow all the communication.
[CP-L-6]	Unauthorized network-view manipulation†	**[V-1]**	Lack of authorization	Unauthorized SDN controller components(or applications) may arbitrarily manipulate the global network view maintained within the control plane [53].
[CC] Control Channel attacks				
[CC-1]	Eavesdropping	**[V-9]**	Lack of practical encryption	An adversary may sniff the control channel to steal sensitive information: For example, an adversary may sniff the ongoing control messages on the control channel to learn the network topology.
[CC-2]	Man-In-The-Middle	**[V-10]**	Lack of integrity checks	An adversary may actively intervene in the control channel. For example, an adversary modify the flow rule message that is being transferred, and corrupt the behavior of a network.
[DP] Data Plane attacks				
[DP-1]	Flow-rule flooding†	**[V-3]**	Architectural bottleneck	Numerous flow rules may lead a data plane to be in an unpredictable state[63], [10], [52]. An adversary may intervene in the control channel and install a number of flow rules to the target switch to fill up the flow table.
[DP-2]	Switch firmware abuse	**[V-11]**	Hardware abuse	The characteristics or traits of a certain switch model may be abused. For example, an adversary may install the crafted flow rules that cannot be processed in the hardware table of a certain switch model.
[DP-3]	Malformed control message injection	**[V-6]**	Improper exception handling	A malformed control message may put the data plane in an unpredictable state. An adversary may inject a malformed control message to the data plane to interrupt the connection between the control plane and the data plane.
[DP-4]	Data leakage†	**[V-12]**	Architectural weakness	An adversary may retrieve sensitive network information of an SDN [52].

†Previously known/documented attacks (10 out of 22 attacks)

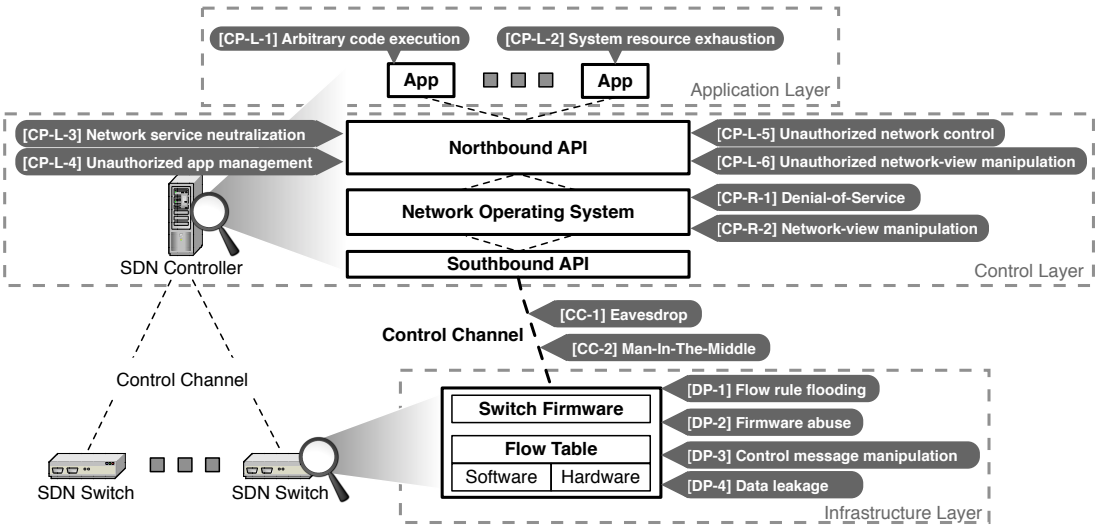


Fig. 2. Overview of the SDN attack surface: Presented attack scenarios cover all three layers of the SDN architecture and specific vulnerability locations targeted by each attack are highlighted.

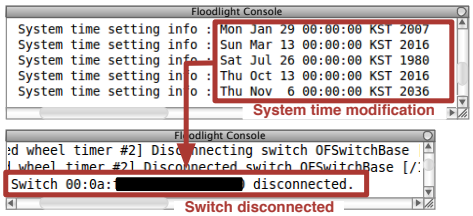


Fig. 3. Result: [CP-R-1-v] System Time Modification causing switch disconnection

the connection between the switch and the controller, as it does will receive a response from the switch to its heartbeat message within a proper time window. As in the Floodlight case, similar switch disconnections are observed when the time modification attack is launched against OpenDaylight. Unlike FloodLight and OpenDaylight, this attack does not affect POX because this controller does not refer to the system time. Given that this attack is feasible as long as adversaries can manipulate the system clock, it can be inferred that adversaries can also cause switch disconnections by launching network time protocol attacks from *remote locations* [37].

B. Network-view manipulation [CP-R-2]

In order to handle dynamically changing locations of the network nodes and hosts in modern networks, many SDN controllers often implement Host Tracking Services (HTS) and Link Discovery Services (LDS). Hong et al. [22] have demonstrated that such services are prone to topology poisoning attacks[1]. They have shown that it is possible to deceive the HTS of most currently available controllers by forging some network packets, since most HTS implementations only use simple identifiers to distinguish among the hosts. They have demonstrated this attack against a Floodlight managed network to show the effectiveness. They mention that POX and OpenDaylight are also susceptible to this attack.

The HTS, in modern controllers, automatically tracks each host across migrations using specific identifiers. Hong et al. have analyzed the source code of most currently available SDN controllers and discovered that they only use simple identifiers to distinguish among the hosts. Based on their findings, they introduced one possible attack scenario that leverages their findings on HTS to poison the network topology as shown in Figure 4.

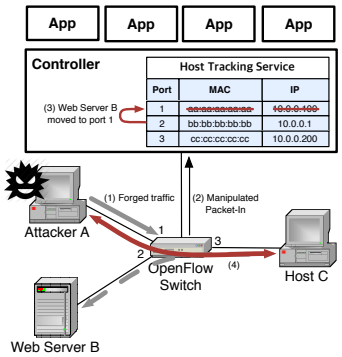


Fig. 4. Scenario: [CP-R-2-i] Host Location Hijacking.

As illustrated in Figure 4, WebServer B (at port 2) and Host C (at port 3) are the benign participants of the network, and Attacker A (at port 1) is malicious. When Host C attempts to access WebServer B, a *packet_in* message is generated and the controller issues flow rules that forward all traffic coming from port 3 to port 2 and *vice versa*. This is called ingress port-based forwarding, and in such instances, it is possible for the attacker to manipulate the network topology.

It is possible for (1) Attacker A to impersonate WebServer B by forging some network packets to deceive the HTS of the controller. Specifically, Attacker A can spoof packets to have the source MAC and IP addresses of WebServer B, and consequently, the switch issues (2) *packet_in* based on

[1]Please refer to the cited paper for additional details on the Link fabrication attack.

such manipulated traffic information. The *packet_in*, which is passed to the controller, is then processed by the HTS, and (3) the HTS believes that WebServer B has migrated and newly linked to port 1. Hence, the HTS updates its table and (4) all traffic destined to WebServer B is directed to Attacker A according to the manipulated network topology information.

VI. Local Attacks on Control Plane

In addition to the remote attack scenarios introduced in the previous section, attacks could also be launched locally, within an SDN controller. SDN controller implementations are also general software applications hosted on computing machines. One should not assume that the security of the control plane is ensured entirely through its network isolation.

In particular, similar to the mobile application ecosystems (e.g. Google Play or Apple App Store), one aspect of the SDN application ecosystem is its potential to offer rapid innovation through third-party application integration. Indeed, Hewlett Packard constructed an SDN App Store [25], and many other SDN controller vendors have published Northbound APIs to encourage the open development of SDN applications. In such an environment, various intrusion scenarios are possible as illustrated in previous work [48].

A. Arbitrary system termination [CP-L-1]

As most SDN controllers are designed to run SDN applications within the controller instance, a system call misuse scenario is possible. If (1) an SDN application invokes the system exit command, (2) it terminates not only the application itself but also the controller instance. As Shin et al. demonstrated, such (3) loss of the control plane is clearly undesirable and devastating to the data plane [53].

B. System resource exhaustion [CP-R-2]

Current SDN controller implementations are end-host user-level applications, and their performance is limited by the capabilities of the hosting machine. Shin et al. [53] demonstrate that it is possible to consume the system resources of the hosting machine with a simple SDN application and cause system failures.

C. Network service neutralization [CP-L-3]

1) Control message delivery obstruction I: Current SDN controllers often employ the *Observer pattern*, which is a software design pattern for handling distributed events, to efficiently and automatically notify subscribing SDN applications of newly generated control events. In such SDN controllers, an SDN application registers its *listener* to the list of *listeners* awaiting control events. When a *packet_in* arrives at the SDN controller, it is automatically delivered to all the subscribers. We demonstrate that it is possible to unsubscribe an arbitrary SDN application from the listener list. Figure 5 (top) illustrates such a misuse scenario. Here, (1) an SDN application accesses the list of *packet_in* subscribers to (2) forcibly unsubscribe App4. As a result, (3) *packet_in*s are no longer delivered to App4. (See Algorithm 1 for the detailed operation of the application.)

Algorithm 1 Control message delivery obstruction I

```
1:  procedure PktInUnsubscription
2:      pktInAppList ← list of packet-in subscribers
3:      targetApp ← name of target application
4:      i ← 1
5:      len ← length of pktInAppList
6:      while i ≠ len do
7:          name ← pktInAppList[i].name
8:          if name = targetApp.name then
9:              pktInAppList.remove(name)
10:         i ← i + 1
```

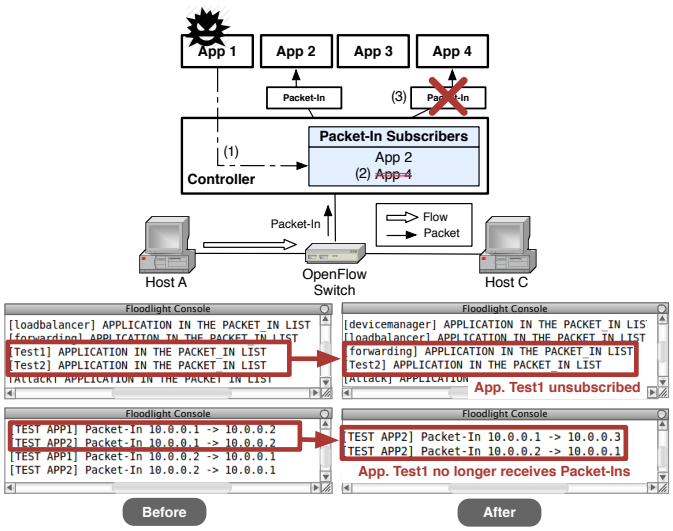


Fig. 5. Scenario (top) and Result (bottom): [CP-L-3-i] *packet_in* Unsubscription

We find that we can remove arbitrary Floodlight applications from the *packet_in* subscription list. We deploy three SDN applications to the controller: two test applications that subscribe themselves to the *packet_in* subscription list and one malicious application that unsubscribes one of the test applications from the list. As shown in Figure 5 (bottom), the malicious application successfully unsubscribes one test application from the list. As a result, the unsubscribed application is unable to receive any *packet_in*s. We find that such an attack is also possible with OpenDaylight. POX does not employ the *Observer pattern* to deliver control events to its applications, and is therefore not susceptible to the API misuse attack.

2) Control message delivery obstruction II: Applications deployed to an SDN controller are usually executed in a specific order. Such a series of application handlers is referred to as a *Service Chain*, and below we demonstrate how malicious applications may interfere with the chained packet handling of other SDN applications.

An example API misuse scenario that may cause service chain interference is illustrated in Figure 6 (top). In this scenario, the FWD app is responsible for forwarding a packet depending on a *packet_in* event. When Host A sends a packet to Host C, a *packet_in* is sent to the controller. Then, (1) the controller passes the *packet_in* to App1, App2 and App3 as a pre-defined order. (2) App3 (which happens to be malicious)

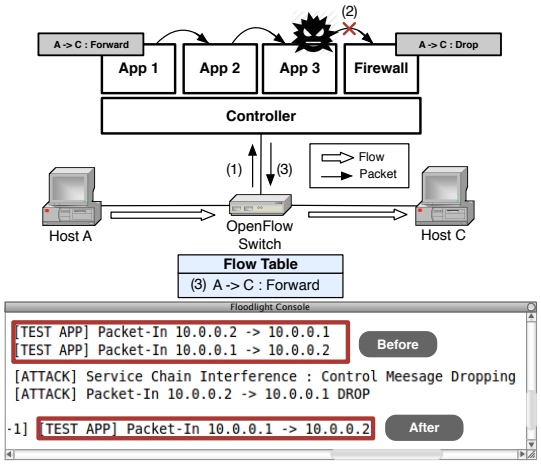


Fig. 6. Scenario (top) and Result (bottom):
[CP-L-3-ii] Control Message Drop causing *packet_in* cut-off

intentionally drops the *packet_in* without passing it to the FWD application. Accordingly, the FWD application cannot receive *packet_in* messages because App 3 intentionally dropped the messages. (3) The FWD application does not reply to the *packet_in* message; consequently, (4) the OF switch does not have any installed flow entries corresponding to the request from Host A. Thus, (5) Host A is unable to communicate with Host C.

To test this API misuse case, we deploy a malicious SDN application that selectively drops particular incoming *packet_in*s to the three SDN controllers, and Figure 6 (bottom) shows the outcome observed in the case of testing with Floodlight. The message of *TEST APP* in this Figure represents an SDN application that is part of the service chain, and it can receive any *packet_in*s before the interference (see Figure 6: Before). However, once a malicious application, which is scheduled to execute right before the *TEST APP*, starts to drop particular *packet_in*s, the *TEST APP* no longer receives such *packet_in*s (see Figure 6: After; only one *packet_in* has been observed).

We confirm that this attack is also effective in the OpenDaylight controller. However, POX passes the control messages to each POX application as function arguments, and the arguments are passed by value. In other words, each POX application takes a different copy of each control message, and therefore, POX is resilient to this API misuse attack.

3) Service chain jamming: A second service chain interference attack involves intentional insertion of infinite loops into the application. Similar to the previous API misuse case, App3 can be intentionally programmed to fall in an infinite loop, thus causing the entire controller instance to freeze for an indefinite time. We demonstrate the feasibility of this API misuse by evaluating it against the three SDN controllers under the same experimental condition as the previous API misuse attack, except for the fact that the malicious service chain application now includes an infinite loop. As shown in Figure 7, this application causes the entire Floodlight instance to be stuck in an infinite loop as the malicious application is executed reactively upon the arrival of a *packet_in* event. The same behavior was also observed when such applications were deployed on POX and OpenDaylight.

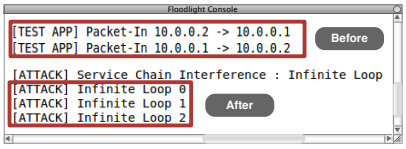


Fig. 7. Result: [CP-L-3-iii] Infinite Loop Insertion

D. Unauthorized application management [CP-L-4]

SDN applications with unrestricted authority may offer maximal network programmability. However, such unconstrained power may also introduce new security threats. We present a scenario that illustrates how an SDN application can misuse Northbound APIs to evict other legitimate applications.

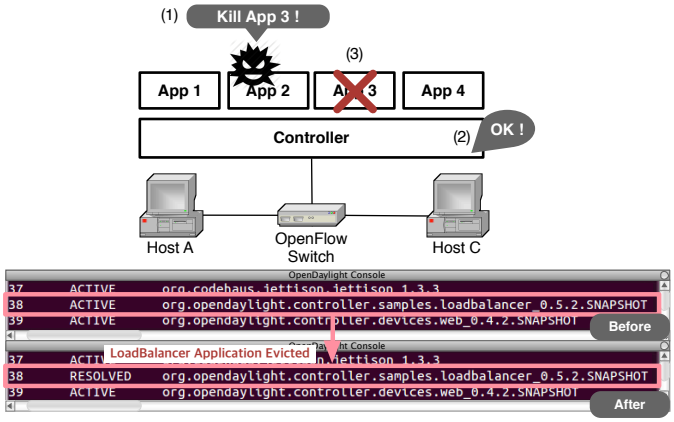


Fig. 8. Scenario (top) and Result (bottom):
[CP-L-4] Application Eviction unloading LoadBalancer app

As shown in Figure 8 (top), (1) a malicious SDN application may legitimately call the function that terminates other application via the Northbound API. In response, (2) (3) the controller terminates the innocent application as requested. This misuse case may directly compromise the managed network as it may terminate the basic networking applications, such as forwarding or routing applications. In addition, it may also lead to termination of security critical applications like firewalls or IDSs.

Of the three SDN controllers that we considered in this paper, the aforementioned API misuse is only effective against OpenDaylight, because it is the only controller that allows for dynamic loading and unloading of SDN applications. Figure 8 (bottom) shows how a malicious OpenDaylight application can dynamically force the unloading of other deployed applications.

E. Unauthorized network control [CP-L-5]

The *packet_in* control message in OpenFlow notifies an SDN controller of a newly incoming network flow. Similarly, there are other types of control messages that support various other networking features (e.g., *flow_mod*, *port_mod*, *table_mod*). However, most SDN controller implementations

do not manage or restrict the use of such control messages; therefore, we assert that it is possible to manipulate the control plane by abusing such messages. Below, we discuss two such control message misuse scenarios.

1) Flow-rule modification: Since there is no restriction on issuing control messages, an SDN application can issue *any* control messages at *any* time. In the scenario illustrated in Figure 9 (top), (1) a malicious application issues the crafted flow rule to (2) override one of the flow table entries; consequently, (3) the existing network connection between A and C is disallowed.

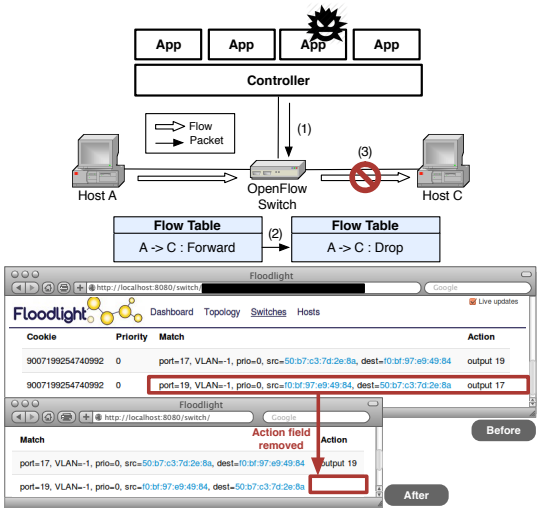


Fig. 9. Scenario (top) and Result(bottom):
[CP-L-5-i] Flow Rule Modification

We deploy a malicious Floodlight application that implements this attack and evaluate its viability by verifying that the application can indeed override the existing flow table entry. Figure 9 (bottom) illustrates the state of the flow table before and after this attack. As shown in the figure, a flow table entry that instructs the switch to forward flows to port 17 existed in the flow table before the API misuse attack. After the attack, we see that the flow table entry has been overridden with a different flow rule that does not take any action on the same flow. As this attack can override any flow rules issued by benign SDN applications, it has the ability to arbitrarily control the behavior of the network or to neutralize any security policies. Furthermore, we confirm that both POX and OpenDaylight are also vulnerable to this attack.

2) Flow table flushing: Next, we evaluate another API misuse scenario as shown in Figure 10 (top). In this case, (1) a malicious application continuously sends a control message that (2) clears all flow table entries. This scenario can cause flow entry mismatch for all incoming network flows and (3) degrade the network performance.

Figure 10 (bottom) illustrates the impact of this attack on our Floodlight testbed. We deployed a malicious application that persistently clears the flow entries installed in the flow table of a switch. The initial ping time before the attack is around 5 ms as the flow was uninitialized[2]. Once the flow

[2]The first packet of an unseen flow incurs extra latency as it incurs the controller communication cost

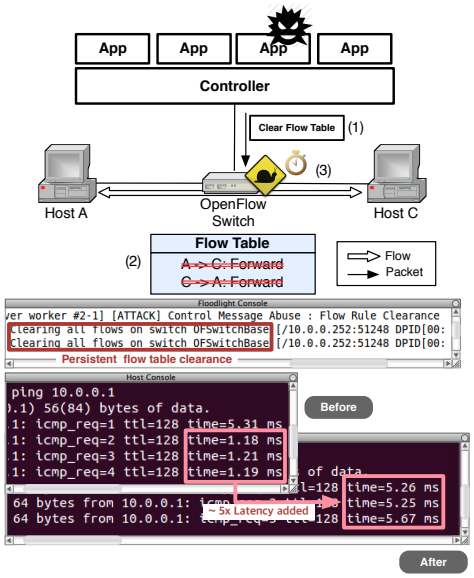


Fig. 10. Scenario (top) and Result (bottom):
[CP-L-5-ii] Continuous Flow table flush degrading network performance

is initialized, the ping time drops to 1 ms. However, during the attack, the flow table of the switch is consistently cleared; therefore, both the first and subsequent pings incur the flow initialization latency (Figure 10:After). The impact of such an attack in production environments can be severe, as it not only adds the flow initialization cost to every packet transferred on the network, but also eventually causes the packet_in *Flooding* attack. In the case of POX and OpenDaylight, such API misuse produces the same outcome as with Floodlight.

F. Unauthorized network-view manipulation [CP-L-6]

An advantage of the centralized SDN architecture is that it offers network-wide topology information to the control plane. For example, a routing application may leverage such information to efficiently calculate and offer the best routing path. Contemporary SDN controller implementations typically include a shared internal database for storage and management of a variety of network information that applications use for decision-making.

As has been demonstrated, most SDN controllers do not restrict access or modifications to such internal storage [53]. This misuse scenario is depicted in Figure 11 (left).

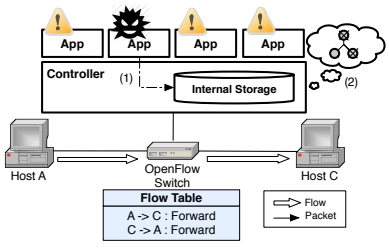


Fig. 11. Scenarios: [CP-L-6] Internal Data Storage Modification

As illustrated in the figure, (1) a malicious application makes unrestricted modification to data stored in internal storage. If the malicious application modifies the link information

stored in the internal storage, (2) all other applications that make decisions based on such information are affected, and may lead the network to an unpredictable state.

G. Summary of the Results

We summarize below our results of all discussed attacks launched against the SDN CP in Table III. Successful and unsuccessful attacks against POX, Floodlight (FL) and OpenDaylight (ODL) are marked by (O) and (X) respectively in the three columns.

TABLE III
SUMMARY OF CONTROL PLANE ATTACK RESULTS

		POX	FL	ODL
[CP-R-1-i]	Packet-In flooding	O	O	O
[CP-R-1-ii]	Switch table flooding	N/A[a]	O	X
[CP-R-1-iii]	Switch ID spoofing	N/A[a]	O	X
[CP-R-1-iv]	Malformed control message injection	O	O	O
[CP-R-1-v]	System time manipulation	X	O	O
[CP-R-2-i]	Host location hijacking	O	O	O
[CP-R-2-ii]	Link fabrication	O	O	O
[CP-L-1]	Arbitrary system termination	O	O	O
[CP-L-2]	System resource exhaustion	O	O	O
[CP-L-3-i]	Control message delivery obstruction I	N/A[b]	O	X
[CP-L-3-ii]	Control message delivery obstruction II	N/A[c]	O	△
[CP-L-3-iii]	Service chain jamming	N/A[c]	O	O
[CP-L-4]	Application eviction	N/A[d]	N/A[d]	O
[CP-L-5-i]	Flow-rule modification	O	O	O
[CP-L-5-ii]	Flow table flushing	O	O	O
[CP-L-6]	Unauthorized network-view manipulation	N/A[e]	O	O

[a]POX does not maintain switch table.
[b]POX does not use observer pattern to deliver control messages.
[c]POX does not employ service chaining mechanism.
[d]POX and FL do not support dynamic application loading/unloading.
[e]POX does not maintain global network-view.

VII. ATTACKS ON CONTROL CHANNEL

In this section, we introduce two attacks that could be launched against the SDN control channel, specifically in the OpenFlow protocol. We observe that although control channel communication in OpenFlow can be adequately secured using the SSL/TLS protocol, the use of such encrypted channel is not widely adopted [4]. Below, we describe attacks against the OpenFlow control communication channel and demonstrate their effectiveness on our testbeds.

A. Eavesdropping [CC-1]

Due to the absence of encryption in the control channel, it is possible to eavesdrop on the connection between the control plane and data plane. The scenario is that an adversary may sniff the ongoing OpenFlow messages to exfiltrate the topology information of the managed network.

To demonstrate the attack on our testbed, we wrote a small program that passively captures ongoing OpenFlow messages and parses the messages to extract the network topology information.

B. Man-In-The-Middle [CC-2]

Figure 12 (top) illustrates the MITM attack scenario that actively intervenes in the communication between the control plane and data plane to manipulate the ongoing OpenFlow messages on the control channel. If (1) the controller sends a flow rule that instructs the switch to forward a set of flows from host A to C, (2) the adversary can actively modify the action field of the rule to be "drop". As a result, (3) the flow rule that has been manipulated is installed to the switch, and ultimately (4) the flow from host A to C is dropped at the switch.

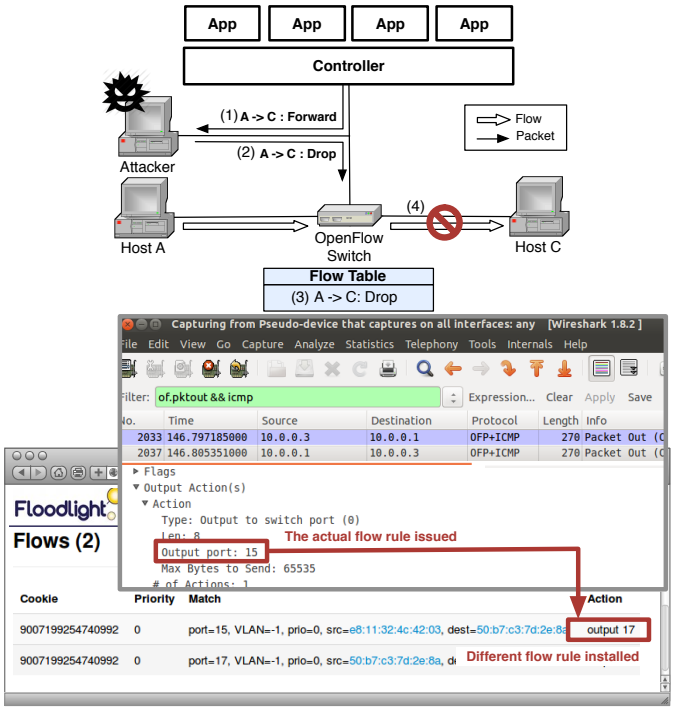


Fig. 12. Scenario (top) and Result (bottom): [CC-2] Man-In-The-Middle - Flow Rule Forgery

As shown in Figure 12 (bottom), we attempted to modify the action field of the flow rule to have a different switch port number. We demonstrate the successful flow rule modification using both Wireshark capture of the OpenFlow *flow_mod*[3] message at the network interface of the controller and also by listing the inserted flow rule via the Floodlight Web UI.

VIII. ATTACKS ON DATA PLANE

In this section, we introduce three attacks that can be launched against the SDN data plane. We evaluate these attacks on the testbeds with three different switch models; (i) HP 3500yl, (ii) HP 3800, and (iii) Pica8 P-3290. We provide detailed results of attacks launched against the HP 3500yl switch model (the results are similar to other models unless otherwise noted).

[3]flow_mod is a type of OpenFlow message that installs a flow rule to an OpenFlow enabled switch.

A. Flow-rule flooding [DP-1]

Since there is no restriction on issuing control messages, an SDN application can issue any control messages at any time. In this scenario, (1) a malicious application continuously generates flow rules to (2) consistently fill up the flow table of the switch, and finally, (3) the switch cannot handle more flow rules [52].

Figure 13 illustrates the dramatic change in the network latencies before and after this attack on HP 3500yl. Before the attack, the first ping time of a new flow was about 5 ms. When the flow table of the switch was filled with more than 30K flow rules, the same task took about 787 ms. This implies that this switch will delay any new flow by approximately 800 ms. We found that the other two forwarding devices are also vulnerable to this attack.

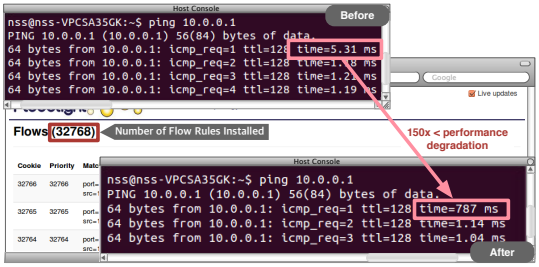


Fig. 13. Result: [DP-1] Switch Flow Table Flooding

B. Switch firmware abuse [DP-2]

Most OpenFlow-enabled switch models run custom and independent switch firmware implementations with varying capabilities. For example, the HP 3500yl and 3800 switch models [23], [24], do not support all of the OpenFlow specified 12-tuple match fields in the hardware (TCAM) flow table. To be more specific, if the source and destination MAC addresses are specified in the flow rules, that particular flow is forced to be processed in a software table, which is significantly slower than in-hardware processing.

This behavior of the switch firmware may be misused to degrade the overall network performance. In the scenario illustrated in Figure 14 (top), the malicious application (1) installs crafted flow rules that override the existing flow rules (IP matching) with hardware-unsupported match fields (MAC matching) specified. Consequently, (2) the network flow from host A to C is no longer processed in the hardware table, but rather in the software table resulting in (3) network performance degradation.

To verify the impact of this vulnerability on network performance, we launched this resource attack against our test-beds. Figure 14 (bottom) shows the ping times measured before and after the attack, and as shown, overall increase in ping time was observed. Meanwhile, Pica8's P-3290 switch model supported hardware-matching of all 12-tuple fields. Hence, the misuse was observed to be ineffective against this particular switch model.

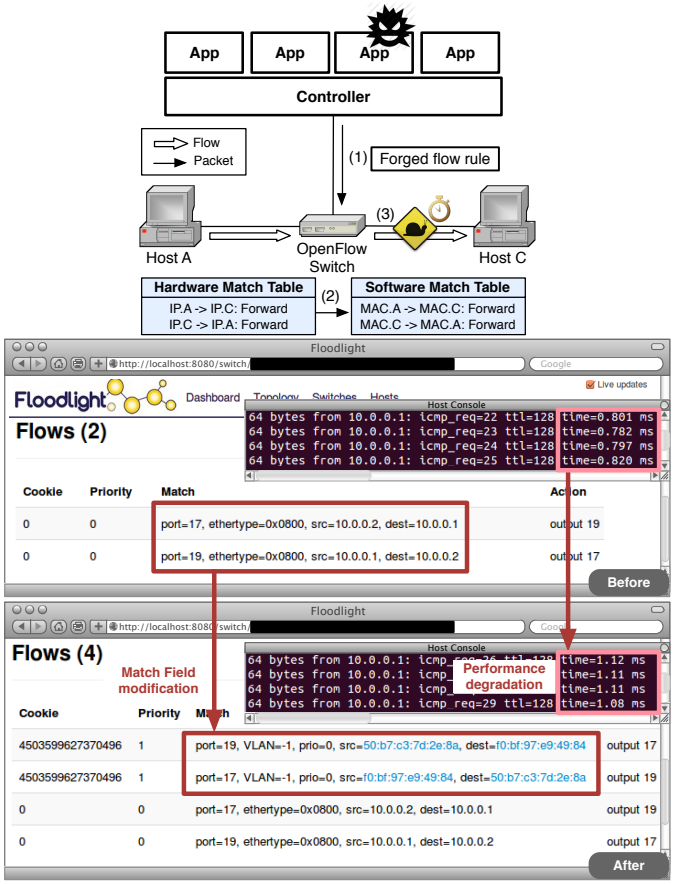


Fig. 14. Scenario (top) and Result (bottom):[DP-2] Switch Firmware abuse

C. Malformed control message injection [DP-3]

Manipulated control messages may be sent to the data plane with malicious intent, and it may cause the switch to end up in an unpredictable state.

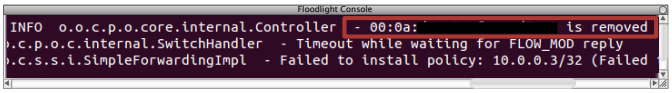


Fig. 15. Result: [DP-3] Malformed control message injection

We modified the length field of the OpenFlow message header to have an incorrect value, sent such malformed messages to the target switch, and observed the logs of the SDN controller to verify the impact of the attack. As shown in Figure 15, the attack resulted in a switch disconnection.

D. Data leakage [DP-4]

OpenFlow-enabled switches query SDN controllers for each flow table miss, and this significantly delays the flow processing. Such an architecturally inevitable delay is known as *control path delay*, and Shin et al. have shown that it is possible to fingerprint if the target network is an SDN or not by leveraging the control path delay as a distinguishing factor [52]. Moreover, Sonchack et al. have further developed the fingerprinting attack and shown that it is even possible to infer extremely sensitive information, such as ACL policies and host

communication patterns, by scanning the target network [55]. Both of the attacks scan the target network to measure the RTT (Round-Trip Time) for different network packets and analyze the collected data to infer the sensitive network information (and thus known as the *side-channel attacks*).

IX. Vulnerabilities and Defense mechanisms

As discussed in prior sections, various attack vectors are effective against contemporary SDN components. While we attempt to comprehensively evaluate the SDN attack surface and present a diverse set of attack scenarios, we believe that additional new and variant attack vectors will emerge in the future. Hence, it is important to track down the root cause of various SDN vulnerabilities. To that end, we abstract each attack to fundamental weaknesses (or vulnerabilities) of underlying SDN components and analyze them in greater detail.

In this section, we present the vulnerabilities (Table IV) extracted from the attack scenarios and discuss several proposed defense mechanisms that aim to mitigate or minimize the effects of aforementioned attacks (Table V).

A. Control Plane Security

1) Assuring Confidentiality: Current control plane implementations lack capability to authenticate applications (V-1), and allow applications to execute arbitrary system commands (e.g., system termination) (CP-L-1). As demonstrated in Section VI-A, the SDN controllers allowed SDN applications to execute sensitive commands; however, these are just a small set of all possible attacks that may leverage this vulnerability. Other attacks could also impact the confidentiality of the control plane and allow for exfiltration of sensitive information (e.g., controller configuration, global network-view, remote login credentials).

In order to prevent SDN applications from executing sensitive system commands on ONOS, a security extension has been recently added to ONOS. Their security feature revokes all the permissions granted to applications, and only grant minimum required permissions based on the security policy file. For example, recent security extension to ONOS [42] successfully mitigates the arbitrary system termination attack (CP-L-1) by revoking the permission to execute sensitive commands by default.

2) Assuring Integrity: Due to various vulnerabilities existing in the control plane, the integrity and confidentiality of the control plane cannot be guaranteed. Vulnerabilities that lead to the integrity issues include the following: *Lack of authorization (V-1)* and *Weak authentication (V-2)*.

[V-1] Lack of authorization: Arbitrary system termination attack (CP-L-1) has demonstrated that applications can execute any sensitive system commands to mount any attack against the control plane, and thus it is possible to manipulate sensitive information stored in not just the controller process but also the host machine running the controller. For example, current SDN controllers often locally store configuration files on the file system, and it is possible to easily manipulate such files thus breaking the integrity of the control plane.

Another important asset that should never be manipulated is *list of active network services*, and as demonstrated in Section VI-D, unauthorized application management (CP-L-4) attack breaks the integrity of the list by arbitrarily activating/deactivating SDN applications. We observe that SDN controllers tend to grant excessive authority to SDN applications via the flexible Northbound API. This allows the applications to exploit the API and perform malicious actions against peer applications. To restrict such unintended capabilities of SDN applications, the APIs must be carefully vetted and retrofitted. In a similar fashion, the global network-view can be unrestrictedly manipulated by any SDN application (CP-L-6), and such attacks may put the network in an unpredictable state, as all the network services make network control decisions based on the network-view. The security extension of ONOS also allows restriction of such capabilities by specifying and enforcing a security policy for each application.

Finally, all the applications are also capable of directly controlling the managed network by either directly issuing control messages or leveraging the related Northbound APIs (CP-L-5). Rosemary [53] and SE-Floodlight [44] employ an application permission model as a security enhancement, and its permission model constrains application interactions with the data plane to maintain the integrity of the network behavior. In contrast, the security extensions in ONOS allow for fine-grained API-level permissions that explicitly allow or deny applications' access to the network control APIs. PANE [14] provides mechanisms for delegating network authority which can be used for policy enforcement of untrusted network applications.

[V-2] Weak authentication: When SDN controllers establish connections with network devices and detect network hosts, each SDN controller authenticates the devices and hosts using different authentication mechanisms. However, Dover et al. [12], [11] and Hong et al. [22] have reported that Floodlight and OpenDaylight employ weak network element authentication mechanisms. Their attack scenarios (CP-R-1-ii,iii and CP-R-2) exploit the weaknesses in network device or host authentication mechanisms to manipulate the global network-view, thus breaking the integrity.

In response to this problem, Hong et al. have proposed a new security extension called TopoGuard [22], which provides enhanced network element authentication. TopoGuard is capable of verifying the legitimacy of host migrations, integrity of LLDP packets and switch port property upon each topology update.

[V-8] Dependence on external variable: As demonstrated in CP-R-1-v, the behavior of SDN controllers that reference system time was affected by the attack. The lesson that could be learned from this attack example is that referencing untrusted external variables could widen the attack surface of SDN controllers as well as any other systems. Hence, the use such untrusted and unpredictable external variables should be avoided in designing and implementing SDN controllers. Furthermore, this principle also applies to the network services (or SDN applications) that are hosted by SDN controllers. If an SDN application that directly controls the network uses system time to make routing decisions, the entire network could be affected by time manipulation attacks (e.g., NTP attack [37]).

TABLE IV
MAPPING SDN VULNERABILITIES AND ATTACKS USING THE CIA MODEL

Location	*Potential risk (CIA)*	*Vulnerability*	*Attack examples*	*Affected assets*
Control Plane	Confidentiality	[V-1] Lack of authorization	[CP-L-1] Arbitrary code execution	*
	Integrity	[V-1] Lack of authorization	[CP-L-1] Arbitrary system termination	*
			[CP-L-4] Unauthorized application management	Network behavior
			[CP-L-5] Unauthorized network control	Network behavior
			[CP-L-6] Unauthorized network-view manipulation	Global network-view
		[V-2] Weak authentication	[CP-R-1] Denial-of-Service	Global network-view
			[CP-R-2] Network-view manipulation	Global network-view
		[V-8] Dependence on external variable	[CP-R-1] Denial-of-Service	*
	Availability	[V-1] Lack of authorization	[CP-L-1] Arbitrary code execution	*
			[CP-L-4] Unauthorized application management	Network behavior
			[CP-L-5] Unauthorized network control	Network behavior
			[CP-L-6] Unauthorized network-view manipulation	Global network-view
		[V-2] Weak authentication	[CP-R-1] Denial-of-Service	Global network-view
			[CP-R-2] Network-view manipulation	Global network-view
		[V-3] Architectural bottleneck	[CP-R-1] Denial-of-Service	Global network-view
		[V-4] Monolithic controller design	[CP-L-1] Arbitrary code execution	*
			[CP-L-2] Resource exhaustion	Network service
		[V-5] Lack of resource management	[CP-L-2] Resource exhaustion	Network service
		[V-6] Improper exception handling	[CP-R-1] Denial-of-Service	Network connectivity
		[V-7] Naïve service chaining mechanism	[CP-L-3] Network service neutralization	Network service
		[V-8] Dependence on external variable	[CP-R-1] Denial-of-Service	*
Control Channel	Confidentiality	[V-9] Lack of practical encryption	[CC-1] Eavesdropping	Control messages
	Integrity	[V-10] Lack of integrity checks	[CC-2] Man-in-the-middle	Control messages
	Availability	-	-	-
Data Plane	Confidentiality	[V-12] Architectural weakness	[DP-4] Data leakage	*
	Integrity	-	-	-
	Availability	[V-3] Architectural bottleneck	[DP-1] Flow-rule flooding	*
		[V-6] Improper exception handling	[DP-3] Control message manipulation	*
		[V-11] Hardware abuse	[DP-2] Switch firmware abuse	*

* denotes that the attack can potentially affect all the critical assets mentioned in Section III-A.

TABLE V
VULNERABILITIES AND DEFENSES

Vulnerability	*Defenses*
Control plane security	
[V-1] Lack of authorization	PANE [14] Rosemary [53] SE-Floodlight [44] ONOS security extensions [42]
[V-2] Weak authentication	TopoGuard [22]
[V-3] Architectural bottleneck	ONIX [32] ONOS [5] DIFANE [63] DevoFlow [10] AVANT-GUARD [54] Kotani et al. [33] Scotch [59] FloodGuard [60]
[V-4] Monolithic controller design	Rosemary [53]
[V-5] Lack of resource management	Rosemary [53]
[V-6] Improper exception handling	
[V-7] Naïve service chaining mechanism	
[V-8] Dependence on external variables	
Control channel security	
[V-9] Lack of practical encryption	
[V-10] Lack of integrity checks	
Data plane security	
[V-3] Architectural bottleneck	DIFANE [63] DevoFlow [10]
[V-5] Improper exception handling	
[V-11] Hardware abuse	
[V-12] Architectural weakness	Sonchack et al. [55]

3) Assuring Availability: SDN controllers often manage and control many (if not all) network devices operating in the network. Hence, when availability issues arise in the control plane, it invariably affects the entire network. We discuss various vulnerabilities that jeopardize the availability of the control plane assets and the network itself. Furthermore, we introduce and discuss defense mechanisms relating to each vulnerability.

[V-1, 2, 8] Integrity issues and availability: Failure to protect the integrity of the control plane assets also compromises the availability of the control plane. As demonstrated in the previous section (Section VI), unrestricted application deactivation can cause arbitrary network services to be unavailable (CP-L-4), and the manipulated global network-view and network behavior (CP-L-4,5,6) can disrupt network connectivity. We believe that it is possible to eliminate many of these availability issues by properly protecting the integrity of the control plane. In addition, the use of untrusted external variables should be avoided as it could affect network availability. System time manipulation attack (CP-R-1-v) caused switch disconnections in the networks managed by Floodlight and OpenDaylight.

In the case of the arbitrary code execution (CP-L-1-ii), applications may execute system commands to terminate the controller instance to make the entire control plane unavailable [53]. Systems such as Rosemary and defense mechanisms, such as ONOS' security extension and TopoGuard, introduced earlier for the confidentiality and integrity protection, provide resilience against these availability issues.

[V-3] Architectural bottleneck: This vulnerability naturally exist in the SDN architecture as the control plane manages the

network in a centralized manner, and it is possible to remotely exploit such weakness to affect the network availability (CP-R-1-i). Unlike the other vulnerabilities, this issue has been studied by many researchers as shown in Table V.

One approach to significantly reducing the impact of such attack is improving the scalability of the control plane, and ONIX [32] and ONOS [5] employ distributed SDN controller architecture to this end. Other approaches that take advantage of the data plane to reduce the burden on the control plane have been proposed as well [63], [10], [54], [33], [59], [60]. However, they have not been integrated into commodity controllers.

[V-4] Monolithic controller design: Most of the SDN controller implementations available today are, in fact, *monolithic* software applications. For example, Floodlight and OpenDaylight are Java applications that run on a single Java Virtual Machine (JVM), and the SDN applications also run on the same JVM. Shin et al. [53] have shown the risk of such monolithic architecture of SDN controllers by demonstrating system command execution (CP-L-1-ii) and resource exhaustion (CP-L-2) attacks, which ultimately crash the controller instances. Rosemary employs a *micro-kernel* architecture to enable application containment and eliminate this vulnerability.

[V-5] Lack of resource management: Even if an SDN controller has employed a micro-kernel architecture, the controller may still be susceptible to resource exhaustion attacks because different core modules of the controller and SDN applications are sharing the system resources. Although Shin et al.[53] has demonstrated a simple attack scenario, more sophisticated and critical attack scenarios could be effective against SDN controllers without resource management (e.g., exhaustive exploitation of network service/control plane APIs). In response to this problem, Rosemary [53] employs a resource monitor that dynamically monitors and constrains the resource usage of various SDN controller components.

[V-6] Improper exception handling: The capability to handle exceptions that might be raised during the execution of an SDN application is crucial to guarantee controller robustness. SDN controllers that we evaluated seemed to lack such capability as the manipulated control messages (CP-R-1-iv) induced unexpected behaviors at the SDN controllers. To efficiently discover and deal with such exceptions, formal techniques and traditional fuzz testing approaches may be adopted.

It is important that such exceptions must be properly handled as the unexpected behaviors of SDN controllers directly affect the managed network. Interestingly, the OpenFlow protocol provides no mechanisms to inform the control plane when it receives a malformed or illegitimate OpenFlow messages. For example, if there was an OpenFlow message for notifying the data plane of the reception of such messages, the SDN controller could have reacted more flexibly rather than simply disconnecting the switch from the network.

[V-7] Naïve service chaining mechanism: We found that the Floodlight and OpenDaylight controllers employed *Service Chain* mechanism for SDN applications to systematically control and manage the network. However, such *Service Chains* are susceptible to the possible interferences (CP-L-3-ii, iii), and the fundamental vulnerabilities lie in both the application and control layer of the SDN architecture. CP-L-3-ii and iii inform two separate design concerns that the control layer must consider; the former demonstrates the importance of control message delivery guarantees, and the latter illustrates the necessity of a fail-safe service chaining mechanism (e.g., network application time-out mechanism). In order to assure the network service availability, SDN controllers must deliver control message to all network applications in any circumstances and deal with possible network application failures.

In case of the application layer, there is no restriction on what SDN applications might be deployed. To date, no standards have been proposed to analyze and verify the behavior of SDN applications prior to deployment. Since SDN applications are commonly written in common programming languages, such as Java and Python, traditional methods for software profiling or analysis would be a good starting point of developing a method to effectively review or verify SDN applications.

B. Control Channel Security

1) Assuring Confidentiality: Current SSL/TLS protection adds noticeable performance penalty to the control channel (V-9) [13]. Although OpenFlow optionally offers such protection, it is rarely used; even worse, some of the switch models or SDN controllers do not support it for this reason [4]. Therefore, an alternative encryption mechanism that is both lightweight and reliable is desirable in the future.

2) Assuring Integrity: As we demonstrated, OpenFlow allowed active flow-rule modification during the man-in-the-middle attack (CC-2), and this is possible because OpenFlow does not implement the control message integrity checking mechanism (V-10). Such mechanism is unnecessary if SSL/TLS protection is enabled; however, as mentioned, current SSL/TLS is not suitable for protecting large SDN networks.

3) Assuring Availability: In the case of OpenFlow, we are unaware of specific vulnerabilities or attack scenarios that affect southbound OpenFlow communications; however, it is prudent to rigorously test and verify the protocol with formal protocol verification techniques to guarantee the availability of the control-data plane connections.

C. Data Plane Security

1) Assuring Confidentiality: Due to the separation of the control plane and the data plane, SDNs are naturally prone to the timing-based side-channel attacks (V-12) [52], [55], and Sonchack et al. have proposed a timeout proxy that could be deployed to SDN switches as a countermeasure [55]. This timeout proxy normalizes the overall amount of packet processing time by directly participating in the flow rule installation process. It keeps track of every packet that is forwarded to the controller, and if the controller responds too soon or too late, the proxy installs the flow rule instead of the controller.

2) Assuring Availability: We have identified three SDN-specific vulnerabilities that compromise the availability of the data plane.

[V-1] Architectural bottleneck: Architectural bottleneck attacks have been discussed in prior papers[63], [10], and there are some approaches to mitigate the effect of this attack. For

example, DIFANE [63] tries to merge some related flow rules to save the space of the flow table in a switch, and DevoFlow proposes a way of reducing the number of flow rules with an extended OpenFlow message [10]. All these approaches assume that some flow rules are related and could be merged into a single (or a few) wildcard rules. However, if SDN applications want to handle each flow with a different method, then the proposed methods may not be applicable. A commercial switch (e.g., Arista 7050T [2]) addresses this problem by simply dropping requests from the control plane when it receives many flow rule requests in a short time. This approach is easy to realize, but it is not an ideal solution, because SDN applications often need to process many flow requests simultaneously to handle routine network traffic surges.

[V-6] Improper exception handling: As discussed in the previous section, switch firmware should be able to handle exceptional inputs. Again, SDN control protocols (e.g., OpenFlow) need to implement features for handling exceptions to minimize the impact of such attack.

[V-11] Hardware abuse: Just like traditional networking devices, OpenFlow enabled switch models also employ TCAM to perform fast hardware table lookups. However, such hardware TCAM is too costly, and accordingly, only a small portion of the forwarding table is often implemented on TCAM. Due to the limited capacity of the TCAM table, the switch firmware restricts the use of the hardware table, and we have shown how such vulnerabilities can be abused to affect the network performance. As demonstrated in Section VIII-B, the Pica8 switch model supports all 12-tuple OpenFlow (version 1.0) match fields and thus avoided such an attack. In order to cope with such a flow table availability problem, different approaches [63], [10] to efficiently use flow tables have been proposed. However, such methods are rather temporary and limited as they cannot keep up with constantly evolving SDN protocols. For example, a recent version (1.4) of OpenFlow has become even more complicated than version 1.0; it has 41 fields to be matched [6]. Hence, SDN switches should be reconfigurable; the switches should be flexible enough to support new match fields that may appear in the future, and P4 [6] proposes an effective approach to this problem.

X. Research Challenges in SDN Security

In this section, we will discuss key research problems that warrant additional research attention as well as promising approaches to addressing these problems.

A. Control Plane: Challenges and Research Horizons

Our survey finds that the area of SDN control plane security remains understudied and narrowly focused despite the growing popularity and industrial adoption of SDN. Our systematic security evaluation of current SDN controller implementations highlight several areas that warrant additional research.

The architecture of modern network operating systems (NOSs) is growing in feature and complexity, much like traditional operating systems. For example, contemporary NOSs, such as ONOS and ODL, have a rich library set and allow for dynamic loading and unloading of applications. We believe it makes sense to extend security mechanisms from traditional OSes to NOSes as well. For example, SE-Linux [47] could be an excellent reference model to design an authorization model for NOSes. Porras et al. [44] have implemented role-based access control mechanism to constrain applications' the control-data plane interactions, which partially solves *'lack of authorization' (V-1)* problem.

Our study also demonstrates that there exist architectural vulnerabilities, and hence, a fundamental rethinking of NOS architectures should be considered. Here, the architecture of the traditional OS could be used as a reference in designing a more robust NOS. For example, Shin et al. adopt a *micro-kernel* OS approach for securing the controller from buggy and malicious co-resident applications. However, their system is not resilient to control-flow saturation attacks or to rootkits that infect the NOS without crashing.

Furthermore, the security of the SDN application ecosystem should also be seriously considered. While the SDN community is encouraging open development and distribution of SDN applications for rapid evolution the technology, security problem associated with such environments is not new. In the case of the mobile application ecosystem, third-party distribution sites have been widely abused as attack vectors to compromise millions of end-users. We believe that SDNs should incorporate proactive security measures to address the potential of such attacks. For example, the Android operating system includes diverse mechanisms (e.g., sandboxing, application manifest) to protect itself and other benign applications from a compromised or malicious application. Inspired by Android's application security system, ONOS, in security mode, enforces security policy to applications and thus protects the core operating system from potentially untrusted applications. ONOS further extends Android's concept of application security mechanism to support large scale SDNs; it employs its own permission model that works in distributed settings, efficient permission checking mechanism that minimizes network performance impact, and additional network header space access control mechanism. However, there still remains several research questions that should be answered: How do we construct a safe application distribution environment for SDNs? How do we ensure the authenticity of applications? How can we detect malicious SDN applications? How can we guarantee the integrity of applications? We believe that formal techniques like NICE [7] provide a good starting point toward answering these questions.

B. Control Channel: Challenges and Research Horizons

A simple means to defend against certain attacks introduced in Section VII is encrypting the communication between the control and data plane, using SSL/TLS, as recommended by the OpenFlow specification. However, such encryption is invariably turned off in large enterprise and datacenter networks, due to the associated performance overhead. The centralized architecture of an SDN implies that the controller in a modern datacenter typically needs to respond to millions of flow requests made by hundreds of switches[58]). For this reason, network administrators often tend to simply disable encrypted channels, which represents a classic tradeoff (dilemma) between performance and security.

Therefore, the key research question is, *how can we achieve the best tradeoff between performance and security?* We believe that the simplest and most effective way to handle this problem is exploring alternative encryption techniques that are lightweight, secure and scalable. This objective could also be realized by employing custom high-performance encryption hardware (e.g., secure co-processors).

C. Data Plane: Challenges and Research Horizons

SDN data planes are designed to be simple and efficient by implementing just the basic hardware logic for forwarding packets with high performance. This design choice implies that that critical attack vectors are less likely to exist in the data plane. However, we argue that the firmware in the data plane can be abused for attacking SDNs. For example, firmware in wireless routers are notorious for various security holes. To the best of our knowledge, there is no study that has considered security problems associated with SDN data planes. We note that the firmware of such SDN devices are different from that of existing network devices. Hence, these differences should be well understood and new methods to detect (switch) firmware defects in terms of security or reliability should be investigated.

Also, SDN switches and an SDN controller communicate with each other, and thus can affect each other. Hence, the following questions should be answered: How to ensure the trustworthiness of forwarding devices? How to securely deploy and install devices (registration, authentication, and authorization) in a network infrastructure?

Other potential threats, beyond attacks demonstrated in this paper, also make the case for a more robust data plane. For example, a recent study revealed that a large number of commodity routers on the Internet could be simply scanned and accessed using default passwords [9]. In the case of an SDN network, the impact of such an attack is much more detrimental than in a legacy network, because one or more compromised SDN devices may leak sensitive information (e.g., network topology) or even directly attack the linked SDN controller. Such problems argue for the development of resilient data planes and new countermeasures to protect an SDN network from compromised data planes.

XI. Related Work

Scott-Hayward et al. and Kreutz et al. have recently published a broad survey of security in SDN [49], [34]. These surveys include studies of various SDN-specific security vulnerabilities [56], [29], [35]. In particular, Kreutz et al. divided SDN attacks into seven different categories [35]. Other studies have identified specific security deficiencies that arise in both SDN architectures and implementations. For example, [61], [52], [45], [4], [53] have critically examined the lack of isolation, access control, and the protection mechanism within the control layer. Chandrasekaran et al. [8] have presented vulnerabilities in the control plane that arise from software errors that exist within SDN applications, and Jarraya et al. [27] have explored deficiencies in the security of the northbound interface. Kloti et al. proposed various forwarding rule guessing and controller fingerprinting strategies via spoofing attacks [31]. Another survey paper, [50], has suggested a possible denial of service attack scenario that exploits limitations between then the centralized controller and the flow-table.

These early studies motivate our work, and we incorporate all described attack cases in our taxonomy. Furthermore, while these studies have generally hypothesized these attack scenarios, they do not present implementation studies of these attacks within a real test environment, which leaves open the following question: which attacks are theoretical, and which are reducible to practice on real SDN implementations? In addition, some studies propose a narrow attack scenario; here, we explore broad variants of the attack strategy. The result of our efforts have produced 12 new attack scenarios, whose feasibility have all been validated, through practical implementations, on a physical SDN test network. Furthermore, a SDN security assessment tool called DELTA [36] was recently developed. DELTA is a practical use case of our attack taxonomy, and this paper forms the basis for several of the attack instances that are implemented within DELTA. The key contribution of the DELTA framework is in the development of the attack management module and the control-flow and input fuzzing agents

XII. Conclusion

The paper presents a comprehensive analysis of the vectors for potential abuse or attack that arise in OpenFlow network stacks. We provide a generalized categorization of 22 separate vectors for abuse or direct attack, which arise from diverse interaction avenues that are supported by popular OpenFlow network implementations. Furthermore, we validate the feasibility and impact of these abuse and attack scenarios through implementation and testing. We believe that 12 of these threat scenarios are novel and unpublished.

Like prior attack taxonomy efforts [39], the proposed organization of vulnerabilities, attacks, and defenses is not intended to be all-encompassing. However, we believe that it lays a good touchstone for classifying future threats in this area. Our intent is to offer a survey of concrete interface points upon which both security and stability of SDNs depend. Where such attacks or abuses are identified, we argue for either the application of security services or an acknowledged effort to establish trust among those components with access to these interfaces. We hope that the insights provided by our work will inspire a more principled and systematic approach to designing the next generation of resilient SDNs.

Acknowledgment

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.2016-0-00078, Cloud based Security Intelligence Technology Development for the Customized Security Service Provisioning)

This material is based upon work supported by the National Science Foundation under Grant No. 1547206. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] A Linux Foundation Collaborative Project. OpenDaylight SDN Controller. http://www.opendaylight.org.

[2] Arista. Arista 7050 series. http://www.arista.com/en/products/7050-series/articletabs.

[3] R. Beckett, X. K. Zou, S. Zhang, S. Malik, J. Rexford, and D. Walker. An assertion language for debugging sdn applications. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, 2014.

[4] K. Benton, L. J. Camp, and C. Small. Openflow vulnerability assessment. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 151–152. ACM, 2013.

[5] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, et al. Onos: towards an open, distributed sdn os. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6. ACM, 2014.

[6] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[7] M. Canini, D. Venzano, P. Perešíni, D. Kostić, and J. Rexford. A NICE Way to Test OpenFlow Applications. In *Usenix Symposium on Networked Systems Design and Implementation*, April 2012.

[8] B. Chandrasekaran and T. Benson. Tolerating sdn application failures with legosdn. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 235–236. ACM, 2014.

[9] A. Cui and S. J. Stolfo. A quantitative analysis of the insecurity of embedded network devices: results of a wide-area scan. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 97–106. ACM, 2010.

[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 Computer Communication Review*, volume 41, pages 254–265. ACM, 2011.

[11] J. M. Dover. A denial of service attack against the open floodlight sdn controller. http://dovernetworks.com/wp-content/uploads/2013/12/OpenFloodlight-12302013.pdf.

[12] J. M. Dover. A switch table vulnerability in the open floodlight sdn controller. http://dovernetworks.com/wp-content/uploads/2014/03/OpenFloodlight-03052014.pdf.

[13] R. Durner and W. Kellerer. The cost of security in the sdn control plane.

[14] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi. Participatory Networking: An API for application control of SDNs. In *ACM SIGCOMM*, 2013.

[15] FloodLight. Open sdn controller. http://floodlight.openflowhub.org/.

[16] N. Foster, M. J. Freedman, R. Harrison, J. Rexford, M. L. Meola, and D. Walker. Frenetic: a high-level language for openflow networks. In *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, 2010.

[17] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A Clean Slate 4D Approach to Network Control and Management. In *Proceedings of ACM Computer Communications Review*, 2005.

[18] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an Operating System for Networks. In *Proceedings of ACM SIGCOMM Computer Communication Review*, July 2008.

[19] A. Guha, M. Reitblatt, and N. Foster. Machine-verified network controllers. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2013.

[20] N. Handigol, B. Heller, V. Jeyakumar, D. Maziéres, and N. McKeown. Where is the debugger for my software-defined network? In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2012.

[21] M. Handley, O. Hodson, and E. Kohler. Xorp: an open platform for network research. In *SIGCOMM Comput. Commun. Review*, 2003.

[22] K. Hong, L. Xu, H. Wang, and G. Gu. Poisoning network visibility in software-defined networks: New attacks and countermeasures. In *Proceedings of the 22nd Annual Network and Distributed System Security Symposium (NDSS15)*, February 2015.

[23] HP. Hp 3500 and 3500yl switch. http://h17007.www1.hp.com/us/en/networking/products/switches/HP_3500_and_3500_yl_Switch_Series/index.aspx.

[24] HP. Hp 3800 switch. http://h17007.www1.hp.com/us/en/networking/products/switches/HP_3800_Switch_Series/index.aspx.

[25] HP. Hp sdn dev center: Sdn app store. http://www.hp.com/go/sdndevcenter.

[26] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined wan. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, 2013.

[27] Y. Jarraya, T. Madi, and M. Debbabi. A survey and a layered taxonomy of software-defined networking. 2014.

[28] N. Katta, H. Zhang, M. Freedman, and J. Rexford. Ravana: Controller fault-tolerance in software-defined networking. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, 2015.

[29] S. M. Kerner. Is sdn secure? http://www.enterprisenetworkingplanet.com/netsecur/is-sdn-secure.html.

[30] A. Khurshid, W. Zhou, M. Caesar, and P. B. Godfrey. Veriflow: verifying network-wide invariants in real time. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, 2012.

[31] R. Klöti. Openflow: A security analysis. *Proc. Wkshp on Secure Network Protocols (NPSec). IEEE*, 2013.

[32] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: A Distributed Control Platform for Large-scale Production Networks. In *The Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[33] D. Kotani and Y. Okabe. 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*, ANCS '14, pages 29–40, New York, NY, USA, 2014. ACM.

[34] D. Kreutz, F. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *arXiv preprint arXiv:1406.0440*, 2014.

[35] D. Kreutz, F. M. V. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. In *Proceedings of ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN'13)*, August 2013.

[36] S. Lee, C. Yoon, C. Lee, S. Shin, V. Yegneswaran, and P. Porras. Delta: A security assessment framework for software-defined networks. In *Proceedings of NDSS*, volume 17, 2017.

[37] A. Malhotra, I. E. Cohen, E. Brakke, and S. Goldberg. Attacking the network time protocol. In *Network and Distributed System Security Symposium*, 2016.

[38] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38, March 2008.

[39] J. Mirkovic and P. Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *SIGCOMM Computer Communication Review*, 2004.

[40] Open Networking Foundation. https://www.opennetworking.org/.

[41] Open Networking Foundation. Software-defined networking: The new norm for networks. https://www.opennetworking.org/images/stories/downloads/white-papers/wp-sdn-newnorm.pdf.

[42] Open Networking Laboratory. Recent security extensions to onos. https://wiki.onosproject.org/display/ONOS/Security-Mode+ONOS.

[43] OpenFlow Security Analysis Project. OpenFlowSec.org. http://www.openflowsec.org.

[44] OpenFlowSec.org. Se-floodlight. http://www.openflowsec.org/Technologies.html.

[45] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu. A security enforcement kernel for openflow networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, 2012.

[46] POX. Python network controller. http://www.noxrepo.org/pox/about-pox/.

[47] S. Project. Selinux. http://selinuxproject.org/page/Main_Page.

[48] C. Röpke and T. Holz. Sdn rootkits: Subverting network operating systems of software-defined networks. In *Research in Attacks, Intrusions, and Defenses*, pages 339–356. Springer, 2015.

[49] S. Scott-Hayward, S. Natarajan, and S. Sezer. A survey of security in software defined networks. *IEEE Communications Surveys & Tutorials*, 18(1):623–654, 2015.

[50] S. Scott-Hayward, G. O'Callaghan, and S. Sezer. Sdn security: A survey. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–7. IEEE, 2013.

[51] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky. Advanced study of sdn/openflow controllers. In *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, CEE-SECR '13, pages 1:1–1:6, New York, NY, USA, 2013. ACM.

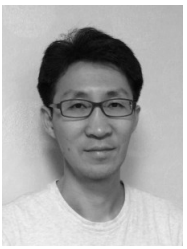
[52] S. Shin and G. Gu. Attacking software-defined networks: A first feasibility study (short paper). In *Proceedings of ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN'13)*, August 2013.
[53] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang. Rosemary: A robust, secure, and high-performance network operating system. In *Proceedings of the 21th ACM Conference on Computer and Communications Security (CCS14)*, November 2014.
[54] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS13)*, November 2013.
[55] J. Sonchack, A. Dubey, A. J. Aviv, J. M. Smith, and E. Keller. Timing-based reconnaissance and defense in software-defined networks. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 89–100. ACM, 2016.
[56] S. Sorensen. Security implications of software-defined networks. http://www.fiercetelecom.com/story/security-implications-software-defined-networks/2012-05-14.
[57] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb. Building a robust software-based router using network processors. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, SOSP '01, 2001.
[58] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. On controller performance in software-defined networks. In *Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, 2012.
[59] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen. Scotch: Elastically scaling up sdn control-plane using vswitch based overlay. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 403–414. ACM, 2014.
[60] H. Wang, L. Xu, and G. Gu. Floodguard: A dos attack prevention extension in software-defined networks. In *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*, pages 239–250. IEEE, 2015.
[61] M. Wasserman and S. Hartman. Security analysis of the open networking foundation (onf) openflow switch specification. 2013.
[62] Wired. Going With the Flow: Googles Secret Switch to the Next Wave of Networking. http://www.wired.com/wiredenterprise/2012/04/going-with-the-flow-google/.
[63] M. Yu, J. Rexford, M. J. Freedman, and J. Wang. Scalable flow-based networking with difane. *ACM SIGCOMM Computer Communication Review*, 40(4):351–362, 2010.

Changhoon Yoon received his BSE degree in Computer Engineering from the University of Michigan, Ann Arbor in 2010 and his MS degree in Information Security from KAIST in 2014. He is currently working toward his PhD degree at KAIST with research focusing on software defined network security.

Seungsoo Lee is a Ph.D. student in Graduate School of Information Security at KAIST working with Dr. Seungwon Shin in NSS Lab. He received his B.S. degree in Computer Science from Soongsil University in Korea. He received his M.S. degree in Information Security from KAIST. His research interests include secure and robust SDN controller, and protecting SDN environments from threats.

Heedo Kang is a Ph.D. student in Graduate School of Information Security at KAIST working with Dr.Seungwon Shin in NSS Lab. He received his B.S degree in Computer Engineering from Ajou University, Korea, in 2014. He received his M.S degree in Information Security from KAIST in 2016. His research interests include SDN security and performance.

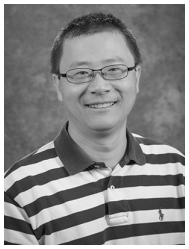
Taejune Park is currently pursuing his Ph.D. degree in School of Computing at KAIST, Republic of Korea, from September 2015. He received his B.S. degree in Computer Engineering at Korea Maritime and Ocean University, Republic of Korea, in August 2013, and his M.S. degree in Information Security at KAIST, Republic of Korea, in August 2015. His research interests focus on the security issues on SDN/NFV environments and data-planes.



Seungwon Shin is an assistant professor in the School of Electrical Engineering at KAIST. He received his Ph.D. degree in Computer Engineering from the Electrical and Computer Engineering Department, Texas A&M University, and his M.S degree and B.S degree from KAIST, both in Electrical and Computer Engineering. Before joining KAIST, he has spent nine years at industry, where he devised several mission critical networking systems. He is currently a Research Associate of Open Networking Foundation (ONF), and a member of security working group at ONF. His research interests span the areas of Software Defined Networking (SDN) security, IoT (Internet of Things) security, and Botnet analysis/detection.

Vinod Yegneswaran received his A.B. degree from the University of California, Berkeley, CA, USA, in 2000, and his Ph.D. degree from the University of Wisconsin, Madison, WI, USA, in 2006, both in Computer Science. He is a Senior Computer Scientist with SRI International, Menlo Park, CA, USA, pursuing advanced research in network and systems security. His current research interests include SDN security, malware analysis and anti-censorship technologies. Dr. Yegneswaran has served on several NSF panels and program committees of security and networking conferences, including the IEEE Security and Privacy Symposium.

Phillip Porras received his M.S. degree in Computer Science from the University of California, Santa Barbara, CA, USA, in 1992. He is an SRI Fellow and a Program Director of the Internet Security Group in SRI's Computer Science Laboratory, Menlo Park, CA, USA. He has participated on numerous program committees and editorial boards, participates on multiple commercial company technical advisory boards, and holds 12 US patents. He continues to publish and conduct technology development on numerous topics including intrusion detection and alarm correlation, privacy, malware analytics, active and software defined networks, and wireless security.



Guofei Gu is an associate professor in the Department of Computer Science & Engineering at Texas A&M University. Before joining Texas A&M, he received his Ph.D. degree in Computer Science from the College of Computing, Georgia Tech, in 2008. He is a recipient of 2010 NSF CAREER Award, 2013 AFOSR Young Investigator Award, Best Student Paper Award from 2010 IEEE Symposium on Security & Privacy (Oakland'10), Best Paper Award from 2015 International Conference on Distributed Computing Systems (ICDCS'15), and a Google Faculty Research Award. He is currently directing the SUCCESS (Secure Communication and Computer Systems) Lab at TAMU.