# Systematizing Attacks and Defenses in Software-Defined Networking: A Survey

Jinwoo Kim [1], Minjae Seo [2], Seungsoo Lee [2], Jaehyun Nam [2], Vinod Yegneswaran [2], Phillip Porras [2], Guofei Gu [2], and Seungwon Shin [2]

[1]Kwangwoon University
[2]Affiliation not available

October 30, 2023

## Abstract

Software-Defined Networking (SDN) has manifested both its bright and dark sides so far. On the one hand, it has been advocated by research communities and industry for its open nature and programmability. Every stakeholder, such as researcher, practitioner, and developer, can design an innovative networking service with a rich set of APIs and a global network view by escaping from the vendor-dependent control plane. On the other hand, its new architecture has introduced many security challenges that did not exist in the legacy environment. However, while new attacks and vulnerabilities within SDN have been steadily discovered, fewer efforts have been made to systematize the vulnerabilities from security aspects. In this paper, we aim to scrutinize prior literature that disclosed attack cases in SDN from an architectural perspective through identifying their root causes, penetration routes, and outcomes. Then, we conduct an in-depth yet comprehensive discussion of their underlying problems and introduce countermeasures proposed by researchers to mitigate those attacks. We believe that this study can contribute to revisiting various security problems around the current SDN architecture and envisioning a guideline for security research for SDN in the future.

# Systematizing Attacks and Defenses in Software-Defined Networking: A Survey

Jinwoo Kim, Minjae Seo, Seungsoo Lee, Jaehyun Nam,
Vinod Yegneswaran, Phillip Porras, Guofei Gu, and Seungwon Shin

***Abstract*—Software-Defined Networking (SDN) has manifested both its bright and dark sides so far. On the one hand, it has been advocated by research communities and industry for its open nature and programmability. Every stakeholder, such as researcher, practitioner, and developer, can design an innovative networking service with a rich set of APIs and a global network view by escaping from the vendor-dependent control plane. On the other hand, its new architecture has introduced many security challenges that did not exist in the legacy environment. However, while new attacks and vulnerabilities within SDN have been steadily discovered, fewer efforts have been made to systematize the vulnerabilities from security aspects. In this paper, we aim to scrutinize prior literature that disclosed attack cases in SDN from an architectural perspective through identifying their root causes, penetration routes, and outcomes. Then, we conduct an in-depth yet comprehensive discussion of their underlying problems and introduce countermeasures proposed by researchers to mitigate those attacks. We believe that this study can contribute to revisiting various security problems around the current SDN architecture and envisioning a guideline for security research for SDN in the future.**

***Index Terms*—Software-Defined Networking (SDN), OpenFlow, Security, Survey, Systematization of Knowledge**

## I. Introduction

Software-Defined Networking (SDN), no doubt, has governed the trend of the networking paradigm over the last decade. The concept of separating the control and data planes, first conceptualized by the 4D project [1], has proven to appealing to researchers seeking to overcome the limitations of traditional networks that were resistant to innovation. The centralized control plane—also commonly referred to as an *SDN controller*—has facilitated numerous innovations that were previously impossible with proprietary devices. The advent of programmable interfaces such as OpenFlow [2], killer applications such as FlowVisor [3], and Open vSwitch [4] have further fueled the widespread adoption of SDN in the industry. The deployment of SDN has been extensively studied, with early deployments on campus networks [5] and more recent deployments on large-scale networks such as WAN [6] and data-centers [7], [8].

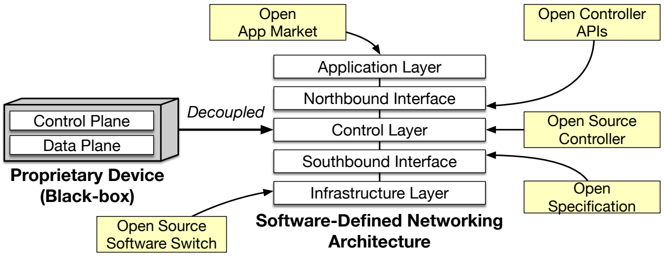


Fig. 1. Illustration of decoupling control-plane and data-plane from proprietary network devices.

Indeed, SDN has been a topic of growing interest in the field of networking in recent years, and it has brought numerous benefits to network operators, including improvements in 'security'; With the centralization of control provided by an SDN controller, operators have the ability to build native security systems that are more effective than traditional systems that rely on middle-boxes. The implementation of security systems with the capability for early detection and proactive mitigation [9] has been demonstrated through various proposals, such as botnet detection [10], DDoS mitigation [11], and network forensics [12]. Additionally, SDN has the ability to coordinate with middle-boxes by incorporating a service chain context, thereby addressing concerns about backward compatibility with legacy markets [13], [14].

However, it is worth noting that the new SDN architecture *opens* the door for layers that can be exploited by attackers (see Figure 1). This has led to an increase in the number of attack surfaces, as more stakeholders can now access previously inaccessible layers. For example, the deployment of malicious applications in app markets is made possible due to a lack of an ecosystem for establishing trusted relationships between app developers and operators [15]–[17]. The centralized architecture of the SDN controller is also found to be susceptible to simple application-level attacks due to its implementation as a general network operating system (NOS) [18]–[20]. In addition, the centralized control plane is at risk of saturation attacks [21]–[23] despite its ability to manage all connected switches. Despite over a decade of the existence of SDN, there are still no formal standards that provide detailed security guidelines, except for a simple description of SSL/TLS encryption in the OpenFlow specification [24].

Jinwoo Kim is with the School of Software, Kwangwoon University, Seoul, 01897, South Korea (e-mail: jinwookim@kw.ac.kr).

Minjae Seo is with the Graduate School of Information Security, KAIST, Daejeon, 34141, South Korea (e-mail: ms4060@kaist.ac.kr).

Seungsoo Lee is with the Department of Computer Science & Engineering, Incheon National University, Incheon, 22012, South Korea (e-mail: seungsoo@inu.ac.kr).

Jaehyun Nam is with the Department of Computer Engineering, Dankook University, Yongin, Gyeonggi-do, 16890, South Korea (e-mail: jaehyun.nam@dankook.ac.kr).

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

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

Seungwon Shin is with the School of Electrical Engineering, KAIST, Daejeon, 34141, South Korea (e-mail: claude@kaist.ac.kr).

TABLE I
PREVIOUS SURVEYS AND THEIR SHORTCOMINGS

| Previous Studies | Main Focus | Focused Layers and Interfaces | Shortcomings | Unveiling Root Cause | Analysis of Penetration Route |
|---|---|---|---|---|---|
| Kreutz et al. [25] (2013) | Overall security issues in SDN | Application and control layer | The study is limited in scope as it only addresses the security issues related to trust between SDN applications and controllers. | ✗ | ✗ |
| Scott et al. [26] (2015) | Overall security issues in SDN | Application and control layer | The study is deficient in the coverage of attack types, providing a limited perspective on the SDN security. | ✗ | ✗ |
| Alsmadi et al. [27] (2015) | Overall security issues in SDN | Application and control layer | The study's classification of attack types (e.g., STRIDE) is well-defined, but the defense criteria is inadequately defined, thereby limiting the scope of the research in terms of its practical applications. | ✗ | ✗ |
| Yan et al. [28] (2015) | Distributed Denial of Service (DDoS) attacks in SDN and cloud computing environments | Application and control layer | The study is narrowly focused on the analysis of a single attack, a DDoS attack, and its mitigation in a cloud computing environment. | ✗ | ✗ |
| Khan et al. [29] (2016) | Analysis of SDN topology discovery method and its threat | Application layer, control layer, and SDN interfaces | The study's primary focus is on the analysis of a specific SDN topology discovery method, neglecting other importance aspects of the security issues. | ✗ | ✗ |
| Shaghaghi et al. [30] (2020) | Security issues in SDN data plane | Infrastructure layer | The study is limited in its examination of attack and mitigation scenarios, only considering those on the SDN data plane. | ✗ | ✗ |
| Chica et al. [31] (2020) | Overall security issues in SDN | Application layer, control layer, infrastructure layer, and SDN interfaces | The study's proposed classification of attack criteria and taxonomy lacks novelty and detail. | ✗ | ✗ |
| Rauf et al. [32] (2021) | Northbound interface security issues in SDN | SDN northbound interface | The study is limited in its examination of the Northbound interface vulnerabilities, providing a limited perspective on the SDN security. | ✗ | ✗ |

In this paper, we aim to address the question of the security implications within the SDN architecture; *what security aspects are fundamentally lacking in the current SDN architecture?* To answer this question, we carry out a comprehensive survey of the security issues in SDN by examining relevant research published in top-tier journals and conferences within the domains of networks, security, and systems. We then propose a taxonomy for classifying SDN attacks into categories, considering their root causes, affected components, and common attack types. Subsequently, we examine existing countermeasures proposed by researchers to defend against these attacks. Through an in-depth analysis of the existing attacks and defenses, we highlight what aspects of SDN are architecturally vulnerable and identify areas that require further attention from security researchers in future studies.

**Contributions.** Our contributions and the overall paper road-map are summarized as follows:

- We present brief background knowledge for understanding the SDN architecture and its operations (§II).
- We introduce a taxonomy of features that define distinct aspects of SDN attacks and defenses in terms of (i) root causes, (ii) compromised components, (iii) attack surfaces, (iv) outcomes, and (v) defense types (§III).
- We review prior attack papers that aim to break security properties using vulnerabilities within SDN across *four* layers; (i) application, (ii) control, (iii) control channel, and (iv) infrastructure (§IV).
- We review existing countermeasure papers that mitigate the vulnerabilities addressed in §IV and introduce what techniques they adopt (§V).
- We conclude with a prediction of possible attack surfaces that may expose vulnerabilities and future research direction that can contribute to building a more secure SDN architecture (§VI).

### A. Comparison with Previous Studies

As shown in Table I, previous studies in the field of SDN security have several limitations, including a narrow scope, insufficient attack and defense definition, lack of analysis and classification. Our study has been conducted with the aim of addressing the limitations of previous SDN security research by taking a comprehensive viewpoint. We focus on the major root cause of both SDN attacks and defenses and carry out an in-depth analysis of attack penetration. This gives security researchers a deeper understanding of the problem and enables them to proactively defend against attacks. Our study provides a much-needed holistic view of overall security issues in SDN, including a thorough examination of the major root cause and attack penetration route.

It is noteworthy that while SDN has been applied to improve network security in various domains such as IoT [33] and moving target defense [34], our focus is solely on analyzing attacks targeting the SDN architecture.

## II. Background

### A. What is Software-Defined Networking (SDN)?

In traditional networks, it is inherently challenging to insert new functions into the device without specialized knowledge or vendor cooperation since the control plane and data plane are often embedded within a proprietary network device [2]. To overcome this fundamental problem, Software-Defined Networking (SDN) presents a new paradigm that emphasizes

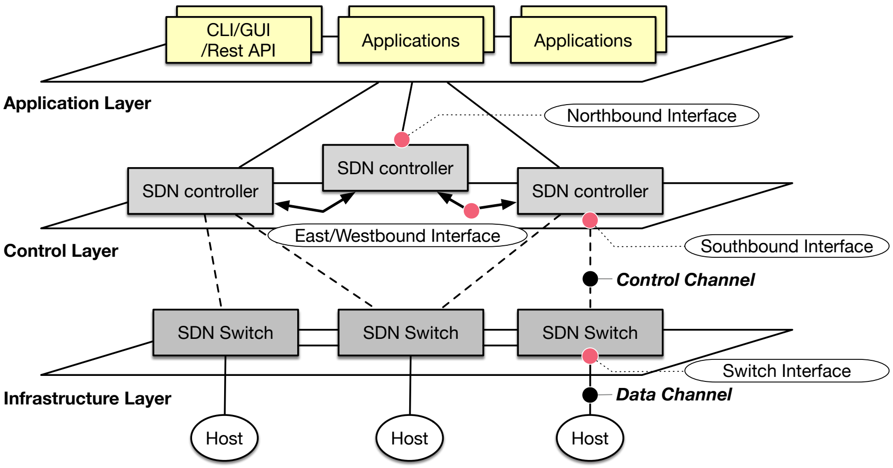


Fig. 2. SDN layers, components, channels, and interfaces.

decoupling the control plane from the data plane with a logically centralized controller operated on high-performance commodity hardware.

Figure 2 shows the overall architecture of SDN. Various SDN applications (or apps) are running on the *application layer* (also known as a management layer), which are designed to operate network management. In the middle, the *control layer* consists of one or multiple SDN controllers that control the underlying forwarding devices and manage the centralized network view. At the bottom, the *infrastructure layer* is composed of distributed forwarding devices that directly handle incoming packets. In particular, the control layer and the infrastructure layer communicate with each other through the *control channel*, and the hosts send and receive packets to the switches through the *data channel*. In addition to these channels, the controller has three different types of interfaces: (i) the *northbound interface* to communicate with the applications, (ii) the *east/westbound interface* to sync states with the neighboring controllers, and (iii) the *southbound interface* to manage switches.

### B. SDN Controller and Application

SDN controllers on the control layer are often referred to as network operating systems (NOS) because most cutting-edge controllers consist of basic control software required to operate and manage an overall network, analogous to a legacy operating system (e.g., Linux). Also, it provides a global view of the entire network for the SDN apps by simplifying the complex control logic details. Most controllers typically involve the basic core modules and interfaces necessary for topology detection and traffic management as shown in Figure 3.

While the detailed implementations of controllers might be different, they commonly include the following modules: a topology manager along with a link discovery service, a host tracking service, and switch managers, which maintain the latest topology information that involves the discovery of each network element. The two other core modules are a storage service and a flow manager. The storage service stores all the necessary network information and provides it to the SDN apps whereas the SDN apps can define and modify flow rules in switches through the flow manager leveraging the southbound interface.

Moreover, one noticeable reason why SDN has been popularly used is *programmability*. For example most SDN controllers (e.g., NOX [35], Maestro [36], Onix [37], Floodlight [38], Beacon [39], ONOS [40], OpenDaylight [41]) provide operators with APIs. Thus, operators can develop diverse network applications with less expensive cost and optimize them according to their policies.

### C. SDN Switch and OpenFlow

The most widely used southbound interface between the control layer and the infrastructure layer is OpenFlow [24], and it defines commands and behaviors that enable the controller to perform fine-grained and dynamic policy enforcement in the OpenFlow-enabled switches. The switch maintains a number of flow tables, which manage a set of flow rules. Basically, when an incoming packet arrives on the switch and has no matching flow rule entry, the switch sends a PACKET_IN message, including the partial information of the packet to the controller. Then, the controller and its SDN apps decide how to handle the packet and send a flow rule to the switch through a FLOW_MOD message. In addition to handling packets, the switch counts up the total number of packets and bytes per each flow rule. The values can be leveraged by the controller or SDN apps later to provide better network quality of service.

## III. Systemization Taxonomy

Here, we introduce our taxonomy to classify existing attack cases and countermeasures. For each criterion, we elaborate on key reasons behind choosing them with discussion of security challenges. We refer the readers to Figure 3 for better understanding of the threat model of taxonomy in the SDN architecture.

### A. Root Cause

The *root cause* aims to analyze why the proposed attack scenarios are feasible within the SDN components. From our survey of prior literature, we classify 9 major root causes that have been regarded as key problems that security researchers have put great attention.

*1) Lack of NBI Authorization:* This criterion indicates the absence of an authorization measure in SDN northbound interfaces. Despite their critical impact on the entire network operations in case of misuse by malicious apps, they are not properly protected from malicious intents [18]–[20] or human errors [42].

*2) Lack of SBI Authorization:* The southbound interface is the most critical boundary that immediately affects network forwarding behavior on the data plane or visibility on the control plane. Given that, it should be secured from malicious actions. However, there is no proper authorization to prevent abuse by a malicious component, creating more security concerns [17], [43].

*3) Lack of Control Event Integrity:* Most SDN controllers maintain a service chain that dictates how an internal control event is processed by which order of applications or core modules [40], [41]. When processed, their integrity should be guaranteed to preserve the original messages or finish whose chain sequence without unintended modifications.

*4) Lack of Control Message Integrity:* An SDN controller and network devices are connected via a control channel through which some critical messages will be sent/received. Thus, a secure channel (e.g., TLS/SSL) is recommended [24] to avoid such a situation. However, in the real world, it is rarely employed due to its performance issue, giving an attacker a chance to monitor and manipulate control messages [44].

*5) Lack of Application Authentication:* It is clear that verifying the reliability of an application developer is an indispensable option in maintaining a safe and secure SDN ecosystem. However, in our analysis, we recognize that most popular SDN controllers do not support application authentication, implying that a malicious application, which is disguised as a benign application, could be installed without any restrictions.

*6) Lack of Switch and Host Authentication:* A southbound interface, such as OpenFlow, does not specify any authentication measure when establishing control channels from switches to a controller [24], and contemporary SDN controllers also do not support data-plane authentication for switches and hosts.

*7) Lack of Controller Resource Control:* The design philosophy of an SDN controller is alike the traditional operating system in that both need to manage a variety of user-level applications concurrently executed with shared resources. However, the lack of this necessary element in several ancestor controllers led to harmful attack scenarios [18], [45].

*8) Side Channel:* The core design philosophy of SDN, decoupling the control plane (i.e., controller) from the data plane, requires a communication channel between these two planes, exposing a new attack surface. For example, an attacker can fingerprint a channel between controller and switches to leak confidential information [46], [47].

*9) Implementation Flaw:* While there is a clear standard reference for SDN (e.g., OpenFlow [24]), it does not mean that the implementations of such references are always clean, no bug or no critical errors. In this context, researchers have investigated if SDN implementations (e.g., open-source SDN controllers) include any critical program bugs or holes, and they have revealed critical implementation problems that can cause serious security issues [42].

### B. Penetration Route

We observe that typical attacks in SDN require penetrating SDN architecture internals. We model this concept by defining a *penetration route* where an attacker's message or event is propagated. A typical penetration route needs at least one *source* in order to affect a *target* through a sequence of the following components and interfaces:

*1) Application:* Malicious SDN apps have been a popular stepping stone to penetrate SDN controllers and switches [15], [18]–[20], [48], [49], similar to the way an Android malware infiltrates a user's mobile device [50].

*2) Northbound Interface (NBI):* We scope the NBIs into all accessible interfaces of a controller from SDN apps, such as system APIs (e.g., Java native methods, Linux system calls), controller core services [51], [52], peer app services [48], [52], and REST APIs [42].

*3) Controller:* An SDN controller is the most important target for attackers and defenders. It can be targeted by any other components.

*4) Southbound Interface:* A southbound interface is a boundary where a controller and switches communicate each other. So, it can be abused to affect controller and switch operation.

*5) Switch:* Switches can act as either a reflector that sends control packets invoked by hosts or an attack source if they are controlled by an attacker. For example, it is widely known that commodity switches can be compromised due to switch firmware vulnerabilities [53], [54].

*6) Switch Interface:* It refers to the communication point between hosts and SDN switches. This is the only way to inject malicious packets from compromised hosts.

*7) Host:* Hosts can be a variety of entities, such as physical machines, VMs (Virtual Machines) and even containers that participate in the target SDN network. Attackers can compromise one of those hosts to use them as an attack source.

Note that SDN apps and switches can be either source and target.

### C. Attack Outcome

We now define six goals attackers want to achieve:

*1) Information Leakage – Architecture:* The architectural information of SDN networks should be kept confidential as it is primarily related to the control path between switches and a controller. However, prior studies demonstrated that such information can be leaked by observing various physical factors (e.g., latency) [21], [55], [56].

*2) Information Leakage – Configuration:* It represents the situation when network configurations are leaked. The configuration information may include running applications upon a controller [57] or network policies [46], [47].

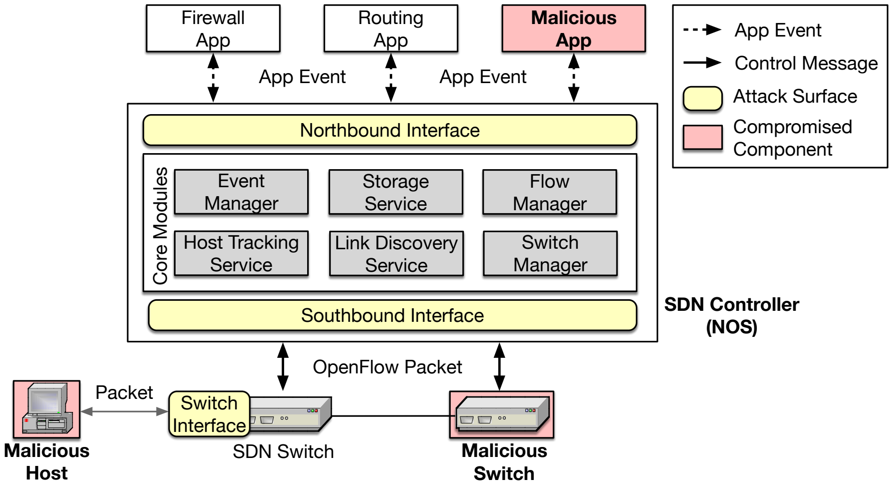


Fig. 3. The internal architecture of an SDN controller and our threat model.

*3) Denial of Service – Controller:* It refers to all circumstances when a controller is malfunctioning as due to penetration performed by attackers (e.g., harmful API invocation [18] or control channel saturation packets [21], [58], [59]).

*4) Denial of Service – Switch:* This criterion indicates the result when a switch is unavailable or its performance is downgraded due to external entity [54] or data plane saturation attacks [21], [60].

*5) Inconsistent Network State:* In SDN, it is necessary to strictly maintain a consistent network view between a controller and switches. While this is important given that SDN applications rely on the current network states of the data plane, some inconsistent cases have been discovered due to erroneous inputs [42], [61].

*6) Network Policy Evasion:* It refers to the failure states of security policies for application-level access control [48] or traffic blocking [17], [49] when they violate network invariant.

### *D. Defense Type*

To defend SDN components from attacks and remove their vulnerabilities, diverse countermeasures have been proposed so far. With a thorough survey and analysis, we classify them into the following 6 defense types:

*1) Control Plane Extension:* It indicates the work that extends prior architecture, implementations, and functionality of the control plane with security features. For example, it can be adding security modules to existing controllers [15], [17] or rebuilding new and architecturally secured controllers [18], [62].

*2) Data Plane Extension:* This implies the case when the functionality of an existing SDN data plane is extended to improve them with security add-on features. It can be either patching switch modules [63], [64] or expanding OpenFlow protocol capability [23].

*3) Pen Testing:* As the foundation of SDN operations is the software-defined logic, reliability heavily depends on if the implementations of SDN components (e.g., controllers and switches) have no flaws. Security researchers have devised tools that automatically generate test cases using black-box fuzz-testing for controllers [65]–[67] or protocol conformance testing for switches [68].

*4) Program Analysis:* This method examines program behavior to find flaws that abuse security-sensitive APIs or violate network policies (invariant). Prior studies have utilized diverse program analysis techniques, such as static analysis looking into control flows [69], [70] or dynamic instrumentation investigating execution traces [48], [71].

*5) API Monitoring:* To facilitate application development, a variety of APIs have been developed in contemporary SDN controllers. However, there is no built-in security measure that audits application behavior when accessing APIs pertaining to critical resources. Thus, one way to repair such security holes would be to monitor APIs invoked by applications [15]–[17].

*6) Message Monitoring:* SDN components typically interact each other with a series of messages which are either controller messages (events) or OpenFlow messages (control packets). Given the importance of such information, operators should guarantee that those messages are not manipulated by a malicious app or compromised switch. For this, it is possible to build a shim layer between SDN components and check the message validity [60], [72], [73].

### *E. Classification Method*

In our study, we group the SDN attacks and defenses into four layers: the application layer, the control layer, the control channel, and the infrastructure layer. This categorization is based on the observation that the majority of SDN attacks and defenses are targeted towards these layers. However, it is important to note that correspondence between the penetration routes and these layers may not always be exact. This means that the target of a penetration route could differ from the corresponding layer.

## IV. SDN Attack Classification

In this section, we present major categories of attacks and vulnerabilities that have been discussed in academia. Table II shows the summary of systematization for disclosed SDN

TABLE II
SYSTEMATIZATION OF SDN ATTACKS.
○ SOURCE → PENETRATION DIRECTION ● ROOT CAUSE/TARGET/ATTACK OUTCOME ◑ SOURCE AND TARGET

| | | | Root Cause | | | | | | | | | Penetration Route | | | | | | | Attack Outcome | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Layer** | **Sec.** | **Attack** | Lack of NBI Authorization | Lack of SBI Authorization | Lack of Control Event Integrity | Lack of Control Message Integrity | Lack of Application Authentication | Lack of Switch/Host Authentication | Lack of Controller Resource Control | Side Channel | Implementation Flaw | Application | Northbound Interface | Controller | Southbound Interface | Switch | Switch Interface | Host | Information Leakage - Architecture | Information Leakage - Configuration | Denial of Service - Controller | Denial of Service - Switch | Inconsistent Network State | Network Policy Evasion |
| Application | §IV-A1 | System API Abusing<br>[18], [51] | ● | | | | | | | | | ◑ | ←<br>→ | ● | | | | | | | ● | | | |
| | | Controller Resource Exhaustion<br>[18], [61] | ● | | | | | | ● | | | ◑ | ←<br>→ | ● | | | | | | | ● | | | |
| | | Event Hijacking<br>[19], [51], [66] | | | ● | | | | | | | ◑ | ←<br>→ | ● | | | | | | | ● | | | |
| | | Event Unsubscription<br>[19], [51], [66] | ● | | | | | | | | | ◑ | ←<br>→ | ● | | | | | | | ● | | | |
| | | Rootkit Injection and Hiding<br>[20] | | | | | ● | | | | | ◑ | ←<br>→ | ● | | | | | | ● | | | | ● |
| | §IV-A2 | Unhandled Event Injection<br>[70] | | | ● | ● | | | | | | ● | ← | ← | ← | ← | ← | ○ | | | | | | ● |
| | | App Race Condition<br>[45] | | | | | | | | | ● | ● | ← | ← | ← | ← | ← | ○ | | | ● | | | |
| | | App Remote Code Execution<br>[74] | | | ● | ● | | | | | | ● | ← | ← | ← | ○ | | | | ● | ● | | | |
| | | Cross-app Poisoning<br>[48] | ● | | | | ● | | | | | ◑ | ←<br>→ | ←<br>→ | | | | | | | | | ● | ● |
| Control | §IV-B1 | PACKET_IN Message Flooding<br>[21], [58], [59], [64] | | | | | | | ● | | | | | ● | ← | ← | ← | ○ | | | ● | | | |
| | | READ_STATE Message Flooding<br>[63] | | | | | | | ● | | | | | ● | ← | ← | ← | ○ | | | ● | | | |
| | §IV-B2 | Topology Information Removal<br>[18] | ● | | | | | | | | | ○ | → | ● | | | | | | | ● | | ● | |
| | | Link Fabrication Attack<br>[43], [60], [75] | | | | ● | | ● | | | | | | ● | ← | ← | ← | ○ | | | | | ● | |
| | | Host Identifier Spoofing<br>[43], [60], [75], [76] | | | | ● | | | | | ● | | | ● | ← | ← | ← | ○ | | | | | | ● |
| | §IV-B3 | Corrupted REST API Rule Injection<br>[42] | | | ● | | | | | | ● | ○ | → | ● | | | | | | | | | | ● |
| | | Malformed Configuration Injection<br>[61] | | | ● | | | | | | | ○ | → | ● | | | | | | | ● | | | ● |
| | | App Configuration Manipulation<br>[19], [66] | | | ● | | | | | | | ○ | → | ● | | | | | | | ● | ● | | ● |
| | §IV-B4 | Malformed OpenFlow Packet Injection<br>[66], [67], [77] | | | | ● | | ● | | | | | | ● | ← | ○ | | | | | ● | | | |
| | | Malformed LLDP Packet Injection<br>[78] | | | | ● | | ● | | | | | | ● | ← | ← | ← | ○ | | | ● | | | |
| | | Abnormal Protocol Behavior Injection<br>[65], [67] | | | | ● | | ● | | | | | | ● | ← | ○ | | | | | ● | | | |
| Control Channel | §IV-C1 | Slow Path Fingerprinting<br>[21], [55] | | | | | | | | ● | | | | | | ● | ← | ○ | ● | | | | | |
| | | Policy Fingerprinting<br>[46], [47], [79], [80] | | | | | | | | ● | | | | | | ● | ← | ○ | | ● | | | | |
| | §IV-C2 | Control Channel MitM<br>[51], [67] | | | | ● | | ● | | | | | | | | ● | ← | ○ | | ● | | | | ● |
| | | In-band Channel Fingerprinting<br>[56] | | | | | | | | ● | | | | | | ● | ← | ○ | ● | | | | | |
| | | In-band Channel Flooding<br>[56] | | | | | | | | ● | | | | | | ● | ← | ○ | | | ● | | | |
| | | App Fingerprinting<br>[57] | | | | | | | | ● | | | | | | ● | ← | ○ | | ● | | | | |
| | | Topology/Protocol Fingerprinting<br>[81] | | | | | | | | ● | | | | | | ● | ← | ○ | | ● | | | | |
| Infrastructure | §IV-D1 | Flow Table Overloading I<br>[51] | ● | ● | | | ● | | | | | ○ | → | → | → | ● | | | | | | ● | | |
| | | Flow Table Overloading II<br>[21], [58], [60], [63], [82] | | | | | | | | ● | | | | ←<br>→ | ←<br>→ | ←<br>● | ← | ○ | | | | ● | | |
| | §IV-D2 | Switch Firmware Abuse Attack<br>[51] | | | | | | | | | ● | ○ | → | → | → | ● | | | | | | ● | | |
| | | Malformed Control Message Injection<br>[51] | | | ● | | | ● | | | | ○ | → | → | → | ● | | | | | | ● | | |
| | | Dynamic Tunneling Attack<br>[17], [83] | ● | | | | ● | | | | ● | ○ | → | → | → | ● | | | | | | | | ● |
| | | Buffered Packet Hijacking Attack<br>[49] | | | ● | ● | | | | | | ○ | → | → | → | ● | | | | | ● | | ● | ● |
| | | Switch Race Condition<br>[84]–[86] | | | | | | | | | ● | ←<br>→ | ←<br>→ | ←<br>→ | ←<br>→ | ←<br>● | ← | ○ | | | | | | ● |
| | §IV-D3 | Switch Remote Code Execution<br>[54] | | | | | | | | | ● | | | | | ● | ← | ○ | ● | ● | | ● | | ● |
| | | Middlebox Tag Manipulation<br>[87] | | | | | | ● | | | | | | | | ● | ← | ○ | | | | | | ● |

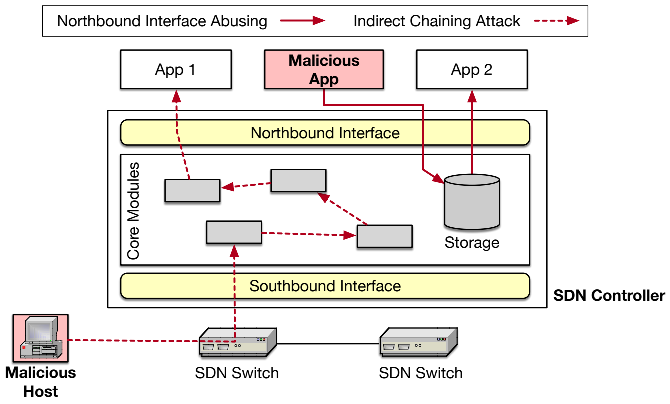


Fig. 4. Penetration routes of application-layer attacks: Northbound Interface Abusing (§IV-A1) and Indirect Chaining Attack (§IV-A2).

attacks by the taxonomy defined in §III, enumerated with the four SDN layers. Finally, we summarize our findings and insights for each layer.

### A. Application Layer

*1) Northbound Interface Abusing:* SDN apps are typically able to access all northbound APIs of a controller by default. While this design choice comes from the fact that SDN apps will not have malicious intents or critical bugs, it cannot be fully guaranteed in practice.

For example, a malicious app can easily abuse *system APIs* directly related to controller run-time operation. Rosemary [18] shows that it is possible to illegally terminate Java-based SDN controllers (e.g., Floodlight, ONOS, OpenDayLight) by calling `System.exit()`. Yoon *et al.* [51] introduce the system time manipulation attack that modifies time variables, causing switches to be disconnected from a controller. In addition, some APIs can be exploited to exhaust system resources. Rosemary [18] demonstrates that a malicious app can allocate large-sized data structures to exhaust controller memory. AIM-SDN [61] shows that it is possible to produce many configuration entries to flood a controller storage.

A more sophisticated attack abusing northbound interfaces is to disrupt the *service chain* of a controller. For example, a malicious app can hijack an *event* between SDN apps to manipulate its payload [66] or drop it [19], [51] (i.e., a control message hijacking attack). Further, a malicious app can disable event subscriptions of benign apps so that they cannot receive subscribed events (i.e., control message unsubscription attack attack) [19], [51], [66].

It is demonstrated that a malicious app can hide due to the absence of an access control mechanism in SDN controllers. Ropke *et al.* [20] show that an SDN rootkit app can remove its app ID from a controller storage. The rootkit then can make a covert channel to steal sensitive information (e.g., configurations) from a target controller.

*2) Indirect Chaining Attack:* Contemporary SDN controllers are built based on an event-driven platform where apps and core modules interact using events, making a complicated event chain between components. Under the shadow of the chain, an attacker can design a *indirect chaining attack*, which cannot be detected easily.

TABLE III
POPULAR OPEN SOURCE SDN CONTROLLER REPOSITORIES.
(● STRONG ◑ MODERATE ○ WEAK)

| Controller | Repository | Code Review |
|---|---|---|
| ONOS | https://github.com/opennetworkinglab/onos | ● |
| OpenDaylight | https://github.com/opendaylight | ● |
| POX | https://github.com/noxrepo/pox | ◑ |
| Ryu | https://github.com/faucetsdn/ryu | ◑ |
| Faucet | https://github.com/faucetsdn/faucet | ◑ |
| Floodlight | https://github.com/floodlight/floodlight | ○ |
| NOX | https://github.com/noxrepo/nox | ○ |

For example, an attacker can abuse a *host-to-application* event chain, denoting a case when events generated from hosts affect an application's behavior. EventScope [70] presents a cross-plane attack that focuses on "unparsed" events by applications. They show that an attacker can trigger a malformed HOST_ADDED event containing an invalid IP address (e.g., 10.0.0.256). Subsequently, it makes a controller fail to install a flow rule as it can not parse the event. Thus, the attacker can bypass a security policy. ConGuard [45] demonstrates that an attacker can deliberately raise a harmful race condition on shared variables using TOCTOU (Time-Of-Check to Time-Of-Use) attacks. For instance, SWITCH_JOIN and SWITCH_LEAVE represent events when a switch is connected and disconnected, respectively. A `dpid` variable is created when the former event is detected, while the variable is removed for the latter event. Suppose those events are produced from the data plane intermittently. In that case, it is possible to try accessing the shared variable after being removed, causing a null point exception.

On the other hand, a *switch-to-application* event chain can be used to penetrate a controller. Xiao *et al.* [74] reveal that a compromised switch can inject a malicious payload into an OpenFlow message to execute arbitrary commands on SDN apps or core modules. It exploits the fact that the payload of an OpenFlow message is often used in controller internal components, enabling an attacker to extract the configuration information or remotely terminate a target controller.

An attacker can abuse an *application-to-application* event chain. Specifically, ProvSDN [48] introduces a cross-app poisoning attack where a malicious app poisons storage to affect the decision of other apps. For example, suppose that a malicious app injects a PACKET_READ event into a controller by spoofing a victim's MAC address with the attacker's. The controller then updates a host-to-location pair maintained by a host tracking service from the packet. As a result, a forwarding application installs a rule that forwards the victim's traffic to the attacker, violating a security policy.

Figure 4 illustrates example penetration routes for application-layer attacks.

**Summary.** Overall, there are numerous attack scenarios in which SDN apps are either sources or targets of penetration. The root cause contributing to these attacks is the inadequate security measures employed in the northbound interfaces of the controllers, resulting in a high frequency of denial of service (DoS) attacks on the controllers.

**Why is the application layer vulnerable?** As stated, most known SDN controllers do not have the necessary security mechanisms or sanitization approaches in place, which leaves them vulnerable to attacks from malicious or buggy SDN apps. This is due to the fact that during the early stages of SDN development (around 2009), developers primarily focused on implementing new features and improving performance, rather than considering security. Additionally, the implementation of diverse northbound interfaces in SDN controllers to support app functionality resulted in inadequate protection, making these interfaces susceptible to attacks from malicious or malfunctioning SDN apps.

**Can an attacker poison SDN app stores?** At the beginning of the SDN era, it was anticipated that public SDN app stores [51] would be popular, akin to the Docker Hub. This would have made it easy for an attacker to deploy malicious SDN apps. However, as recent programmable networking trends have moved towards the data plane, SDN app stores have failed to gain widespread adoption. For instance, HP Enterprise (HPE) had established an SDN app store for their controller (HPE VAN SDN controller), but it has since been discontinued. Consequently, the potential for attackers to utilize SDN app stores as a means of distributing malicious apps has been diminished.

**Can an attacker poison open-source repositories?** One potential avenue for deployment of malicious SDN apps is through code repositories (e.g., GitHub). At first glance, poisoning them may be difficult because most code repositories only allow trusted contributors to upload code. However, an attacker can spoof the identity of a trusted contributor or alter commit timestamps [88]. Therefore, the security of open-source controllers heavily relies on the developer's code reviews. We analyzed the threat model by surveying controller repositories. As shown in Table III, popular controllers such as ONOS and OpenDaylight conduct strict code reviews through discussion with several developers before merging into the main branch. However, other controllers may lack a comprehensive review process, relying on a single developer's evaluation or having no review process at all, which leaves them more susceptible to the deployment of malicious code.

**Is there a trend for application-layer attacks?** Initially, attackers focused on exploiting vulnerabilities in the northbound interfaces of SDN controllers due to their relative ease of accessibility (i.e., northbound interface abusing). However, as SDN controllers have evolved and become more sophisticated, offering a wider range of features, their internal code-base has become more complex, making it increasingly challenging for developers to anticipate the outcomes of internal execution. This has resulted in the exposure of another potential attack surface (i.e., indirect event chaining).

### *B. Control Layer*

*1) Reflective DDoS Attack:* Whereas the separation of control and data planes enables the management of all switches, it has been suggested that the centralized control plane is architecturally weak. Specifically, a single controller can be overloaded when switches request lots of control messages. Exploiting this fact, an attacker can mount *reflective DDoS attacks* that use SDN switches as reflectors to saturate control channels. This attack can degrade network performance significantly and even take down the control plane.

Shin and Gu [21] propose a concept of the reflective DDoS attacks, abusing OpenFlow PACKET_IN messages. They suggest that attackers can send a series of packets with different headers to trigger table-mismatch, making a target switch generate many PACKET_IN messages to a controller. FloodDefender [59] shows that the PACKET_IN flooding attacks overload the CPU utilization of a target controller. SWGuard [58] further employs a probing method that observes round-trip-times (RTTs) to learn which match fields trigger PACKET_IN messages. Besides, many other works [22], [23], [60], [64] are motivated by the DDoS attacks due to its serious impact.

In addition, an attacker can exploit READ_STATE messages that are used for collecting switch statistics to exhaust controller resources. If a malicious host conducts the PACKET_IN flooding attack, it subsequently makes a target switch install many rules. The more rules are installed, the more resources are needed to collect READ_STATE messages from the target controller. DevoFlow [63] analyzs this problem from a performance point of view by evaluating it on hardware OpenFlow switches.

*2) Topology View Poisoning:* It is crucial to keep a consistent view between a controller and switches because the control plane is separated and physically distant from the data plane. For this purpose, most SDN controllers typically maintain the storage that contains the current *topology view* of the data plane, such as link status and host information. As applications running on a controller refer to the view before making a decision, the integrity of the storage should be kept strictly. However, researchers show that it is possible to compromise the topology view by exploiting the *link discovery service* and *host tracking service* of a controller (§II-B).

The link discovery service is used to learn links on the data plane. First, it instructs a switch to broadcast LLDP (Link Layer Discovery Protocol) packets into neighbors with PACKET_OUT messages. When a neighbor switch receives the LLDP packet, it sends a PACKET_IN message to a controller, and then the link discovery service recognizes a link between those switches.

The problem is that most controllers neither restrict usage of APIs that affect the service nor investigate whether those LLDP packets come from a real switch. This vulnerability allows attackers to manipulate link information from various layers. On the application layer, Rosemary [18] demonstrates that a malicious app can illegally remove link information of the storage. On the infrastructure layer, TopoGuard [43] and SPHNIX [60] show that an attacker can inject fake link

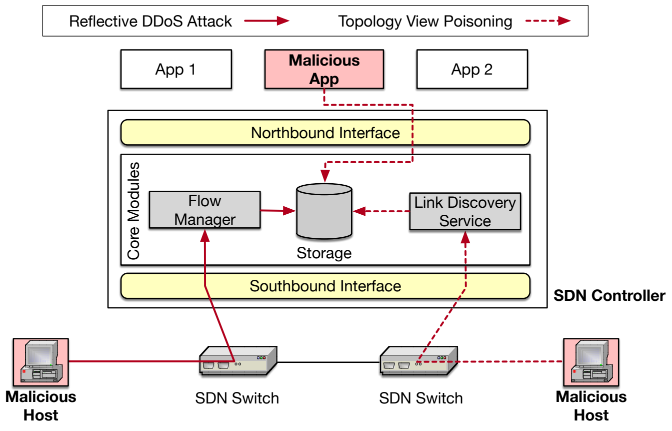


Fig. 5. Penetration routes for control-layer attacks: Reflective DDoS Attack (§IV-B1) and Topology View Poisoning (§IV-B2).

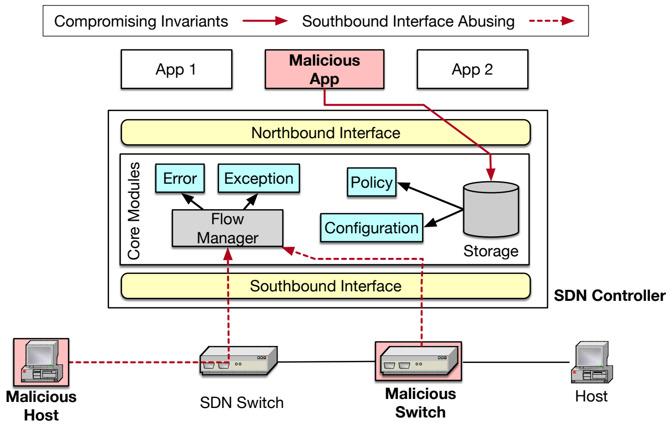


Fig. 6. Penetration routes for control-layer attacks: Compromising Invariants (§IV-B3) and Southbound Interface Abusing (§IV-B4).

information by relaying LLDP packets between two malicious hosts. While TopoGuard [75] proposes a defense that distinguishes actual link events based on precondition (e.g., PORT_DOWN or PORT_UP), TopoGuard+ proposes a port amnesia attack that bypasses the defense through artificially generating fake PORT_DOWN events (§V-B3).

The host tracking service is in charge of binding host identifiers (i.e., MAC and IP addresses) with current locations (i.e., the ports connected to switches). It updates the host location based on the most recently detected PACKET_IN message. However, since this does not verify if binding is valid, it gives an attacker a chance to disguise herself with the existing host information.

TopoGuard [43] and SPHNIX [60] introduce a host location hijacking attack, which poisons the service with spoofed host identifiers. For instance, when an attacker's host sends a packet that spoofs the victim's IP address, the service updates the victim's location to the attacker's. A controller believes that the victim migrates to the new location; thus, it redirects the victim's traffic to the attacker. In addition, TopoGuard+ [75] presents a port probing attack that periodically probes victim status and attempts to take the victim's binding when it goes offline. SecureBinder [76] introduces a similar attack, called a persona hijacking attack against DHCP.

Figure 5 shows examples of penetration routes for the two categories mentioned above.

*3) Compromising Invariants:* Other important assets that controllers should keep integrity are *invariants* of the control and data plane. The invariants refer to the intents of network operators, such as network policies and controller configurations, that network operators want to enforce in managing SDN networks.

It is shown that data-plane invariants can be compromised by poisoning a controller storage. AudiSDN [42] demonstrates that malformed input messages delivered to REST APIs can be translated into incorrect flow rules. For example, suppose an operator mistakenly writes the `tcp_dst` field without specifying the `ip_proto` field, which is a prerequisite for using TCP fields when making an OpenFlow FLOW_MOD message. The Floodlight SDN controller [38] updates its storage with this rule and attempts to synchronize with a target switch. However, the switch rejects the rule because it lacks the prerequisite field. The storage in the Floodlight controller keeps the rule unless an operator removes it; thus, this sequence repeats infinitely, thereby consuming controller resources significantly. AIM-SDN [61] introduces a similar attack where a malicious app directly corrupts a storage entry using northbound APIs, causing the infinite loop.

On the other hand, control-plane invariants can be corrupted to affect network performance. For example, there is a controller configuration that dictates how an SDN app handles control events. The `fwd` app [89] of the ONOS controller allows an operator to configure a `packet_out_only` option, which instructs a switch to forward packets without rule installation. However, the option can be abused to make all packets be sent to a controller as demonstrated by Lee *et al.* [19], [66]. This reduces the performance of SDN networks significantly.

*4) Southbound Interface Abusing:* Since the control layer needs to handle a variety of southbound interfaces, unexpected implementation bugs may exist. Here, an attacker can abuse implementation holes of southbound interfaces to put a target controller into an unpredictable state.

For example, many SDN testing tools reveal that attackers can create malformed control messages that do not follow OpenFlow protocol specification [24]. Shalimov *et al.* [77] propose a method that tests if controllers process malformed OpenFlow messages. For example, when an incorrect `length` value is injected into an OpenFlow header, a target controller crashes. DELTA [66] shows that manipulating OpenFlow headers with a randomized value causes a target controller to disconnect the connection from a switch. BEADS [67] presents many similar attack cases in several controllers by fully randomizing all possible OpenFlow headers and message fields using fuzz-testing.

Also, it is shown that LLDP packets are good targets to trigger exceptional cases on southbound interfaces. Marin *et al.* [78] propose reverse loop and topology freezing attacks. The former exploits the fact that a controller typically probes an opposite link when receiving a LLDP packet whose LINK_TYPE field is 0x01. By transmitting such LLDP packets, a target controller falls into generating probe packets indefinitely, which causes resource exhaustion. The latter is the case when an attacker injects fake links originated from the same port. As the link discovery service in Floodlight considers it as a broadcast port, it is removed from the topology view. However, a forwarding app tries to read the unavailable link without recognition, which triggers a null pointer exception.

On the other hand, one may inject abnormal protocol behavior. Specifically, ATTAIN [65] discovers that dropping OpenFlow messages can cause denial-of-service on a target network, as a controller cannot install any flow rule. BEADS [67] reveals that dropping, replaying, and delaying OpenFlow messages can make a controller lose connection from switches.

Figure 6 displays examples of penetration routes for the two categories mentioned above.

**Summary.** The control layer of SDN is highly vulnerable to injection attacks, which involve the utilization of fake or invalid information or protocol messages. These attacks frequently stem from the lack of integrity checks on messages or events, leading to controller DoS and policy failure.

**Why is the control layer vulnerable?** In SDN, a controller has a global view of the entire network, with the help of its logically centralized architecture, and this design philosophy provides many benefits in managing underlying network devices flexibly and efficiently. However, this design philosophy also presents several challenges, including the issue of a single point of failure (DDoS attacks against a controller) and consistency issues (poisoning topology information and network policies). As such, this centralized design of SDN is a *double-edged sword*; thus, it is important to design SDN controllers with fault tolerance and the ability to verify updated states in order to mitigate these risks and fully realize the benefits of a centralized architecture.

**Is a controller itself secure?** Looking into the controller internal, the architecture of vanilla SDN controllers was poorly designed from the view point of traditional operating systems. As highlighted by Shin *et al.* [18], the tight coupling of core module processes with SDN apps results in the termination of all core modules in the event of a malicious app's termination. Additionally, the bundling of several sub-modules in existing core modules presents an unnecessary attack surface to attackers, even if they are not utilized. Consequently, a redesign of the internal architecture of controllers is necessary to ensure robustness and enhance security.

**Is there a trend for control-layer attacks?** The initial threat landscape for SDN controllers was characterized by reflective DDoS attacks, and later, topology poisoning attacks were proposed. Due to the centralized control plane, these attack vectors are relatively simple to execute. As SDN controllers became more complex in size and functionality, protocol implementation bugs and policy consistency issues were also identified. Our analysis corresponds to the real-world vulnerability report trend as well. As illustrated in Figure 7, the number of Common Vulnerabilities and Exposures (CVEs) for popular controllers has increased over the years. It is clear that more vulnerabilities have been observed in ONOS, compared to the others (i.e., OpenDaylight, Floodlight) in recent years due to the addition of various features, making it more complex and hence more susceptible to vulnerabilities.

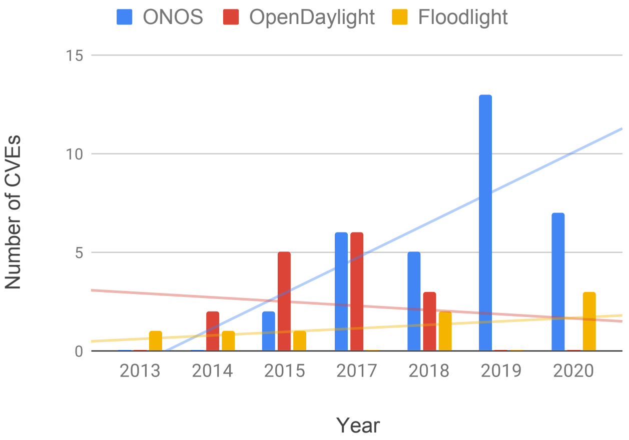


Fig. 7. Number of CVEs reported per year for popular controllers (Note that the lines denote trend lines.). We omit the year where no CVEs are reported. A full list of the surveyed CVEs is available at [90].

### C. Control Channel

*1) Control Path Delay Measurement:* Researchers have paid attention to the fact that a unique forwarding behavior of SDN switches can leak useful information to attackers. For example, Shin and Gu [21] and Bifulco *et al.* [55] propose that attackers can infer whether a target switch is SDN-enabled or not by measuring latency differences created by table mismatch. A switch on the data plane needs to remotely contact to the SDN controller when there is no matched table entry for incoming packets; the switch sends a PACKET_IN message to the controller and it subsequently receives a FLOW_MOD message that instructs flow installation. The incoming packets are blocked in queue while waiting for the OpenFlow procedure, and they are subsequently forwarded as soon as the switch installs a flow rule. This is the moment that makes end-users experience high-latency for a first packet, as the packet takes a control path, aka *slow path*.

By analyzing the timing difference in depth, attackers can fingerprint more detailed information for a target network. Sonchack *et al.* [46] demonstrate it is possible to measure RTTs from certain destinations with *packet streams*. If RTTs are high, it means that the control plane was involved in forwarding the packets and that there was no installed rule that matches the probe. With this insight, attackers can infer various network policies such as host communication patterns, ACL, and monitoring rules. Liu *et al.* [80] propose a more formalized method that models switch flow tables as a Markov model, which infers fine-grained rules among complex flow

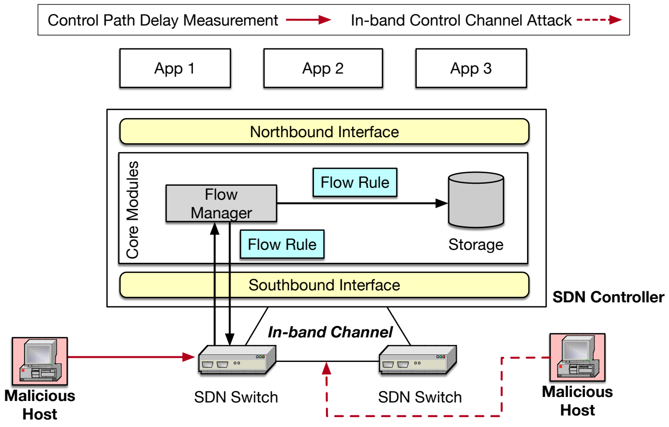


Fig. 8. Penetration routes for control-channel attacks: Control Path Delay Measurement (§IV-C1) and In-band Control Channel Attack (§IV-C2).

rules. Yu *et al.* [79] take a similar idea, but they focus on switch parameters, such as flow table size, cache replacement policy, and load. Achleitner *et al.* [47] propose flow rule reconstruction techniques with the carefully crafted probing packets that spoof a specific header field (e.g., MAC and IP addresses) to know if it is used as a match field in flow rules. If a destination host replies with the probe even though its header is spoofed, we can deduce that the field is not used. By eliminating answered headers, attackers can find a unanswered one, which is the target rule's match field.

*2) In-band Control Channel Attack:* As a network size increases, it becomes hard to construct dedicated and physically isolated control channels (i.e., *out-of-band*) from a central controller to data-plane switches due to physical distances and expensive costs. Instead, operators can choose *in-band* control-channels that make control traffic and production traffic share the same links. This design choice, however, makes attackers mount diverse malicious actions.

Basically, unencrypted in-band control channels are weak to man-in-the-middle attacks. OpenFlow specification formally recommends the SSL/TLS encryption of control channels [24] and many controller and switch vendors also support it as an option [91]. However, we observe that it is barely used in practice because of significantly degraded performance [92]. This indicates that if attackers compromise a switch or conduct ARP spoofing attacks to hijack control traffic, he can wiretap all OpenFlow messages. As demonstrated by Yoon *et al.* [51] and BEADS [67], such man-in-the-middle attackers can manipulate an action field of FLOW_MOD messages to DROP so that benign traffic is blocked.

Furthermore, since in-band control channels share the same medium with data channels, attackers can indirectly interfere with transmission of control traffic. The Crosspath attack [56] aims to flood an in-band control path by generating low-rate DDoS traffic. Although it is challenging to locate an in-band control-path, attackers can use the fingerprinting technique that measures timing differences on the control-path (§IV-C1).

Even under the SSL/TLS encryption of control channels, it cannot be fully guaranteed that there is no leakage. Cao *et al.* [57] demonstrate that attackers can analyze patterns of encrypted control traffic with deep learning (DL) and infer what kinds of SDN apps are currently running on a target SDN controller. The idea is that control traffic shows directional patterns according to SDN applications. Seo *et al.* [81] expand the scope of analysis to the context of a distributed SDN controller environment. They specifically examine the traffic exchanged between distributed SDN controllers that are widely used in SD-WAN (Software-Defined WAN) and demonstrate that attackers can gain access to confidential information such as the topology and protocols being employed in the SD-WAN through the use of deep learning-based techniques.

**Summary.** One of the major causes of vulnerabilities in the control channel of SDN is the lack of proper consideration of side channels by SDN developers, leading to information leakage and exposure of confidential information such as apps, topology, and policies. Additionally, there is a slight difference between penetration routes depending on whether a switch communicates with a controller.
**Why is the control channel vulnerable?** The separation of the control plane from the data plane in SDN introduces an additional attack surface through the network channel connecting the two planes. To mitigate this vulnerability, an out-of-band control channel [56], which uses a dedicated network line, has been proposed. However, due to feasibility constraints in the real world, in-band control channels are more commonly used in practical implementations. This constraint is exacerbated in distributed SDN controller instances, making the security of the control channel a crucial aspect to consider in SDN design.
**Is there a trend for control-channel attacks?** Initially, simple man-in-the-middle attacks were prevalent due to the reluctance of operators to use secure channels, such as SSL/TLS for their potential impact of performance. However, as the necessity for secure communication became evident, the adoption of SSL/TLS for securing control channels increased. Nevertheless, encrypted traffic analysis attacks, which aim to uncover hidden information, remain a threat. These attacks have advanced from simple timing measurements to sophisticated deep-learning techniques.

### D. Infrastructure Layer

*1) Flow Table Overloading:* Switch TCAM (Ternary Content Addressable Memory) is a critical resource that should be carefully managed. As they are normally scarce in proprietary devices, a controller should carefully install flow rules in switch flow tables. However, as there is no proper restriction for southbound interfaces, it is possible to saturate flow tables by flooding unnecessary flow rules. For example, Yoon *et al.* [51] demonstrate that a malicious app can invoke a massive number of FLOW_MOD messages with distinct match fields, causing a switch to install many different flow rules. Further, a malicious host can send randomly spoofed packets to a switch. This makes the target switch forward PACKET_INs to a controller, thereby receiving many flow rules [21], [58], [60], [63].

*2) Protocol Feature Abusing:* OpenFlow is a de-facto standard protocol that specifies controller-switch control channels, enabling any switches to communicate with a controller. However, its protocol specification delegates many detailed requirements to vendors, making security holes in switch protocol implementations.

One aspect of the holes is that of abusing protocol implementation. Yoon *et al.* [51] propose a *switch firmware abuse attack* that a malicious application deliberately replaces match fields of flow entries with the ones unsupported by hardware (e.g., MAC addresses), making packet matching to be processed by software stack. This abuses the fact that some OpenFlow switches do not support all OpenFlow match fields. It significantly degrades packet processing performance in the end. Further, a malicious app can inject a malformed OpenFlow packet with an invalid length into a switch, causing the switch disconnected from the controller [51].

In addition, an attacker can exploit OpenFlow dynamic actions to bypass security policies. Porras *et al.* [17], [83] demonstrate that a malicious app can abuse the OpenFlow `Set` action to violate network invariant, aka *dynamic tunneling attack*. OpenFlow protocols support a variety of manipulation operations for packet headers, and they facilitate diverse built-in network services within a switch forwarding pipeline without a need for middle-boxes. For example, an SDN switch can implement NAT operations using OpenFlow `Set` actions that modify packet header values to desired ones. Here, a malicious application can install a flow rule whose `Set` action is to modify blocked IP addresses to unblocked ones. While these rules conflict with an original security policy, none of the controller core services orchestrates this contention.

A vulnerability within packet forwarding logic can lead to a critical security breach. When a switch invokes a PACKET_IN message, it temporarily stores the incoming packet in a switch buffer and assigns a buffer ID, waiting for the controller's decision. The ID is used for retrieving the packet when a controller instructs the switch to forward the packet. Cao *et al.* [49] introduce a *buffered-packet hijacking attack* which exploits the fact that OpenFlow switches do not examine match fields except for a *buffer ID* when forwarding a buffered packet. So, they show that a malicious app can hijack those buffered packets if it uses the same buffer ID. If a switch receives FLOW_MOD or PACKET_OUT having the same buffer ID, the switch considers that they are a valid control message. Then, the malicious app can forward a packet illegitimately so that the packet can bypass security policies. This stems from that OpenFlow switch specification does not explicitly state that match fields should be strictly checked when buffered packets are handled [24].

On the other hand, SDNRacer [84], [85] and BigBug [86] discover race conditions on control messages between a controller and switches. Suppose that the controller needs to install bidirectional flow rules with two FLOW_MOD messages before forwarding a requested packet using PACKET_OUT. However, if BARRIER_REQUEST is not used, OpenFlow messages sent to a target switch are processed non-deterministically. For example, consider a PACKET_OUT is sent to the switch first. In that case, a pending packet is forwarded before installing a flow rule for an opposite path.

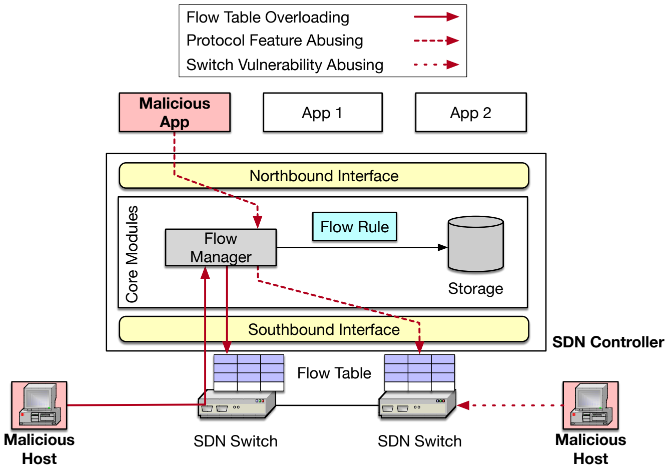


Fig. 9. Penetration routes for infrastructure-layer attacks: Flow Table Overloading (§IV-D1), Protocol Feature Abusing (§IV-D2), and Switch Vulnerability Abusing (§IV-D3).

*3) Switch Vulnerability Abusing:* SDN is often employed with NFV (Network Function Virtualization) to amplify its flexibility in modern cloud environments. For example, operators can use NFV to deploy virtualized network functions on general-purpose machines and SDN to orchestrate traffic between the NFV instances [11], [13], [14], [93]. Open vSwitch (OVS) [4] is the most popular NFV that enables to deploy high-performance virtual switches. As it is fully compatible with OpenFlow, it is widely used in cloud environments that employ SDN.

While the virtualized data-plane contributes to broadening OpenFlow deployment, it also expands attack surfaces of an attacker who looks for a chance to infiltrate inside cloud. Thimmaraju *et al.* [54] propose a remote-code execution attack that exploits a stack buffer overflow vulnerability of MPLS (Multiprotocol Label Switching) parsing logic in OVS. They discover that OVS parses all MPLS labels even if they exceed a predefined threshold. So, if an attacker injects ROP (Return Oriented Programming) gadgets into MPLS packets, they can execute a remote shell on the switch. Consequently, the attacker can compromise the virtual switch and laterally access other virtual switch instances.

Compromised switches can also be abused for bypassing middle-box service chains. Bu *et al.* [87] tackle that a compromised switch manipulates a packet tag which is used for marking service chain context of middle-boxes. As there is a lack of packet integrity check in the current SDN, this attack cannot be mitigated.

Figure 9 displays examples of penetration routes for infrastructure-layer attacks.

**Summary.** As presented, the most dominant factor that provokes infrastructure-layer attacks is the implementation flaw of switches. Switches are often targets for most penetration routes and the outcome can be either a denial of service for the switch or a violation of security policies.

**Why is the infrastructure layer vulnerable?** Due to a focus on applications and controllers as central components in SDN, the security of SDN switches is often neglected. Although vendors are required to support the OpenFlow protocol, there are inconsistencies between vendors regarding supported specifications and protocol versions. This leads to implementation bugs and unpredicted abuses of the protocol by SDN developers. As such, it is imperative for security researchers to direct greater attention towards the security of SDN switches, as these inconsistencies can lead to vulnerabilities that can be exploited by attackers.

**Is compromising SDN switches a serious attack?** Compromising SDN switches can cause more serious damage compared to legacy network environments. The transmission of corrupted information or messages from compromised devices to an SDN controller can lead to confusion and incorrect decision regarding network policies. Furthermore, attackers often target these devices as their starting point for attacks because they can be remotely accessed and may not have strong security measures in place.

## V. SDN Defense Classification

In this section, we classify countermeasures to defend the aforementioned SDN attacks. Table IV summarizes all surveyed countermeasures and their covered attack root causes, components, and defense types according to the taxonomy presented in §III. Again, we discuss our insights and findings in the end.

### *A. Application Layer*

*1) Application Authentication and Authorization Models:* The root cause of enabling a malicious app to do harmful API invocation is that most controllers have no authentication and authorization measures for SDN apps. For this reason, security researchers have proposed several feasible access control methods in SDN context.

First, a digital certificate signed by trusted entities would help operators trust SDN apps. FortNOX [83] and SE-Floodlight [17] use the digital signature when tracking flow rules driven from specific apps. Rosemary [18] employs public key infrastructure (PKI) to verify if SDN apps are correctly signed by a developer. Those cases show that application authentication can fundamentally prevent malicious apps from being installed. We believe that a set of digital certification methods established between SDN developers and operators will incredibly enhance a trusted SDN app ecosystem.

Second, a role-based access control (RBAC) has been introduced by FortNOX [17], SE-Floodlight [83], SM-ONOS [15], and Barista [62]. The main purpose is to restrict application behavior through granting predefined priorities to running SDN apps. For example, SDN apps can be assigned by a *role*, which represents a security-level to access certain APIs. Therefore, the lowest priority app cannot invoke a security-sensitive northbound API, such as `flowruleWrite()` that is used for modifying flow rules.

Third, an API-level permission model would be a proper security measure to be employed in modern SDN controllers. Since most controllers have become complex, and more apps have been developed, the RBAC model would be too coarse-grained to handle all corner cases; e.g., operators may want to block a `flowruleWrite()` API while allowing others. For this, security-mode ONOS (SM-ONOS) [15] proposes an API-level permission model tailored for the ONOS SDN controller. With the combination of the RBAC model, it introduced a hierarchical model that consists of application, bundle, and API-level permissions. SDNShield [16] presents more advanced models that use fine-grained permission filters similar to Berkeley Packet Filter (BPF). Based on a *manifest* file that includes used permissions in SDN apps, operators can choose desired ones with the filters. This requires developers to correctly write a *manifest* file that describes used permissions for the apps, yet attackers can specify a false permission to deploy malicious apps. AEGIS [52] proposes an natural-language-processing (NLP) based analysis system to compare used permissions of an SDN app with its manifest file. SE-Floodlight [17] proposes a permission model for southbound interfaces. It dictates which roles should be assigned to an app so as to use an OpenFlow message.

*2) Dynamic Instrumentation and Provenance Graph Analysis:* Application-level race conditions are mainly related to unexpected implementation bugs typically triggered in a dynamic environment, so it is hard to find them from a simple unit testing during a development phase. Thus, finding hidden bugs requires a more advanced approach that can consider complicated interactions occurred between apps and core services, but it requires manual auditing, which is time-consuming and error-prone.

One line of research is to troubleshoot possible bug points by analyzing controller traces (e.g., logs) with the help of *dynamic instrumentation*. It aims to pinpoint event sequences that may trigger bugs so as to facilitate operator's debugging process. OFRewind [94] is a traffic-replay tool that dynamically records control and data traffic, and reproduces them to find bugs when controller operations fail. STS [71] leverages the delta debugging concept that localizes minimum code snippets which are likely to raise exceptional cases. SDNRacer [84], [85], BigBug [86], and ConGuard [45] investigate happens-before causality relations from recorded event sequences to detect harmful race conditions between multi-threaded applications.

A *provenance graph* is useful for knowing causal relations of a complex attack chain. Here, dynamic instrumentation is also used to hook controller APIs and build the provenance graph. ForenGuard [12] proposes a provenance-based root cause analysis framework which dynamically records how an event is propagated from the data plane to the control plane. ProvSDN [48] aims to locate a root cause of cross-app poisoning attacks by backtracking poisoned data that guides a victim app to make a harmful decision. GitFlow [96] takes inspiration from version control systems, such as Git, to create

TABLE IV
SYSTEMATIZATION OF SDN DEFENSES.
● ROOT CAUSE/TARGET/DEFENSE

| Layer | Sec. | Defense | Root Cause | | | | | | | | | Target Component/Interface | | | | | | | Defense Type | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Lack of NBI Authorization | Lack of SBI Authorization | Lack of Control Event Integrity | Lack of Control Message Integrity | Lack of Application Authentication | Lack of Switch/Host Authentication | Lack of Controller Resource Control | Side Channel | Implementation Flaw | Application | Northbound Interface | Controller | Southbound Interface | Switch | Switch Interface | Host | Control-Plane Extension | Data-Plane Extension | Pen-Testing | Program Analysis | API Monitoring | Event/Message Monitoring |
| Application | §V-A1 | Application Authentication<br>[17], [18], [83] | | | | | ● | | | | | ● | | | | | | | ● | | | | | |
| | | Role-based Authorization<br>[15], [17], [62], [83] | ● | | | | ● | | | | | ● | ● | | | | | | ● | | | | | |
| | | Northbound API Permission Model<br>[15], [16], [52] | ● | | | | | | | | | | ● | | | | | | ● | | | | ● | |
| | | Southbound API Permission Model<br>[17] | | ● | | | | | | | | | | | ● | | | | ● | | | | ● | |
| | §V-A2 | Dynamic Instrumentation<br>[45], [71], [84]–[86], [94] | | | | | | | | | ● | ● | | ● | | | | | | | | ● | | |
| | | Provenance Graph Analysis<br>[12], [48], [95], [96] | | | | | | | | | ● | ● | | | | | | | | | | ● | ● | ● |
| | §V-A3 | Control/Data Flow Analysis<br>[12], [48], [69], [70] | | | | | | | | | ● | ● | | | | | | | | | | ● | | |
| | §V-A4 | Control-Plane Invariant Verification<br>[17], [97]–[99] | | | | | | | | | ● | ● | | | | | | | | | | ● | | |
| Control | §V-B1 | Proactive Rule Installation<br>[63], [64] | | | | | | | ● | | | | | ● | ● | | | | | ● | | | | |
| | | Switch Module Extension<br>[22], [23], [63], [64], [75] | | | | | | | ● | | | | | ● | ● | | | | | ● | | | | |
| | | OpenFlow Protocol Extension<br>[23], [63] | | | | | | | ● | | | | | ● | | | | | | ● | | | | |
| | §V-B2 | Multi Controller<br>[37], [100], [101] | | | | | | | ● | | | | | ● | | | | | ● | | | | | |
| | | Controller Failure Recovery<br>[102], [103] | | | | | | | ● | | | | | ● | | | | | ● | | | | | |
| | | Controller Sand-boxing<br>[18], [62], [103] | | | | | | | ● | | | | | ● | | | | | ● | | | | | |
| | §V-B3 | Topology Event Verification<br>[43], [60], [75] | | | ● | ● | | | | | | | | ● | | | | | | | | | | ● |
| | §V-B4 | Switch/Host Authentication<br>[43], [76], [104] | | | | | | ● | | | | | | | ● | ● | | ● | | ● | | | | ● |
| | §V-B5 | Control Event Blackbox Fuzzing<br>[66] | | | ● | | | | | | ● | | ● | | | | | | | | ● | | | |
| | | Control Message Blackbox Fuzzing<br>[65]–[67] | | | | ● | | | | | ● | | | | ● | | | | | | ● | | | |
| Control Channel | §V-C1 | SSL/TLS Encryption<br>[24] | | | | | | ● | | | | | | | ● | | | | ● | ● | | | | |
| | §V-C2 | Delay Normalization<br>[46] | | | | | | | | ● | | | | | ● | | | | ● | ● | | | | |
| | | Delay Randomization<br>[55], [56], [80] | | | | | | | | ● | | | | | ● | | | | ● | ● | | | | |
| Infrastructure | §V-D1 | Data-Plane Invariant Verification<br>[60], [72], [73] | | | | | | | | | ● | | | | ● | | | | | | | | | ● |
| | §V-D2 | Protocol Conformance Testing<br>[66], [68] | | | | | | | | | ● | | | | | ● | | | | | ● | | | |
| | §V-D3 | Malicious Switch Detection<br>[105]–[107] | | | | | | ● | | | | | | | | ● | | | | | | | | ● |

a versioned provenance graph that tracks the evolution of flow states. In contrast, PicoSDN [95] offers a more comprehensive provenance graph to tackle the limitations of previous approaches. These limitations include difficulties with managing dependencies and a lack of complete provenance information that hinders the ability to detect cross-plane attacks.

*3) Control and Data Flow Analysis:* Typical execution flows in an application layer can be represented as a series of API call sequences triggered by an event (§IV-A2). The hidden attack chains normally stem from these intractable processing sequences. From the complex call sequence, it is hard to pinpoint a suspect API that contributes to violating a security policy.

To address this, operators can employ the *static analysis* that examines the control flow graph (CFG) extracted from an application source code. INDAGO [69] leverages a machine learning (ML) approach to find suspicious patterns of API call chains from malicious apps. It investigates diverse features pertaining to security-sensitive APIs that manipulate states of SDN controllers. By conducting clustering analysis, it is shown that malicious API chains can be detected with high accuracy. EventScope [70] extends the CFG into an event flow graph to catch how events are propagated over code blocks within an application. Its purpose is to detect "unhandled" data-plane events by the application, which makes a hole that an attacker can bypass security logic. Besides, the provenance-based defenses also utilize static analysis for pre-processing API call chains before instrumentation [12], [48].

*4) Control-Plane Invariant Verification:* Formal methods would be useful when checking correctness of network policies enforced by SDN apps prior to deploying rules into switches. The key benefit behind this approach is that it could guarantee that apps will not violate network invariant. SE-Floodlight [17] proposes the rule-based conflict analysis (RCA) algorithm that investigates conflicts between a newly created OpenFlow rule and existing ones. For example, a malicious app can abuse the OpenFlow `Set` action that modifies packet headers to create a rule chain, bypassing a security policy. The goal of RCA is to simulate the possible chain and compares it with existing rules to detect conflicts. FLOVER [99] and VeriCon [97] model SDN apps as first-order logic to check invariant with Satisfiability Modulo Theories (SMT) solvers. While this may take long time if that there are many invariants required to investigate, it can correctly verify possible corner cases. NICE [98] uses model checking to examine if invariants hold under a certain controller state. However, exploring all possible states is intractable given the number of possible state transitions determined by diverse inputs (e.g., packets, events); thus, they also use symbolic execution to reduce input space.

**Summary.** The focus of defense strategies primarily centers around controller extension and its program analysis, as the controller is a crucial component for application support.

**Is there a trend for application-layer defenses?** There have been various proposed attack scenarios in the application layer, leading to active research in this area to prevent them. From our analysis, we notice that most studies have borrowed ideas from existing malware analysis techniques. For example, dynamic instrumentation techniques for SDN applications are based on existing binary instrumentation techniques, commonly used in malware analysis [108] and control flow analysis approaches also adopt their main ideas from popular malware static analysis platforms [109].

**Are proposed defenses deployed in practice?** We observe that simple security measures, such as permission models, have already been implemented in real-world SDN controllers (e.g., SM-ONOS [110]). However, it is difficult to find real-world cases of using SDN malware detection methods in controllers, indicating a need for more feasible and practical solutions for the SDN environment.

### *B. Control Layer*

*1) Migrating Control-Plane Function to Data-Plane:* Although the separation of control and data planes is the key to operating SDN, the needs for partial migration of control-plane functions into the data plane were raised. As early SDN researchers pursued the design philosophy of 4D [1], SDN switches become simple forwarding devices whose local-decision capability is limited. The main problem is that SDN switches are too verbose; switches always need to ask a controller when unknown packets are detected (i.e., table-miss). Several security measures have been introduced to reduce the controller burden by letting switches make some decisions by themselves when necessary.

Proactive installation of "wildcard" rules is one option to reduce the controller burden. DIFANE [64] introduces a concept of *authority switches* that are in charge of determining forwarding actions of network partitions (like a default gateway). Those switches have wildcard rules that match with partial flow space of policies. Ingress switches can ask the authority switches when table-miss occurs without querying an instruction to a controller. DevoFlow [63] also proposes the aggressive use of wildcard rules for uninteresting rules such as micro-flows, devolving most decisions to the data-plane. These approaches have a benefit in that they are required to minimally modify existing OpenFlow switches.

Switch modules can be extended to support more intelligence to be robust against the saturation attacks. AVANT-GUARD [23] proposes the *connection migration* technique that relays traffic only if TCP sessions are established correctly. As switches defer a report of a new TCP connection request to control-plane when it is confirmed as a reliable connection through syn cookie, the excessive control-plane dependency is reduced. FloodGuard [22] builds a switch add-on module, called *data-cache* that temporarily stores table-miss packets to avoid control-plane saturated when flooding is detected. It utilizes multi-queues that buffer packets of different protocols from the rationale that an attacker tends to use a single protocol when performing flooding attacks (e.g., TCP, ICMP flooding). SWGuard [75] employs the similar approach that maintains queues inside of a switch according to OpenFlow message types. All those queue-based extensions are used for scheduling packets to alleviate data-to-control-plane messages.

Several work attempted to extend protocol capabilities to incorporate more functions than native OpenFlow actions. DevoFlow [63] proposes *rule cloning* and *local routing* actions. The former action reduces excessive usage of switch TCAM by making an exact-matching rule from a given wildcard rule, and the latter enables switches to determine multi-paths similar to ECMP or automatically reroute a path when a failed output port is detected. AVANT-GUARD [23] introduced *actuating triggers* which extends the switch function to report network status asynchronously without a control-plane operation.

*2) Building Scalable and Fault-Tolerant Control-Plane:* To address the control plane saturation attacks and the single point of a failure problem, various traditional concepts, such as distributed systems, OSes, and database systems have been applied in SDN controllers.

Physical extension of a single SDN controller has been adopted to obtain resilience against DDoS attacks, and even high-performance by partitioning a network into several segments. Onix [37] is the first trial to design a multi-controller platform running on a large-scale production network. It maintains NIB (Network Information Base), which is a logically centralized graph abstraction for data-plane elements (e.g., forwarding tables, topology). Multiple controller instances divide NIB into several chunks, and aggregate the part of it to a single node when necessary so as to avoid excessive memory usage per controller instance.

ONOS [100] is a widely-used open-source multi-controller that features a master-slave relationship between switches and controllers. The *master* controller can perform both read and write operations on a switch, while the *slave* can only read the switch's state. OpenDaylight [101] is a model-driven multi-controller that divides all data into units called shards, which are minimum data units like topology and flows. These shards are communicated between controller instances for synchronization. Both ONOS and OpenDaylight use Raft, a consensus algorithm for strong consistency, and distinguish between a *leader* and *followers*, where the leader accepts read/write operations and replicates them to followers.

As controller instances augmented, state replication is employed so that slave controller instances can maintain a consistent view with the master through the east/westbound interface (§II-A). Ravana [102] models the process of state replication as a database transaction, and proposed a *two-phase replication* protocol that guarantees atomicity of a replication process. In addition, failure recovery protocols (or algorithms) would help in restoring states of control and data-planes in case of failure. LegoSDN [103] introduces a *cross-layer rollback* mechanism through managing snapshots. When an app fails, it enables the controller to restore previous states from a saved checkpoint.

To address the monolithic architecture of SDN controllers, researchers have attempted to compartment execution space of applications and core modules. Rosemary [18] and LegoSDN [103] propose a *sandbox* architecture that separates applications from core modules. By doing so, application processes are detached from the space of core modules, and API invocation is delivered using RPC. Further, Rosemary [18] and Barista [62] present a *micro-kernel* architecture that divides core modules into separated ones so that they are isolated from each other.

*3) Topology Event Verification:* The root cause behind the topology poisoning attack is that existing SDN controllers do not check validity of topology events when updating storage even if those events are inconsistent with data-plane context. TopoGuard [43] proposes a validation subsystem to filter unacceptable topology events considering current port states. For example, it uses PORT_UP and PORT_DOWN events as host-specific events that indicate if a host-connected link is up or down, respectively. If a switch receives an LLDP packet from the port where a host was connected with the PORT_UP event before (i.e., *link fabrication*), it regards this link-event invalid. To defend the *host identifier spoofing* attack, TopoGuard sends a probe packet to an original host location when a PORT_DOWN event is detected. This prevents a victim identifier from being hijacked by an attacker during host migration. TopoGuard+ [75] complements the weaknesses of TopoGuard by adding a link latency measurement module. It focuses on the fact that a fake link will show abnormally high latency since malicious hosts relay LLDP packets in the middle. SPHINX [60] proposes a general framework that intercepts all OpenFlow messages and builds a flow graph which reflects a current topology view. It captures the anomaly such as fake link injection or identifier spoofing by verifying host-switch-port binding on the graph.

*4) Data-Plane Entity Authentication:* Current SDN controllers lack authentication measure for data-plane entities (e.g., switches, hosts), which is in contrast to the design philosophy of Ethane [5]. TopoGuard [43] proposes a link-event authentication that embeds a switch signature into an LLDP packet. By checking the signature, it can ensure that the LLDP packet is sent by a trustworthy switch. SecureBinder [76] extends IEEE 802.1x protocol that validates host MAC addresses with certificates signed by CA. This effectively prevents attackers from spoofing other identifiers. DFI [104] proposes a more fine-grained access control system that authenticates an end-host with high-level identifiers such as user- and host-names. With the help of centralized view in SDN, it enforces stateful ACL rules according to the status of a target host.

*5) Control Event and Message Blackbox Fuzzing:* Fuzzing techniques have been proven to be useful for pinpointing hidden bugs pertaining to implementation flaws of handling a control channel protocol, OpenFlow. As these bugs can be abused by hidden attack chains, it is helpful to find and fix them before deployment. Protocol implementation normally involves large input space due to a variety of protocol messages, thus it is hard to find exceptional cases with manual labor. Fuzz-testing can address this challenge by exploring all possible input combinations to find unexpected behavior. DELTA [66], BEADS [67] and ATTAIN [65] are representative fuzzing tools. Their fuzzing techniques aim to either conduct anomaly actions (e.g., packet drop, manipulation, and delay) in the middle of protocol sessions or inject malformed messages that do not correspond to protocol specification. These kind of black box fuzzing techniques, which can be utilized as security assessment tools at the same time, can classify inherent weakness of the target SDN controllers while incredibly reducing manual effort; thus, it would be efficient for operators to measure their own controller's security.

Furthermore, DELTA incorporates a control event fuzzing module that discovers potential vulnerabilities of northbound interfaces. They are mainly raised from mis-implementation of event processing logic in controller internals. By randomizing inputs or control flow sequences in an application service chain, it was able to find possible bugs pertaining to control events, which can be abused by a malicious app.

**Summary.** Most defense mechanisms focus on addressing the shortage of resource control in current SDN controllers. Many works have proposed extending the control-plane or data-plane to enhance security.

**Why is the control-layer security important?** An SDN controller is often referred to as the “brain” of the network [45]. As the name implies, a failure in the controller can result in a malfunction of the entire network. Therefore, SDN controllers must have fundamental security properties to ensure fault-tolerance.

**Is there a trend for control-layer defenses?** Some proposals such as application isolation and resource management were proposed in academia. However, in industry, instead of adopting these ideas, popular controllers like ONOS and OpenDaylight have implemented a multi-controller architecture, which has proven to be robust in carrier-grade networking environments [111].

**Are multi-controllers secure enough?** The use of multi-controller or distributed controllers in SDN has not completely resolved the issue of a malicious app affecting control-layer operations. This is because the current design of distributed controllers adopts a physically distributed but logically centralized architecture, wherein decisions made by a single controller are replicated across all controllers. This architecture creates a risk of the entire system being compromised if a malicious controller is able to manipulate decisions.

### *C. Control Channel*

*1) SSL/TLS Encryption:* The recent OpenFlow specification (v 1.5.0) [24] recommends the use of SSL/TLS for encrypting the control channel in SDN networks. However, proper configuration of a public/private key pair for each site results in increased cost and also decreased performance compared to TCP [92]. Thus, the use of plain TCP is still prevalent, despite not being recommended by the OpenFlow specification.

*2) Timing Obfuscation:* The root cause of timing-based side-channel attacks in SDN is the need for switches to ask the controller for instructions when encountering an unknown packet. This creates a timing difference that can be exploited. To defend against these attacks, obfuscating timing delays is one potential solution. For instance, Sonchack *et al.* [46] propose a timeout proxy on a switch to normalize control path latency. If a packet does not match a flow table, it follows a default forwarding rule installed on the switch. Other researchers have suggested similar approaches, such as adding random delays to control channels [55], [56], [80], [81].

**Summary:** Despite the potential for the exposure of diverse confidential information through the control channel, few practical defense solutions have been proposed to mitigate the risk of fingerprinting attacks. Except for [46], researchers have only discussed general suggestions or guidelines without presenting concrete designs for systems to counteract such attacks.

**Why is control-channel defense important?** The control channel of SDN plays a crucial role in facilitating communication between the controller and switches. However, the security of this channel is compromised by inherent design weaknesses, which make it challenging to fully protect against side-channel attacks. The deployment of obfuscation-based solutions to address these security issues is also problematic due to their adverse effect on performance.

**Is there a feasible defense for control-channel?** A feasible solution to defend against control-channel security issues is to avoid table-mismatch by using proactive rules with wildcard match fields. This approach reduces control-plane interactions and can hide timing differences. This has been proposed in literature [63], [64].

### *D. Infrastructure Layer*

*1) Data-Plane Invariant Verification:* Flow rule verification aims to check if data-plane states correspond to network policies. It can thwart the rule manipulation attack or violated rules from an application bug. Many of prior studies use *control-message hooking* techniques that capture control messages to check if they correspond to intended network policies. VeriFlow [72] designs a real-time invariant verification system that sits between the control and infrastructure layer to intercept all OpenFlow messages. By modeling traffic classes as *equivalence classes*, it enables fast analysis through looking into the required parts of an address space, and pinpoints the violated one. The flow graph proposed by SPHINX [60] can be used to verify network invariants (§V-B3). For example, operators can specify the “waypoint” invariant that enforces a flow to pass through a certain point with a policy language, and then SPHINX verifies the flow on the flow graph. Ropke *et al.* [73] propose a system that compares control events generated from apps and control messages applied to the data-plane. This prevents a malicious app (e.g., rootkit) from installing a false flow rule that violates operator’s network policies.

*2) Switch Protocol Implementation Testing:* As the difference in the OpenFlow implementations of switches leads to unexpected bugs, it can expose critical vulnerabilities to attackers who aim to compromise switches or abuse protocol specification. The root cause of this security hole is that different switch vendors interpret specification in a different manner. With this fact, there have been several studies to find those implementation holes using protocol implementation testing. SOFT [68] attempts to find implementation inconsistencies among different switch vendors. It utilizes symbolic execution to explore control flows of switch agents, and compares their different outcomes. DELTA [66] uses its value-fuzzer module to inject randomized OpenFlow packets into switches, for the purpose of finding abnormal cases in switches.

*3) Malicious Switch Detection:* As discussed, SDN switches can be vulnerable to compromise, which could allow attackers to intercept and drop packets. To prevent this from happening, various techniques for detecting malicious switches that exhibit anomalous behavior have been proposed. Kamisinski *et al.* [105] suggest that there are two types of malicious switches: packet droppers and packet swappers, which forward packets to different ports. They detect these threats by using anomaly detection. Chi *et al.* [106] propose an online detection algorithm that creates an artificial packet from a controller to see if it follows the intended forwarding path. Mohan *et al.* [107] use node-disjoint control paths based on the fact that two control messages will be inconsistent if the source switch is compromised. A more comprehensive survey related this topic is presented in [112].

**Summary.** Various studies have been conducted on the topic of infrastructure-layer security. However, a majority of these proposals concentrate on the verification of enforced rules or the evaluation of protocol correctness. There remains a significant gap in the literature with respect to the security of network devices from malicious attacks.

**Why is the infrastructure-layer defense important?** With the increasing reliance on network infrastructure, it is crucial to ensure its security. Attackers often target network devices as a starting point for their attacks, thereby making it imperative to secure these components. The use of software switches in modern cloud environments adds an additional layer of complexity, as it increases the number of potential attack surfaces. In light of this, it is imperative that strong authentication and authorization mechanisms are in place for network switches to enhance the overall security of the infrastructure layer.

## VI. Future Research Directions

Before concluding the remark, we highlight two future research directions: (i) distributed SDN controllers and (ii) programmable data plane.

### A. Vulnerabilities in Distributed SDN Controllers

As modern networking environments significantly grow, a single SDN controller will not be enough to orchestrate enormous underlying traffic. Distributed SDN architectures enable to reduce the burdensome overload of a single centralized controller and guarantee its resiliency by virtue of a fault-tolerant system. However, while vulnerabilities within a single controller environment have been widely studied, the environments of distributed controllers have *not* been thoroughly investigated yet. Therefore, we need to delve into security issues that may be arisen from the distributed SDN architecture.

In addition, modern distributed SDN architectures have constantly evolved. For example, ONOS [40] has recently separated its underlying distributed storage into an independent project, called Atomix [113]. Now, operators can flexibly design a *cluster* that consists of controllers and storage with their preferred configurations. While this can facilitate the development of various use cases that need distributed systems, we expect that the newly emerged structure will expose new attack surfaces to attackers.

### B. Vulnerabilities in Programmable Data Plane

Recently, the programmable data plane is considered to be the next killer paradigm for innovative networking research beyond SDN. For instance, P4 [114] is advocated by both academia and industry due to its ability to customize packet processing logic while enjoying the benefits of line-rate performance. Similar to SDN, its primary advantage originates from the programmablility, which implies that vulnerabilities can exist in software-defined logic or implementations. However, to our knowledge, security within the programmable data plane has yet to receive significant attention. Thus, it may give attackers another chance to mount harmful attacks again.

## VII. Conclusion

In this paper, we have surveyed disclosed attacks and defenses in each layer of the SDN architecture and systematically investigated their motivations, approaches, and fundamental security issues across the layers.

With our careful analysis, we have observed that most SDN attacks have been discovered while introducing new functionalities or interfaces. The tenet of SDN is to support *programmability* with a rich set of APIs, but these APIs simultaneously increase the attack surfaces that attackers can abuse. We have also found that most SDN controllers have been developed without properly incorporating security features. The control channel in SDN is often regarded as a weak point that can expose confidential information (e.g., global view of a network and identity of SDN resources). However, we have only observed a small number of works that focus on the control channel. The security of the infrastructure layer has been paid less attention than those of the application and control layers.

While many researchers have analyzed potential vulnerabilities and developed feasible defenses in SDN, we conclude that the security of SDN controllers should be considered more seriously. Considering the importance of the secure communication between the control and data planes, we also conclude that its security issues should be more investigated. As the ”softwarization” trend is moving to the data plane (e.g., P4), we conclude that security researchers need to start exploring security issues in this layer. We hope this paper helps revisit existing SDN security works and sheds light on future research directions.

## References

[1] 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,” *ACM SIGCOMM Computer Communication Review*, 2005.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling Innovation in Campus Networks,” *ACM SIGCOMM Computer Communication Review*, 2008.

[3] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, “FlowVisor: A Network Virtualization Layer,” *OpenFlow Switch Consortium, Tech. Rep*, 2009.

[4] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, “The Design and Implementation of Open vSwitch,” in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*. USENIX, 2015.

[5] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, “Ethane: Taking Control of the Enterprise,” *ACM SIGCOMM computer communication review*, 2007.

[6] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, “B4: Experience With a Globally-Deployed Software Defined WAN,” *ACM SIGCOMM Computer Communication Review*, 2013.

[7] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving Energy in Data Center Networks," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*. USENIX, 2010.

[8] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving High Utilization with Software-Driven WAN," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2013.

[9] S. Shin, L. Xu, S. Hong, and G. Gu, "Enhancing Network Security through Software Defined Networking (SDN)," in *Proceedings of the International Conference on Computer Communication and Networks*. IEEE, 2016.

[10] S. W. Shin, P. Porras, V. Yegneswara, M. Fong, G. Gu, M. Tyson *et al.*, "FRESCO: Modular Composable Security Services for Software-Defined Networks," in *Proceedings of the Network & Distributed System Security Symposium*. Internet Society, 2013.

[11] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and Elastic DDoS Defense," in *Proceedings of the USENIX Security Symposium*. USENIX, 2015.

[12] H. Wang, G. Yang, P. Chinprutthiwong, L. Xu, Y. Zhang, and G. Gu, "Towards Fine-Grained Network Security Forensics and Diagnosis in the SDN Era," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018.

[13] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying Middlebox Policy Enforcement using SDN," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2013.

[14] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*. USENIX, 2014.

[15] C. Yoon, S. Shin, P. Porras, V. Yegneswaran, H. Kang, M. Fong, B. O'Connor, and T. Vachuska, "A Security-Mode for Carrier-Grade SDN Controllers," in *Proceedings of the Annual Computer Security Applications Conference*, 2017.

[16] X. Wen, B. Yang, Y. Chen, C. Hu, Y. Wang, B. Liu, and X. Chen, "SDNShield: Reconciliating Configurable Application Permissions for SDN App Markets," in *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2016.

[17] P. A. Porras, S. Cheung, M. W. Fong, K. Skinner, and V. Yegneswaran, "Securing the Software Defined Network Control Layer," in *Proceedings of the Network and Distributed System Security Symposium*. Internet Society, 2015.

[18] 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 ACM SIGSAC conference on computer and communications security*, 2014.

[19] S. Lee, C. Yoon, and S. Shin, "The Smaller, the Shrewder: A Simple Malicious Application Can Kill an Entire SDN Environment," in *Proceedings of the ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*. ACM, 2016.

[20] C. Röpke and T. Holz, "SDN Rootkits: Subverting Network Operating Systems of Software-Defined Networks," in *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*. Springer, 2015.

[21] S. Shin and G. Gu, "Attacking Software-Defined Networks: A First Feasibility Study," in *Proceedings of the ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013.

[22] H. Wang, L. Xu, and G. Gu, "FloodGuard: A DoS Attack Prevention Extension in Software-Defined Networks," in *Proceedings of the Conference on Dependable Systems and Networks*. IEEE, 2015.

[23] 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 ACM SIGSAC Conference on Computer & Communications Security*, 2013.

[24] "OpenFlow Switch Specification v1.5.1," https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf, 2022.

[25] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards Secure and Dependable Software-Defined Networks," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, 2013.

[26] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys & Tutorials*, 2015.

[27] I. Alsmadi and D. Xu, "Security of software defined networks: A survey," *Computers & security*, vol. 53, pp. 79–108, 2015.

[28] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE communications surveys & tutorials*, vol. 18, no. 1, pp. 602–622, 2015.

[29] S. Khan, A. Gani, A. W. A. Wahab, M. Guizani, and M. K. Khan, "Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 303–324, 2016.

[30] A. Shaghaghi, M. A. Kaafar, R. Buyya, and S. Jha, "Software-defined network (sdn) data plane security: Issues, solutions, and future directions," *Handbook of Computer Networks and Cyber Security: Principles and Paradigms*, pp. 341–387, 2020.

[31] J. C. C. Chica, J. C. Imbachi, and J. F. B. Vega, "Security in sdn: A comprehensive survey," *Journal of Network and Computer Applications*, vol. 159, p. 102595, 2020.

[32] B. Rauf, H. Abbas, M. Usman, T. A. Zia, W. Iqbal, Y. Abbas, and H. Afzal, "Application threats to exploit northbound interface vulnerabilities in software defined networks," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–36, 2021.

[33] I. Farris, T. Taleb, Y. Khettab, and J. Song, "A survey on emerging sdn and nfv security mechanisms for iot systems," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 812–837, 2018.

[34] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: transparent moving target defense using software defined networking," in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, pp. 127–132.

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

[36] Z. Cai, A. L. Cox, and T. Ng, "Maestro: A System for Scalable OpenFlow Control," Tech. Rep., 2010.

[37] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A Distributed Control Platform for Large-Scale Production Networks," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*. USENIX, 2010.

[38] "Floodlight Controller," https://github.com/floodlight/floodlight, 2023.

[39] D. Erickson, "The Beacon OpenFlow Controller," in *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, 2013.

[40] "ONOS Github Repository," https://github.com/opennetworkinglab/onos, 2022.

[41] "OpenDaylight Github Repository," https://github.com/opendaylight/, 2023.

[42] S. Lee, S. Woo, J. Kim, V. Yegneswaran, P. Porras, and S. Shin, "AudiSDN: Automated Detection of Network Policy Inconsistencies in Software-Defined Networks," in *Proceedings of the IEEE Conference on Computer Communications*. IEEE, 2020.

[43] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures," in *Proceedings of the Network and Distributed System Security Symposium*. Internet Society, 2015.

[44] B. Agborubere and E. Sanchez-Velazquez, "Openflow Communications and TLS Security in Software-Defined Networks," in *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2017.

[45] L. Xu, J. Huang, S. Hong, J. Zhang, and G. Gu, "Attacking the Brain: Races in the SDN Control Plane," in *Proceedings of the USENIX Security Symposium*. USENIX, 2017.

[46] 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 Annual Conference on Computer Security Applications*, 2016.

[47] S. Achleitner, T. La Porta, T. Jaeger, and P. McDaniel, "Adversarial Network Forensics in Software Defined Networking," in *Proceedings of the Symposium on SDN Research*. ACM, 2017.

[48] B. E. Ujcich, S. Jero, A. Edmundson, Q. Wang, R. Skowyra, J. Landry, A. Bates, W. H. Sanders, C. Nita-Rotaru, and H. Okhravi, "Cross-app Poisoning in Software-Defined Networking," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018.

[49] J. Cao, R. Xie, K. Sun, Q. Li, G. Gu, and M. Xu, "When Match Fields do not Need to Match: Buffered Packet Hijacking in SDN," in *Pro-*

*ceedings of the Network and Distributed System Security Symposium*. Internet Society, 2020.

[50] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 2012.

[51] C. Yoon, S. Lee, H. Kang, T. Park, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Flow Wars: Systemizing the Attack Surface and Defenses in Software-Defined Networks," *IEEE/ACM Transactions on Networking*, 2017.

[52] H. Kang, S. Shin, V. Yegneswaran, S. Ghosh, and P. Porras, "AEGIS: An Automated Permission Generation and Verification System for SDNs," in *Proceedings of the Workshop on Security in Softwarized Networks: Prospects and Challenges*, 2018.

[53] F. F. Lindner, "Router Exploitation," in *Black Hat Briefings USA*, 2009.

[54] K. Thimmaraju, B. Shastry, T. Fiebig, F. Hetzelt, J.-P. Seifert, A. Feldmann, and S. Schmid, "Taking Control of SDN-based Cloud Systems via the Data Plane," in *Proceedings of the Symposium on SDN Research*, 2018.

[55] R. Bifulco, H. Cui, G. O. Karame, and F. Klaedtke, "Fingerprinting Software-Defined Networks," in *Proceedings of the International Conference on Network Protocols*. IEEE, 2015.

[56] J. Cao, Q. Li, R. Xie, K. Sun, G. Gu, M. Xu, and Y. Yang, "The CrossPath Attack: Disrupting the SDN Control Channel via Shared Links," in *Proceedgins of the Security Symposium*. USENIX, 2019.

[57] J. Cao, Z. Yang, K. Sun, Q. Li, M. Xu, and P. Han, "Fingerprinting SDN Applications via Encrypted Control Traffic," in *Proceedings of the International Symposium on Research in Attacks, Intrusions and Defenses*, 2019.

[58] M. Zhang, G. Li, L. Xu, J. Bi, G. Gu, and J. Bai, "Control Plane Reflection Attacks in SDNs: New Attacks and Countermeasures," in *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018.

[59] G. Shang, P. Zhe, X. Bin, H. Aiqun, and R. Kui, "FloodDefender: Protecting Data and Control Plane Resources under SDN-Aimed DoS Attacks," in *Proceedings of the IEEE Conference on Computer Communications*. IEEE, 2017.

[60] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "SPHINX: Detecting Security Attacks in Software-Defined Networks," in *Proceedings of the Network and Distributed System Security Symposium*. Internet Society, 2015.

[61] V. H. Dixit, A. Doupé, Y. Shoshitaishvili, Z. Zhao, and G.-J. Ahn, "AIM-SDN: Attacking Information Mismanagement in SDN-datastores," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018.

[62] J. Nam, H. Jo, Y. Kim, P. Porras, V. Yegneswaran, and S. Shin, "Barista: An Event-centric NOS Composition Framework for Software-Defined Networks," in *Proceedings of the IEEE Conference on Computer Communications*. IEEE, 2018.

[63] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling Flow Management for High-Performance Networks," in *Proceedings of Conference of the ACM Special Interest Group on Data Communication*, 2011.

[64] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable Flow-based Networking with DIFANE," in *Proceedings of the ACM Special Interest Group on Data Communication*. ACM, 2010.

[65] B. E. Ujcich, U. Thakore, and W. H. Sanders, "Attain: An Attack Injection Framework for Software-Defined Networking," in *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2017.

[66] S. Lee, C. Yoon, C. Lee, S. Shin, V. Yegneswaran, and P. A. Porras, "DELTA: A Security Assessment Framework for Software-Defined Networks," in *Proceedings of the Network and Distributed System Security Symposium*. Internet Society, 2017.

[67] S. Jero, X. Bu, C. Nita-Rotaru, H. Okhravi, R. Skowyra, and S. Fahmy, "BEADS: Automated Attack Discovery in OpenFlow-based SDN Systems," in *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2017.

[68] M. Kuzniar, P. Peresini, M. Canini, D. Venzano, and D. Kostic, "A SOFT Way for Openflow Switch Interoperability Testing," in *Proceedings of the International Conference on Emerging Networking Experiments and Technologies*. ACM, 2012.

[69] C. Lee, C. Yoon, S. Shin, and S. K. Cha, "INDAGO: A New Framework for Detecting Malicious SDN Applications," in *Proceedings of the IEEE International Conference on Network Protocols*. IEEE, 2018.

[70] B. E. Ujcich, S. Jero, R. Skowyra, S. R. Gomez, A. Bates, W. H. Sanders, and H. Okhravi, "Automated Discovery of Cross-Plane Event-Based Vulnerabilities in Software-Defined Networking," in *Proceedings of the Network and Distributed System Security Symposium*. Internet Society, 2020.

[71] C. Scott, A. Wundsam, B. Raghavan, A. Panda, A. Or, J. Lai, E. Huang, Z. Liu, A. El-Hassany, S. Whitlock *et al.*, "Troubleshooting Blackbox SDN Control Software with Minimal Causal Sequences," in *Proceedings of the ACM Special Interest Group on Data Communication*. ACM, 2014.

[72] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying Network-Wide Invariants in Real Time," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*. USENIX, 2013.

[73] C. Röpke and T. Holz, "Preventing Malicious SDN Applications from Hiding Adverse Network Manipulations," in *Proceedings of the 2018 Workshop on Security in Softwarized Networks: Prospects and Challenges*, 2018.

[74] F. Xiao, J. Zhang, J. Huang, G. Gu, D. Wu, and P. Liu, "Unexpected Data Dependency Creation and Chaining: A New Attack to SDN," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 2020.

[75] R. Skowyra, L. Xu, G. Gu, V. Dedhia, T. Hobson, H. Okhravi, and J. Landry, "Effective Topology Tampering Attacks and Defenses in Software-Defined Networks," in *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2018.

[76] S. Jero, W. Koch, R. Skowyra, H. Okhravi, C. Nita-Rotaru, and D. Bigelow, "Identifier Binding Attacks and Defenses in Software-Defined Networks," in *Proceedings of the USENIX Security Symposium*. USENIX, 2017.

[77] 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*, 2013.

[78] E. Marin, N. Bucciol, and M. Conti, "An In-depth Look into SDN Topology Discovery Mechanisms: Novel Attacks and Practical Countermeasures," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2019.

[79] M. Yu, T. He, P. McDaniel, and Q. K. Burke, "Flow table security in sdn: Adversarial reconnaissance and intelligent attacks," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1519–1528.

[80] S. Liu, M. K. Reiter, and V. Sekar, "Flow Reconnaissance via Timing Attacks on SDN Switches," in *Proceedings of the International Conference on Distributed Computing Systems*. IEEE, 2017.

[81] M. Seo, J. Kim, E. Marin, M. You, T. Park, S. Lee, S. Shin, and J. Kim, "Heimdallr: Fingerprinting SD-WAN Control-Plane Architecture via Encrypted Control Traffic," in *Annual Computer Security Applications Conference*, 2022, pp. 949–963.

[82] T. A. Pascoal, I. E. Fonseca, and V. Nigam, "Slow denial-of-service attacks on software defined networks," *Computer Networks*, vol. 173, p. 107223, 2020.

[83] 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 Fefined Networks*. ACM, 2012.

[84] J. Miserez, P. Bielik, A. El-Hassany, L. Vanbever, and M. Vechev, "SDNRacer: Detecting Concurrency Violations in Software-Defined Networks," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*. ACM, 2015.

[85] A. El-Hassany, J. Miserez, P. Bielik, L. Vanbever, and M. Vechev, "SDNRacer: Concurrency Analysis for Software-Defined Networks," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2016.

[86] R. May, A. El-Hassany, L. Vanbever, and M. Vechev, "BigBug: Practical Concurrency Analysis for SDN," in *Proceedings of the Symposium on SDN Research*. ACM, 2017.

[87] K. Bu, Y. Yang, Z. Guo, Y. Yang, X. Li, and S. Zhang, "Flowcloak: Defeating middlebox-bypass attacks in software-defined networking," in *Proceedings of the IEEE Conference on Computer Communications*. IEEE, 2018.

[88] "Unverified Commits: Are You Unknowingly Trusting Attackers' Code?" https://checkmarx.com/blog/unverified-commits-are-you-unknowingly-trusting-attackers-code/, 2023.

[89] "ONOS Reactive Forwarding Application," https://github.com/opennetworkinglab/onos/blob/master/apps/fwd/src/main/java/org/onosproject/fwd/ReactiveForwarding.java, 2023.

[90] "CVE List for SDN Controllers," https://docs.google.com/spreadsheets/d/e/2PACX-1vRITg3P4lKXia-b66M6gHEFfBNnXl0sHYp_DxXgZh_

i0h2hFRQWQGmNBCmI7eI9qLBUgqBBbHttFJpD/pubhtml?gid=1659430278&single=true, 2023.

[91] "Configure OVS Connection Using SSL with Self-signed Certificates," https://docs.pica8.com/display/PicOS21116cg/Configure+OVS+Connection+Using+SSL+with+Self-signed+Certificates, 2023.

[92] R. Durner and W. Kellerer, "The Cost of Security in the SDN Control Plane," in *Proceedings of the ACM CoNEXT 2015-Student Workshop*. ACM, 2015.

[93] S. Shin and G. Gu, "Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)," in *Proceedings of the IEEE International Conference on Network Protocols*. IEEE, 2012.

[94] A. Wundsam, D. Levin, S. Seetharaman, and A. Feldmann, "OFRewind: Enabling Record and Replay Troubleshooting for Networks," in *Proceedings of the USENIX Annual Technical Conference*. USENIX, 2011.

[95] B. E. Ujcich, S. Jero, R. Skowyra, A. Bates, W. H. Sanders, and H. Okhravi, "Causal Analysis for Software-Defined Networking Attacks," in *USENIX Security Symposium*, 2021, pp. 3183–3200.

[96] A. Dwaraki, S. Seetharaman, S. Natarajan, and T. Wolf, "GitFlow: Flow revision management for software-defined networks," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, 2015, pp. 1–6.

[97] T. Ball, N. Bjørner, A. Gember, S. Itzhaky, A. Karbyshev, M. Sagiv, M. Schapira, and A. Valadarsky, "VeriCon: Towards Verifying Controller Programs in Software-Defined Networks," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2014, pp. 282–293.

[98] M. Canini, D. Venzano, P. Perešíni, D. Kostić, and J. Rexford, "A NICE Way to Test Openflow Applications," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*. USENIX, 2012.

[99] S. Son, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Model Checking Invariant Security Properties in OpenFlow," in *Proceedings of the IEEE international conference on communications*. IEEE, 2013.

[100] 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*. ACM, 2014.

[101] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight: Towards a Model-Driven SDN Controller Architecture," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*. IEEE, 2014, pp. 1–6.

[102] N. Katta, H. Zhang, M. Freedman, and J. Rexford, "Ravana: Controller Fault-Tolerance in Software-Defined Networking," in *Proceedings of the ACM SIGCOMM Symposium on Software Defined Networking Research*. ACM, 2015.

[103] B. Chandrasekaran, B. Tschaen, and T. Benson, "Isolating and Tolerating SDN Application Failures with LegoSDN," in *Proceedings of the Symposium on SDN Research*. ACM, 2016.

[104] S. R. Gomez, S. Jero, R. Skowyra, J. Martin, P. Sullivan, D. Bigelow, Z. Ellenbogen, B. C. Ward, H. Okhravi, and J. W. Landry, "Controller-Oblivious Dynamic Access Control in Software-Defined Networks," in *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2019.

[105] A. Kamisiński and C. Fung, "Flowmon: Detecting malicious switches in software-defined networks," in *Proceedings of the 2015 Workshop on Automated Decision Making for Active Cyber Defense*, 2015, pp. 39–45.

[106] P.-W. Chi, C.-T. Kuo, J.-W. Guo, and C.-L. Lei, "How to Detect a Compromised SDN Switch," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2015, pp. 1–6.

[107] P. M. Mohan, T. Truong-Huu, and M. Gurusamy, "Towards resilient in-band control path routing with malicious switch detection in SDN," in *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*. IEEE, 2018, pp. 9–16.

[108] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems*, 2014.

[109] V. B. Livshits and M. S. Lam, "Finding Security Vulnerabilities in Java Applications with Static Analysis," in *Proceedings of the USENIX Security Symposium*. USENIX, 2005.

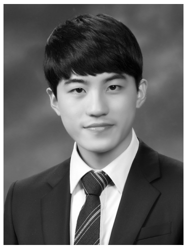
[110] "Security-Mode ONOS," https://wiki.onosproject.org/display/ONOS/Security-Mode+ONOS, 2023.

[111] "CORD: Central Office Re-architected as a Datacenter," https://opennetworking.org/cord/, 2023.

[112] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A survey on the security of stateful SDN data planes," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1701–1725, 2017.

[113] "Atomix:A reactive Java framework for building fault-tolerant distributed systems," https://atomix.io, 2023.

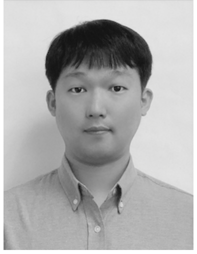
[114] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, 2014.



**Jinwoo Kim** is an assistant professor in the School of Software at Kwangwoon University, Seoul, South Korea. He received his Ph.D. degree in School of Electrical Engineering and his M.S degree in Graduate School of Information Security from KAIST, and his B.S degree from Chungnam National University in Computer Science and Engineering. His research topic ] focuses on investigating security issues with software defined networks and cloud systems.

**Minjae Seo** is a researcher at KAIST, Daejeon, South Korea. He received his M.S. degree from the Graduate School of Information Security at KAIST and his B.S. degree in Computer Engineering from Mississippi State University. His current research interests include Software-defined networking security, network fingerprinting, and deep learning-based network system.

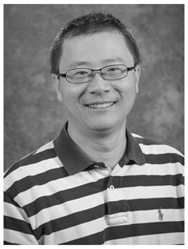
**Seungsoo Lee** is an assistant professor in the Department of Computer Science and Engineering at Incheon National University, Incheon, South Korea. He received his B.S. degree in Computer Science from Soongsil University in Korea. He received his Ph.D. degree and M.S. degree both in Information Security from KAIST. His research interests focus on cloud computing and network systems security. He is especially focusing his attention on software-defined networking (SDN), network function virtualization (NFV), containers, and its security issues.



**Jaehyun Nam** is an assistant professor in the Deaprtment of Computer Engineering at Dankook University, Youngin, Gyunggi-do, South Korea. He received his Ph.D. degree and M.S. degree both in School of Computing from KAIST. He received his B.S. degree in Computer Science and Engineering from Sogang University in Korea. His research interests focus on networked and distributed computing systems. He is especially interested in performance and security issues in cloud computing environments.

**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, and participates on multiple commercial company technical advisory boards. 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** received the Ph.D. degree in computer science from the College of Computing, Georgia Tech, in 2008. He is an currently an Associate Professor with the Department of Computer Science and Engineering, Texas A&M University (TAMU). He is currently Directing the SUCCESS (Secure Communication and Computer Systems) Lab, TAMU. He was a recipient of the 2010 NSF CAREER Award, the 2013 AFOSR Young Investigator Award, the Best Student Paper Award from 2010 IEEE Symposium on Security and Privacy (Oakland '10), the Best Paper Award from 2015 International Conference on Distributed Computing Systems (ICDCS '15), and the Google Faculty Research Award.

**Seungwon Shin** is an associate 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. He is currently a vice president at Samsung Electronics, leading the security team in the IT & Mobile Communications Division. His research interests span the areas of Software-defined networking security, IoT security, Botnet analysis/detection, DarkWeb analysis and cyber threat intelligence.