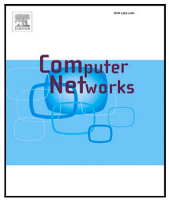
Computer Networks 241 (2024) 110203

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Review article

Enhancing security in SDN: Systematizing attacks and defenses from a penetration perspective

Jinwoo Kim [a,*], Minjae Seo [b], Seungsoo Lee [c], Jaehyun Nam [d], Vinod Yegneswaran [e], Phillip Porras [e], Guofei Gu [f], Seungwon Shin [g]

[a] School of Software, Kwangwoon University, Seoul, 01897, South Korea
[b] ETRI, Daejeon, 34129, South Korea
[c] Department of Computer Science & Engineering, Incheon National University, Incheon, 22012, South Korea
[d] Department of Computer Engineering, Dankook University, Yongin, Gyeonggi-do, 16890, South Korea
[e] SRI International, Menlo Park, CA, 94025, USA
[f] Texas A&M University, College Station, TX, 77843, USA
[g] School of Electrical Engineering, KAIST, Daejeon, 34141, South Korea

ARTICLE INFO

Keywords:
Software-Defined Networking (SDN)
SDN Security
Survey
Systematization of Knowledge (SoK)

ABSTRACT

Over the past 15 years, Software-Defined Networking (SDN) has garnered widespread support in research and industry due to its open and programmable nature. This paradigm enables various stakeholders, such as researchers, practitioners, and developers, to innovate networking services using robust APIs and a global network view, eliminating dependence on vendor-specific control planes. However, the adaptable architecture of SDN has introduced numerous security challenges not present in traditional network environments. While several surveys have highlighted existing attacks, there is a notable absence of a systematic penetration perspective, essential for understanding the attacks and their origins. This paper seeks to analyze prior literature that has exposed instances of attacks in SDN, examining their vulnerabilities, penetration routes, and root causes. Furthermore, we offer a thorough and comprehensive discussion of the underlying issues associated with these attacks, presenting defenses proposed by researchers to mitigate them and analyzing how the root causes are addressed. We also explore how our survey can assist practitioners in preparing suitable defenses by providing insights into penetration routes. Through this study, our goal is to shed light on existing security issues within the current SDN architecture, prompting a reassessment of various security problems and offering a guideline for future research in SDN security.

1. Introduction

Software-Defined Networking (SDN) has emerged as a dominant networking paradigm over the last 15 years, revolutionizing traditional network infrastructure. Initially introduced by the 4D project [1], the concept of decoupling control and data planes has garnered significant attention from researchers seeking to overcome the limitations of conventional networks that hindered innovation. The centralized control plane, known as an *SDN controller*, has enabled unprecedented advancements by providing programmable interfaces (e.g., OpenFlow [2]), impactful applications (e.g., FlowVisor [3]), and flexible components (e.g., Open vSwitch [4]). As a result, SDN has gained widespread adoption, with extensive studies and deployments ranging from campus networks [5] to large-scale networks such as WANs [6] and data centers [7,8].

Furthermore, SDN has garnered significant attention in the realm of network security, presenting several advantages, including heightened security capabilities. The centralized control offered by an SDN controller empowers network operators to craft native security systems surpassing the efficacy of traditional middle-box-based approaches. Various techniques for early detection and proactive mitigation have been proposed by researchers [9], demonstrating the potential of SDN-based security systems. These techniques span areas such as botnet detection [10], DDoS mitigation [11], and network forensics [12]. Moreover, SDN seamlessly integrates with existing middle-boxes through service chain context, ensuring compatibility with legacy markets [13,14].

Nevertheless, it is crucial to recognize that the novel architecture of SDN introduces potential vulnerabilities that attackers may exploit across all SDN components. As depicted in Fig. 1, SDN transforms the

* Corresponding author.
E-mail addresses: jinwookim@kw.ac.kr (J. Kim), ms4060@etri.re.kr (M. Seo), seungsoo@inu.ac.kr (S. Lee), jaehyun.nam@dankook.ac.kr (J. Nam), vinod@csl.sri.com (V. Yegneswaran), porras@csl.sri.com (P. Porras), guofei@cse.tamu.edu (G. Gu), claude@kaist.ac.kr (S. Shin).

https://doi.org/10.1016/j.comnet.2024.110203
Received 24 July 2023; Received in revised form 18 January 2024; Accepted 19 January 2024
Available online 23 January 2024

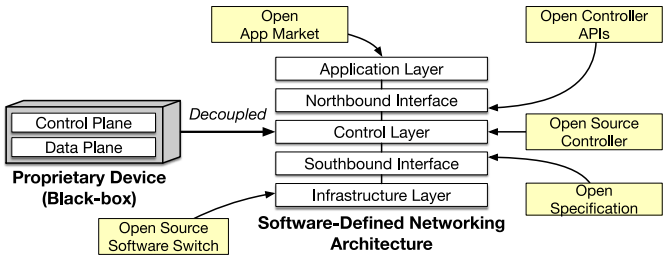


Fig. 1. An illustration of the layered SDN architecture: decoupling the control and data planes from proprietary network devices.

rigid architecture of proprietary devices into multiple layers that can be extended by various stakeholders. In this context, the open-source community plays a significant role in shaping the specifications and implementations of diverse SDN layers and interfaces. While this collaborative approach fosters a variety of contributions towards innovative applications, flexible APIs, and practical use cases, it also results in a complex SDN architecture, posing challenges in tracking internal event flows. This complexity, in turn, creates opportunities for attackers to establish *penetration routes* through any SDN component.

As a result, attackers have successfully infiltrated various layers that were previously inaccessible, leading to heightened security risks. Particularly noteworthy is the absence of a trusted ecosystem for application developers and operators, which has facilitated the proliferation of malicious applications in application markets [15–17]. These malicious applications carry out harmful actions against controllers and switches. Additionally, the centralized architecture of the SDN controller, operating as a general network operating system (NOS), is susceptible to straightforward application-level penetration [18–20]. Furthermore, the centralized control plane is vulnerable to saturation attacks through the exploitation of switch-to-controller penetration [21–23]. Numerous attacks have been reported; hence, it is imperative for practitioners to acknowledge these vulnerabilities and work towards developing secure SDN environments.

This paper aims to comprehensively investigate the security implications of the SDN architecture, *focusing on the penetration of SDN layers by attackers and corresponding defense strategies employed by defenders*. To accomplish this objective, we conduct an extensive survey of reputable research literature from networks, security, and systems conferences and journals. Through our analysis, we propose a taxonomy for categorizing SDN attacks, considering their penetration direction and presenting their root causes, affected components, and common attack types. Additionally, we evaluate existing countermeasures recommended by researchers to mitigate these attacks. By conducting a thorough examination of the identified attacks and defenses, we shed light on the architectural vulnerabilities present in SDN and identify areas that require further attention from the security research community in future studies.

Numerous surveys have been conducted to systematically organize and elucidate the various aspects of security in SDN and its associated attacks [24–36]. These existing surveys predominantly focus on categorizing attacks *based on layers*, as discussed in Section 3, including a detailed comparison with our work and existing surveys. In contrast, our paper introduces a novel taxonomy termed as the *penetration route* which provides a granular analysis of the attack vectors, tracing the trajectory of an attacker's infiltration from the source to the targeted component within the SDN architecture. The significance of this conceptual framework is twofold. Firstly, it offers a strategic advantage to network defenders by elucidating the specific layers and components that are particularly susceptible to exploitation. This understanding is crucial as it enables defenders to not only pinpoint critical vulnerabilities but also to tailor their defensive strategies more effectively, ensuring a robust security posture. Secondly, our penetration route taxonomy serves as a predictive tool, enabling defenders to foresee and prepare for potential attacks on targeted layers and components that, thus far, remain unexploited.

Contributions. Our contributions are outlined as follows:

- Introduction of a novel attack taxonomy from a penetration perspective, derived from comprehensive reviews of previous literature.
- Systematic classification of literature related to SDN attacks and defenses based on the established taxonomy.
- In-depth analysis of existing attacks and defenses, including examination of penetration routes, root causes, and SDN components.
- Thorough discussion providing insights for each type of attack and defense, along with use cases and directions for future research.

Organization. The remainder of this paper is structured as follows: Section 2 provides an overview of SDN. Section 3 examines previous surveys on SDN security and discusses their limitations. Section 4 introduces our taxonomy for categorizing existing literature. Section 5 and Section 6 classify SDN attacks and defenses, respectively, based on the proposed taxonomy. In Section 7, we discuss potential use cases of the penetration routes we present in this paper. In Section 8, we explore future research directions for SDN security. Finally, Section 9 presents the conclusion of our paper.

2. Background

In this section, we provide background information on Software-Defined Networking (SDN) and offer a brief introduction to its architecture, including a discussion on the trust and threat model.

2.1. What is Software-Defined Networking (SDN)?

In traditional networks, the insertion of new functions into devices is inherently challenging due to the embedded nature of the control plane and data plane within proprietary network devices [2]. To address this fundamental issue, Software-Defined Networking (SDN) introduces a new paradigm that underscores the decoupling of the control plane from the data plane. This separation is achieved through a logically centralized controller operated on high-performance commodity hardware.

Fig. 2 illustrates the overall architecture of SDN. The *application layer* hosts various SDN applications (or apps) designed for network management. In the middle, the *control layer* comprises one or multiple SDN controllers responsible for controlling the underlying forwarding devices and managing the centralized network view. At the bottom, the *infrastructure layer* consists of distributed forwarding devices that directly handle incoming packets. Specifically, the control layer and the infrastructure layer communicate through the *control channel*, while hosts exchange packets with switches through the *data channel*. In addition to these channels, the controller features three distinct types of interfaces: (i) the *northbound interface (NBI)* for communication with applications, (ii) the *east/westbound interface* for state synchronization with neighboring controllers, and (iii) the *southbound interface (SBI)* for switch management.

2.2. SDN controller and application

SDN controllers on the control layer are commonly referred to as network operating systems (NOS). This nomenclature arises from the fact that advanced controllers typically incorporate fundamental control software essential for operating and managing an entire network. Additionally, they offer a comprehensive network view for SDN apps by abstracting away intricate control logic details. Controllers typically

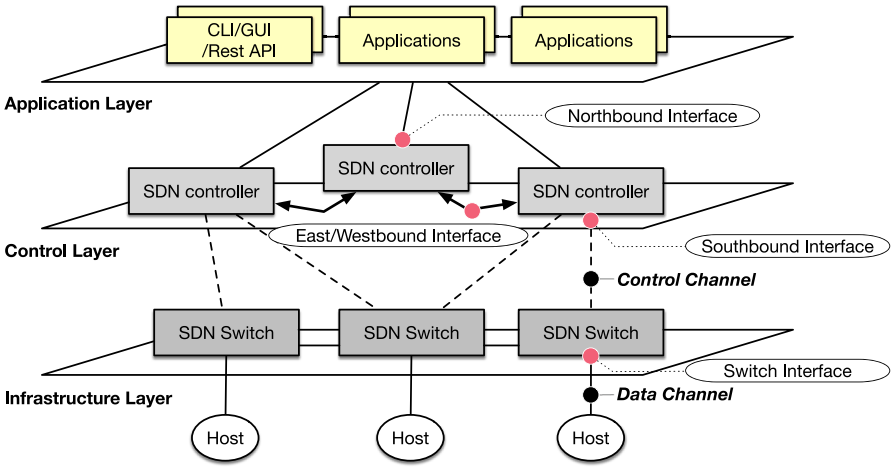


Fig. 2. SDN layers, components, channels, and interfaces.

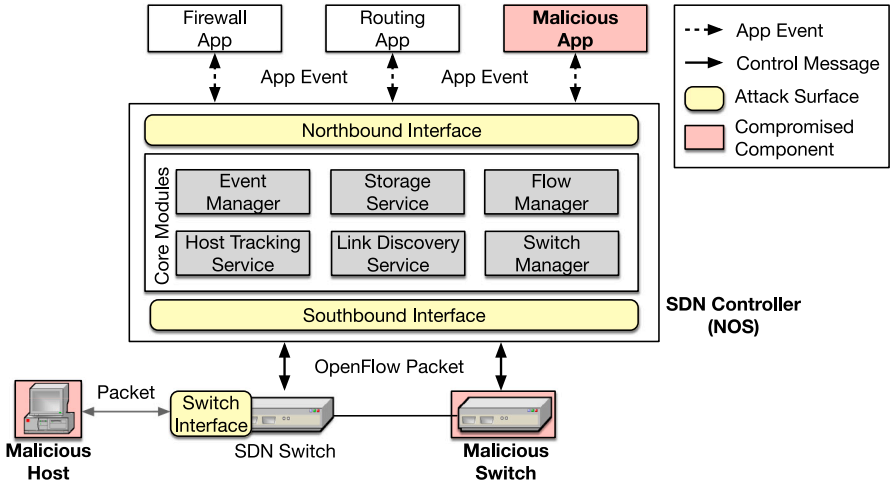


Fig. 3. The internal architecture of an SDN controller and our threat model.

encompass core modules and interfaces essential for tasks such as topology detection and traffic management, as depicted in Fig. 3.

Many SDN controllers, including NOX [37], Maestro [38], Onix [39], Floodlight [40], Beacon [41], ONOS [42], and OpenDaylight [43], typically consist of four modules equipped with flexible APIs to enhance programmability. These modules encompass (i) a topology manager featuring link discovery and host tracking services, (ii) a switch manager responsible for maintaining up-to-date topology information through the discovery of network elements, (iii) a storage service securely storing all essential network information and providing accessibility to SDN applications, and (iv) a flow manager defining and modifying flow rules in switches, utilizing the southbound interface. By incorporating these four modules, operators can develop a variety of network applications at a reduced cost and optimize them in alignment with their specific policies.

2.3. OpenFlow and SDN switch

The most widely used southbound interface between the control layer and the infrastructure layer is OpenFlow [44]. It defines commands and behaviors that enable the controller to perform fine-grained and dynamic policy enforcement in OpenFlow-enabled SDN switches. The switch maintains a set of flow tables, each managing a set of flow rules. When an incoming packet arrives at the switch without a matching flow rule entry, the switch sends a PACKET_IN message, providing partial information about the packet to the controller. Subsequently, 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.

2.4. Trust and threat model in SDN

Typically, the SDN controller holds a central role, widely recognized as the pivotal component, leading to the establishment of a trust boundary around the control layer. The application layer, however, is characterized by a relatively weaker level of trust due to the potential for applications to leverage the controller's capabilities via northbound APIs. In contrast, the infrastructure layer commands a higher degree of trust, primarily attributed to the constrained capabilities of SDN switches compared to their legacy counterparts.

In this context, the commonly adopted threat model in SDN assumes the presence of (i) malicious applications in the application layer, (ii) malicious switches, and (iii) malicious hosts in the infrastructure layer (see Fig. 3). As discussed later, this is possible due to various reasons, such as downloading compromised applications and switch firmware or host vulnerabilities.

Table 1
Previous surveys and their limitations.

Previous surveys	Main focus	Focused layers and interfaces	Limitations	Unveiling root cause	Analysis of penetration route
Kreutz et al. [24] (2013)	Overall security issues in SDN	Application and control layer	Narrow focus on security issues regarding trust between SDN applications and controllers.	✗	✗
Scott et al. [25] (2015)	Overall security issues in SDN	Application and control layer	Limited coverage of attack types, providing a limited perspective on the SDN security.	✗	✗
Ahmad et al. [26] (2015)	Overall security issues in SDN	Application, control, and infrastructure layer	Lack of detailed descriptions and analyses of attacks	✗	✗
Alsmadi et al. [27] (2015)	Overall security issues in SDN	Application and control layer	Inadequate definition of defense criteria limiting the research's scope for practical applications.	✗	✗
Yan et al. [28] (2015)	DDoS attacks in SDN and cloud computing environments	Application and control layer	Constrained analysis focusing solely on a single 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 interfaces	Potential neglect of important security issue dimensions due to the focus on a specific SDN topology discovery method.	✗	✗
Yoon et al. [30] (2017)	Overall security issues in SDN and attack demonstration	Application, control, and infrastructure layer	Limited in-depth discussion for attack trend and root causes.	✓	✗
Shaghaghi et al. [31] (2020)	Security issues in SDN data plane	Infrastructure layer	Limited examination of attack and mitigation scenarios, solely focused on the SDN data plane.	✗	✗
Chica et al. [32] (2020)	Overall security issues in SDN	Application layer, control layer, infrastructure layer, and interfaces	Lack of detail in the study's proposed classification of attack criteria and taxonomy.	✗	✗
Rauf et al. [33] (2021)	Northbound interface security issues in SDN	Northbound interface	Narrow focus on northbound interface vulnerabilities limiting the perspective on SDN security.	✗	✗
Jimenez et al. [34] (2021)	Defense solutions for SDN layers	Application layer, control layer, infrastructure layer, and interfaces	Lack of an attack taxonomy.	✗	✗
Rahouti et al. [35] (2022)	Overall security issues in SDN	Application layer, control layer, and infrastructure layer	Lack of root cause analysis.	✗	✗
Melis et al. [36] (2023)	Utilizing quantitative metrics and qualitative insights to identify key issues in SDN	Application layer, control layer, infrastructure layer, and interfaces	Lack of in-depth analysis associated with SDN security.	✗	✗
Our work	Overall security issues in SDN	Application layer, control layer, infrastructure layer, and interfaces	–	✓	✓

3. Related work

Over the years, numerous studies have conducted surveys and analyses on the security aspects of SDN. One influential study that advanced the understanding of SDN security is the work by Kreutz et al. [24], published in the 2013 HotSDN (Hot Topics in Software Defined Networking) conference.[1] This study played a crucial role by being the first to introduce seven potential threat vectors specific to SDN, contributing significantly to the early development of the field.

In the realm of SDN, several comprehensive surveys have been conducted, each proposing its own taxonomy and providing valuable insights. Scott et al. [25] conducted an analysis of SDN vulnerabilities, categorizing them based on different types of attacks. Rahouti et al. [35] reviewed previous literature and classified it using various attack taxonomies. Chica et al. [32] analyzed overall security issues in SDN by surveying known SDN attacks and defenses. Melis et al. [36] utilized quantitative metrics associated with the literature to derive insightful findings. These surveys have made significant contributions to enhancing our understanding and advancing the field of SDN security.

In addition to the aforementioned surveys, several works have applied existing standard or well-known threat models to analyze SDN attacks and defenses. Ahmad et al. [26] examined the security solutions for SDN based on ITU-T (ITU Telecommunication Standardization Sector) recommendations. Alsmadi et al. [27] and Jimenez et al. [34] surveyed the threats and defense solutions for SDN, categorizing them using the STRIDE (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege) model. Yoon et al. [30] conducted a survey of known SDN attacks, identified new vulnerabilities, and mapped them to the CIA (Confidentiality, Integrity, Availability) model. These works provide valuable insights into the application of established threat models to understand SDN security challenges.

Furthermore, certain works focused on specific environments or layers of SDN to analyze how SDN affects them. Khan et al. [29] analyzed SDN topology discovery and its associated threats. Rauf et al. [33] focused on the security issues of northbound interfaces in SDN. Yan et al. [28] analyzed DDoS (Distributed Denial-of-Service) attacks in SDN and cloud computing environments. Shaghaghi et al. [31] revisited the security issues in the infrastructure layer.

Despite these efforts, previous surveys on SDN security have been limited in their scope, hindering a comprehensive understanding of the field. As shown in Table 1, these limitations include a narrow focus on security issues between SDN applications and controllers, inadequate descriptions and analyses of attacks, insufficient criteria for evaluating attacks and defenses, absence of an attack taxonomy and root cause analysis, and limited in-depth analysis of SDN security. In contrast, our study represents *a significant advancement by addressing the major root causes of SDN attacks, proposing a new taxonomy for attacks and defenses, and providing thorough analysis*. Specifically, we classify existing attacks *vertically* across layers, clearly identifying the source and target components involved in a penetration route. Additionally, we classify existing defenses *horizontally*, enabling us to understand the implementation of defenses at each layer. These approaches contribute to a more comprehensive understanding of SDN security and bridge the gaps left by previous research efforts.

4. Systematization taxonomy

This section introduces our taxonomy to classify existing attacks and countermeasures. For each criterion, we elaborate on the key reasons behind choosing them with the discussion of security challenges. We refer the readers to Fig. 3 for a better understanding of the threat model of taxonomy in the SDN architecture.

[1] HotSDN is considered the predecessor of SOSR (Symposium on SDN Research).

4.1. 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. Note that in Section 5, we classify attacks based on *five* penetration directions that are derived from various penetration routes. A typical penetration route needs at least one *source* to affect a *target* through a sequence of the following components and interfaces:

4.1.1. Application

Malicious SDN applications have been a popular stepping stone to penetrate SDN controllers and switches [15,18–20,45,46], similar to the way an Android malware infiltrates a user's mobile device [47].

4.1.2. Northbound Interface (NBI)

We scope the NBIs into all accessible interfaces of a controller from SDN applications, such as system APIs (e.g., system calls, Java native methods), controller core services [30,48], peer application services [45,48], and REST APIs [49].

4.1.3. Controller

An SDN controller is the most important target for attackers and defenders, and thus it can be targeted by any other components.

4.1.4. Southbound Interface (SBI)

A southbound interface is a boundary where a controller and switches communicate with each other. So, it can be abused to affect controller and switch operation.

4.1.5. 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 [50,51].

4.1.6. 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.

4.1.7. 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. An attacker can compromise one of those hosts to use them as an attack source.

4.2. Attack type

In this paper, we categorize six types of attacks that are commonly found in literature.

4.2.1. Denial of service (DoS)

It refers to all circumstances when either a controller or switch is unavailable or its performance is significantly downgraded due to attacks. As those components are crucial in operating SDN architecture, many related attacks have been proposed.

4.2.2. Information leakage

Maintaining the confidentiality of SDN control-plane information, encompassing policies, configurations, and architecture, is of utmost importance. Nevertheless, there exists a potential risk of data leakage through the measurement of metadata, particularly concerning latency in communication on the control channel.

4.2.3. Protocol abusing

This attack involves the exploitation of protocol features within SDN for malicious purposes. For instance, a malicious application could disrupt communication between the controller core modules and other applications. Similarly, a malicious switch could inject harmful payloads into control packets.

4.2.4. Man-in-the-middle

The initial SDN architecture is designed under the assumption that components are trustworthy. However, a compromised application, switch, or host could perform man-in-the-middle attacks between communications to manipulate, drop, or inject messages.

4.2.5. Policy evasion

This attack refers to a situation in which a malicious application or host violates a security policy. Such policies can encompass both control plane policies and configurations, as well as data plane forwarding behavior.

4.2.6. Storage poisoning

Due to the lack of filtering or access control, SDN's network storage is susceptible to storage poisoning attacks, where malicious data is injected, compromising the controller's decision-making process.

4.3. Defense type

To defend SDN components from attacks and remove their vulnerabilities, diverse countermeasures have been proposed so far. We classify them into the following six defense types. Note that in Section 6, we also categorize those defenses into four layers: application, control, infrastructure, and cross-layer. This classification is based on whether defenses are implemented within a single layer or require the collaboration of multiple layers.

4.3.1. Authentication

This defense strategy focuses on implementing a mechanism to identify and establish trust in components within the SDN architecture. One example of this is the deployment of a public key infrastructure (PKI), which enables the secure exchange of cryptographic keys and certificates.

4.3.2. Access control

This defense refers to the design and implementation of an access control system for SDN. For example, a permission system is suitable for the application and infrastructure layer where many untrusted entities like third-party applications or unknown hosts reside.

4.3.3. Patch/extension

This defense strategy involves extending the prior architecture, implementations, and functionality of SDN components with additional features. For instance, it includes the incorporation of security modules into existing controllers or the development of new controllers that are architecturally secure.

4.3.4. Testing

This defense strategy focuses on identifying vulnerabilities and flaws within the implementation of SDN applications, controllers, or switches. Since the software-defined logic forms the basis of SDN operations, the reliability of the system greatly relies on the absence of any flaws in the implementation of SDN components.

4.3.5. Program analysis

This defense 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 or dynamic instrumentation investigating execution traces.

4.3.6. Monitoring

This defense strategy involves inspecting the behavior of SDN components and detecting abnormal cases that violate network policies. It aims to identify and respond to deviations from expected behavior within the SDN system, enabling the detection and prevention of potential security breaches.

4.4. 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 paid great attention to.

4.4.1. Lack of NBI authorization

This criterion indicates the absence of an authorization measure in SDN northbound interfaces (NBI). Despite their critical impact on the entire network operations in case of misuse by malicious applications, they are not properly protected from malicious intents [18–20] or human errors [49].

4.4.2. Lack of SBI authorization

The southbound interface (SBI) is a critical boundary that can immediately affect 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 [17,52].

4.4.3. 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 [30]. When processed, their integrity should be guaranteed to preserve the original messages or finish the chain sequence without unintended modifications.

4.4.4. 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 control channel (e.g., SSL/TLS) is recommended [44] to avoid such a situation. However, it is possible that operators do not employ SSL/TLS due to its performance issue, giving an attacker a chance to monitor and manipulate control messages [53].

4.4.5. 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.

4.4.6. 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 [44], and contemporary SDN controllers also do not support data-plane authentication for switches and hosts.

4.4.7. 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 [18,54].

4.4.8. Side channel

The central principle of SDN's design philosophy is to separate the control plane (i.e., controller) from the data plane, necessitating a communication channel between them. However, this communication channel introduces a novel attack surface. For instance, even an attacker without direct access to the channel can still exploit it to gather confidential information from the SDN control plane, including network policies, through fingerprinting techniques [55,56].

4.4.9. Implementation flaw

While there is a clear standard reference for SDN (e.g., OpenFlow [44]), it does not mean that the implementations of such references are always clean, with no bugs or 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 [49].

5. SDN attack classification

In this section, we present major categories of attacks and vulnerabilities that have been discussed in academia. Table 2 shows the summary of systematization for disclosed SDN attacks, enumerated with *five* penetration directions.[2] Finally, we summarize our findings and insights for each.

5.1. Application to controller

Rationale behind the penetration route. By default, SDN applications have access to all northbound interfaces of a controller. This design decision is based on the assumption that these applications will not possess malicious intent or contain critical bugs. However, in practical scenarios, it is impossible to provide complete guarantees regarding the behavior of SDN applications. Most application-to-controller penetration begins with the installation of a malicious application on a controller (downloaded from a third-party app store or deployed by a network operator). If succeeds, it invokes APIs (e.g., REST APIs, Java/Python APIs) supported by northbound interfaces of the controller, allowing the malicious application to perform harmful actions. For example, by invoking `ApplicationAdminService` in ONOS, the malicious application can activate and deactivate other applications running on the controller. While this is a relatively simple route, its impact is powerful as discussed in the following sections.

5.1.1. Denial of service

Exploiting northbound interfaces can have detrimental effects on controller functionality and network performance. Malicious applications can abuse *system APIs* directly associated with the runtime operation of the controller, leading to various issues. For instance, Shin et al. [18] demonstrate the possibility of unauthorized termination of Java-based SDN controllers (such as Floodlight, ONOS, and OpenDaylight) by invoking the `System.exit()` function (Fig. 4). Yoon et al. [30] show that manipulating time variables can result in the disconnection of switches from the controller.

Furthermore, a malicious application can illicitly remove link information from storage, impeding the controller's ability to determine correct routing paths [18]. Additionally, certain APIs can be exploited to exhaust system resources. Shin et al. [18] demonstrate that a malicious application can allocate large-sized data structures to deplete controller memory. AIM-SDN [57] reveals the possibility of flooding a controller's storage by generating numerous configuration entries.

[2] Note that ○ is the source, → is the middle, and ● is the target component of a penetration route in Table 2.

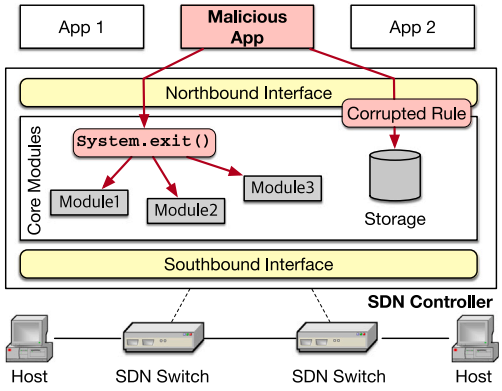


Fig. 4. An illustration of system API abusing and corrupted flow rule injection attacks in application-to-controller penetration. The malicious application can invoke `System.exit()` to terminate the controller modules or inject a corrupted rule into the storage.

Moreover, a malicious application can manipulate critical configurations within a controller. For example, the `fwd` app [84] in the ONOS controller allows operators to configure the `packet_out_only` option, which enables packet forwarding without rule installation. However, this option can be misused to unnecessarily redirect all packets to the controller, significantly degrading network performance [19,58].

5.1.2. Storage poisoning

Malicious applications can exploit northbound interfaces to compromise controller storage, leading to network malfunctions (Fig. 4). Dixit et al. [57] introduce a vulnerability where a malicious application directly corrupts a storage entry using northbound interfaces, resulting in a controller crash. They show that if an application injects many flow rules into a controller, expired flows end up flooding the controller storage even if a timeout is set (CVE-2017-1000411). This vulnerability is so-called *semantic gap*, indicating that the controller does not properly synchronize states (e.g., flow rules) with those in a switch. It causes other critical bugs that are reported to the vendor as well (CVE-2017-1000406, CVE-2018-1078).

Lee et al. [49] present a similar vulnerability with the incorrect rule installation via REST APIs due to implementation gaps between controllers and switches. For example, if an operator creates a flow rule that utilizes the `tcp_dst` match field without specifying the prerequisite `ip_proto` match field for TCP headers, the Floodlight SDN controller updates the storage with the rule. However, the switch rejects the rule due to the missing prerequisite field. Consequently, the Floodlight controller retains the rule in its storage, causing an infinite sequence of unsuccessful synchronization attempts and significantly consuming controller resources (CVE-2019-1010252). Similar vulnerabilities were reported to the vendor as well (i.e., CVE-2019-1010249, CVE-2019-1010250, CVE-2020-18683, CVE-2020-18684, CVE-2020-18685).

Moreover, a malicious application can poison existing storage entries (Fig. 5). Most SDN controllers operate as *event-driven* systems, notifying applications when specific events occur and maintaining an *event list*. Compromising this list can hinder the proper reception of subscribed events by applications [19,30,58]. Similarly, a malicious application can remove configurations to trigger unexpected bugs.

Additionally, attackers can inject a *rootkit* that remains hidden and continues performing malicious actions. Ropke et al. [20] demonstrate that an SDN rootkit application can remove its app ID from controller storage. Subsequently, the rootkit establishes a covert channel to steal sensitive information, such as configurations, from the target controller.

Table 2
Systematization of SDN attacks by penetration direction. (✓ Root Cause ○ Source → Middle ● Target)

Direction	Type	Attack	Root cause									Penetration route						
			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
Application to Controller (Section 5.1)	Denial of Service	System API Abusing [18,30]	✓									○	→	●				
		Controller Resource Exhaustion [18,57]	✓						✓			○	→	●				
		App Configuration Manipulation [19,58]			✓							○	→	●				
		Topology Information Removal [18]	✓									○	→	●				
	Storage Poisoning	Malformed Configuration Injection [57]			✓							○	→	●				
		Corrupted Flow Rule Injection [49]			✓						✓	○	→	●				
		Event Unsubscription [19,30,58]	✓									○	→	●				
		Rootkit Injection [20]					✓					○	→	●				
	Man-in-the-Middle	Event Hijacking [19,30,58]			✓							○	→	●				
	Policy Evasion	Cross-app Poisoning [45]	✓				✓					○	→	●				
Application to Switch (Section 5.2)	Denial of Service	FLOW_MOD Flooding [30]	✓	✓			✓					○	→	→	→	●		
		Malformed Control Message Injection [30]			✓			✓				○	→	→	→	●		
	Protocol Abusing	Switch Firmware Abusing [30]									✓	○	→	→	→	●		
		Dynamic Tunneling [17,59]	✓				✓				✓	○	→	→	→	●		
		Buffered Packet Hijacking [46]	✓		✓						✓	○	→	→	→	●		
Switch to Controller (Section 5.3)	Protocol Abusing	Payload Injection [60]			✓	✓								●	←	○		
		Abnormal Protocol Behavior Injection [61,62]				✓		✓						●	←	○		
		Malformed OpenFlow Packet Injection [58,61,63]				✓		✓						●	←	○		

(*continued on next page*)

5.1.3. Man-in-the-middle

Contemporary SDN controllers employ an *ordered list* that determines the priority of event delivery to different applications. However, this event delivery mechanism can be exploited by attackers. An attacker may seize an event before it reaches other applications, enabling them to manipulate the event's payload [58] or even discard it entirely [19,30]. Consequently, critical applications like a routing application may fail to receive essential events, such as topology changes.

5.1.4. Policy evasion

Ujcich et al. [45] have introduced a cross-app poisoning attack that involves a malicious application poisoning the storage to influence the decision-making of other applications. For example, the attacker's application could inject a PACKET_READ event into the controller, masquerading as the victim's MAC address. As a result, the controller updates the host-to-location pair maintained by the host tracking service based on the manipulated packet. Consequently, a forwarding application may install a rule that redirects the victim's traffic to the attacker, violating established security policies.

Table 2 (*continued*).

Direction	Type	Attack	Root cause									Penetration route						
			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
Host to Controller (Section 5.4)	Denial of Service	PACKET_IN Message Flooding [21–23,64–67]							✓					●	←	←	←	○
		READ_STATE Message Flooding [68]							✓					●	←	←	←	○
		Race Condition [54]									✓			●	←	←	←	○
	Storage Poisoning	Link Fabrication Attack [52,67,69]				✓		✓						●	←	←	←	○
		Host Identifier Spoofing [52,67,69,70]				✓					✓			●	←	←	←	○
		Malformed LLDP Packet Injection [71]				✓		✓						●	←	←	←	○
	Policy Evasion	Cross-Plane Attack [72]			✓	✓		✓						●	←	←	←	○
Host to Switch (Section 5.5)	Information Leakage	Slow Path Fingerprinting [21,73]								✓						●	←	○
		Policy Fingerprinting [55,56,74,75]								✓						●	←	○
		App Fingerprinting [76]								✓						●	←	○
		Topology/Protocol Fingerprinting [77]								✓						●	←	○
		In-band Channel Fingerprinting [78]								✓						●	←	○
	Man-in-the-Middle	Control Channel MitM [30,61]				✓		✓								●	←	○
		Middlebox Tag Manipulation [79]						✓								●	←	○
	Denial of Service	Flow Table Overloading [21,64,67,68,80]							✓	✓						●	←	○
		Switch Race Condition [81–83]									✓					●	←	○
		Switch Remote Code Execution [51]									✓					●	←	○
		In-band Channel Flooding [78]								✓						●	←	○

Table 3
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	○

Summary. Overall, there are numerous attack scenarios in which SDN applications are abused for penetrating controllers. The root cause contributing to these attacks is the inadequate security measures employed in the northbound interfaces of the controllers, resulting in a significant impact on controller operation.

Why is application-to-controller penetration possible? 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 because, 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 [15] would be popular, akin to the Docker Hub. This would have made it easy

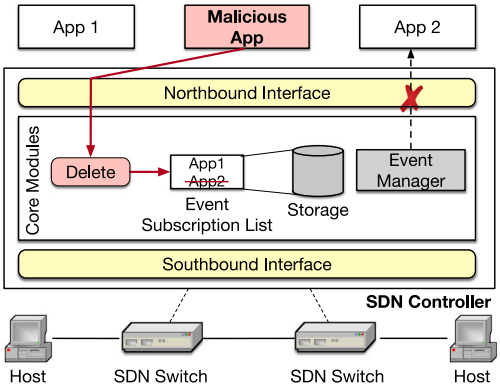


Fig. 5. An illustration of the event unsubscription attack [19,30,58] in application-to-controller penetration. The malicious application deletes the app-2 from the event subscription list so that it cannot receive an event anymore.

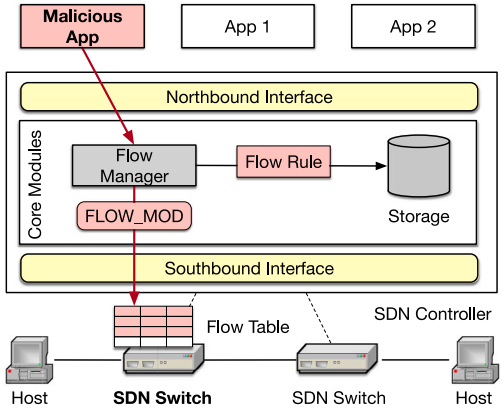


Fig. 6. An illustration of the FLOW_MOD flooding attack [30] in application-to-switch penetration. The malicious application forces the flow manager to install many flow rules on a target SDN switch.

for an attacker to deploy malicious SDN applications. 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 commercial 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 the deployment of malicious SDN applications 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 [85]. 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 3, 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-to-controller penetration? 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 that is complicated to detect.

5.2. Application to switch

Rationale behind the penetration route. SDN applications utilizing northbound interfaces possess significant capabilities that extend beyond controllers to include switches. Specifically, when high-level messages are transmitted from applications through northbound interfaces, they undergo conversion into low-level messages, subsequently being transmitted to switches via southbound interfaces. Thus, a malicious application installed on a controller can call an API of a northbound interface that is capable of interacting with switches. For example, `FlowRuleService` in ONOS is responsible for installing flow rules on a switch. With this, the malicious application can guide the controller to install a flow rule, which is performed by a device-specific handler located in southbound interfaces. This mechanism can be exploited to enable applications to carry out diverse malicious actions on switches.

5.2.1. Denial of service

Switch TCAM (Ternary Content Addressable Memory) is a critical resource that should be carefully managed. As they are normally scarce in switch 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, a malicious app can invoke a massive number of FLOW_MOD messages with distinct match fields, causing a switch to install many different rules [30] (Fig. 6). Furthermore, an attacker can take advantage of the absence of an exception-handling mechanism in a particular switch implementation. For instance, if a malicious application sends a malformed OpenFlow packet with an invalid packet length, it can lead to a failure in the switch's handling process, resulting in disconnection from the controller [30].

5.2.2. Protocol abusing

OpenFlow is a de-facto standard protocol that specifies controller-switch channels, enabling any switch to communicate with a controller. However, an attacker can abuse its protocol features to conduct malicious actions on switches.

An attacker can abuse the mechanism of packet matching in OpenFlow. For example, the *switch firmware abusing attack* [30] shows that a malicious application can deliberately replace the match fields of flow entries with unsupported ones by the hardware (e.g., MAC addresses), causing packet matching to be processed by the software stack. This abuse takes advantage of the fact that some switches do not support all OpenFlow match fields. Consequently, it significantly degrades packet processing performance.

In addition, an attacker can exploit OpenFlow dynamic actions to bypass security policies. Porras et al. [17,59] demonstrate that a malicious app can abuse the OpenFlow `Set` action to violate network invariants, 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 middleboxes. 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 existing controllers orchestrate this contention.

Finally, a vulnerability within the packet forwarding logic can lead to a critical security breach. When a switch triggers a PACKET_IN message, it temporarily stores an incoming packet in a switch buffer

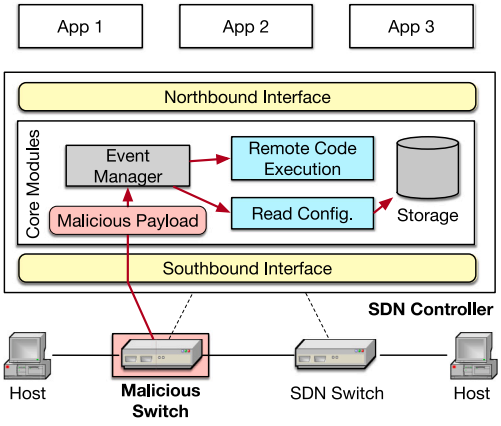


Fig. 7. An illustration of the payload injection attack [60] in switch-to-controller penetration. The malicious switch sends a control message that has a malicious payload, which is executed in the controller internally.

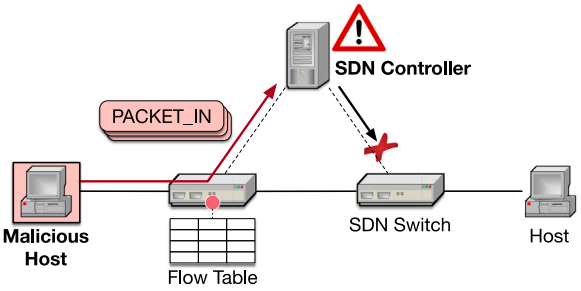


Fig. 8. An illustration of the *PACKET_IN flooding attack* in host-to-controller penetration. When the malicious host sends packets whose rules are not installed in the switch, they are forwarded to the controller by being encapsulated to PACKET_IN. If the controller receives too many PACKET_INs, it cannot handle other requests.

and assigns it a buffer ID, awaiting the controller's instruction. This ID is used to retrieve the packet when the controller instructs the switch to forward it. Cao et al. [46] introduce a *buffered packet hijacking attack* that exploits the fact that OpenFlow switches only examine the buffer ID, ignoring other match fields when forwarding buffered packets. They demonstrate that a malicious application can hijack these buffered packets by using the same buffer ID. If a switch receives a FLOW_MOD or PACKET_OUT message with the same buffer ID, it considers them valid control messages. Consequently, the malicious application can illegitimately forward a packet, allowing it to bypass security policies. This vulnerability arises from the lack of an explicit requirement in the OpenFlow switch specification regarding strict checking of match fields when handling buffered packets [44].

Summary. It is evident that SDN applications can be exploited to abuse protocol implementations, leading to unforeseen actions by switches. As a consequence, switches can malfunction or violate established policies.

Why is application-to-switch penetration possible? There are several reasons behind these vulnerabilities. First, the switch itself often fails to manage resources carefully since most of its functions have been migrated to a controller. Consequently, if the controller does not diligently monitor and manage switch resources, they can be wasted. Second, there is a deficiency in the proper inspection of southbound interfaces, which are responsible for communication between the controller and switches. Unlike northbound interfaces, southbound interfaces receive limited attention regarding access control and message integrity checking. Third, while OpenFlow has contributed to the success of SDN deployment, its protocol specification delegates many implementation details to vendors, thereby creating potential security vulnerabilities in switch implementations.

5.3. Switch to controller

Rationale behind the penetration route. Network devices, such as routers and switches, can be compromised and exploited to perform additional malicious actions [50,51]. One attractive target would be a controller as it has a direct communication channel with all switches. Switch-to-controller penetration targets the southbound interface, reachable via a control channel (i.e., OpenFlow) established between a malicious switch and controller. The malicious switch injects protocol messages that are parsed within the controller, expecting that it causes an unexpected event chain or abnormal state updates. Considering that OpenFlow specification [44] has lots of message fields and different requirements, the security of the controller mainly relies on how southbound interfaces can handle all corner cases, which is difficult to achieve. Thus, various attack cases are possible if there are bugs on it.

5.3.1. Protocol abusing

A switch-to-controller event chain can be used to inject a malicious payload that is executed inside a controller (see Fig. 7). Xiao et al. [60] reveal that a compromised switch can inject a malicious payload into an OpenFlow message to execute arbitrary commands on SDN apps or controller modules. It exploits the fact that the payload of an OpenFlow message is often used by controller internal components, enabling an attacker to extract confidential information (CVE-2018-1000614, CVE-2018-1000616, CVE-2018-1132, CVE-2018-1000613), manipulate network states (CVE-2018-1999020, CVE-2018-1000615), and perform denial of service (CVE-2018-1999020, CVE-2018-1000617).

On the other hand, a malicious switch may inject abnormal protocol behavior. Specifically, ATTAIN [62] discovers that dropping OpenFlow messages can cause denial-of-service on a target network, as a controller cannot install any flow rule. BEADS [61] reveals that dropping, replaying, and delaying OpenFlow messages can make a controller lose connection from switches.

Finally, a malicious switch may violate message specification to trigger an unexpected controller bug. Shalimov et al. [63] 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 [58] shows that manipulating OpenFlow headers with a randomized value causes a target controller to disconnect the connection from a switch. BEADS [61] presents many similar attack cases in several controllers by fully randomizing all possible OpenFlow headers and message fields using fuzz-testing.

Summary. Switch-to-controller penetration is an emerging research field, but its impact on controller operations cannot be ignored. A compromised switch can directly access a controller through a control channel, which is often overlooked during controller design. Considering this threat model is crucial to enhance SDN security.

Is compromising SDN switches a serious attack? Compromising SDN switches can cause more serious damage compared to legacy network environments. This is because the transmission of corrupted information or messages from compromised devices to an SDN controller can lead to confusion and incorrect decisions 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.

5.4. Host to controller

Rationale behind the penetration route. While the majority of critical communication in an SDN architecture takes place between a controller and switches, it is worth noting that a host can indirectly impact the operation of the controller through a switch. This exploits the fact that most messages sent from a host to a switch are not rigorously investigated and even they can update the controller states

remotely. SDN switches are inherently programmed and send a specific message (e.g., PACKET_IN) to a controller when a certain condition is met (e.g., no matching rules). Thus, a malicious host can send a packet that contains a malicious payload, which is crafted for updating controller states abnormally. This creates an additional avenue for potential penetration.

5.4.1. Denial of service

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, a malicious host can mount controller denial of service (DoS) 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 reflective DoS 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 (Fig. 8). FloodDefender [66] shows that the PACKET_IN flooding attacks overload the CPU utilization of a target controller. SWGuard [64] further employs a probing method that observes round-trip-times (RTTs) to learn which match fields trigger PACKET_IN messages. Besides, many other works [22,23,65,67] are motivated by the DoS attacks due to their 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 [68] analyzes this problem from a performance point of view by evaluating it on hardware OpenFlow switches.

By exploiting implementation bugs, an attacker can deliberately raise a harmful race condition to make a controller unavailable. Xu et al. [54] demonstrate that it is possible to perform TOCTOU (Time-Of-Check to Time-Of-Use) attacks on shared variables of a controller. For instance, SWITCH_JOIN and SWITCH_LEAVE represent events that are generated 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 that 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.

5.4.2. Storage poisoning

From a controller's perspective, maintaining a consistent view with switches is crucial due to their physical separation. To achieve this, most SDN controllers typically store the current *topology view* of the data plane, encompassing information such as link status and host details. To ensure the integrity of this storage, applications running on a controller must carefully consider the view before making any decisions. However, researchers have demonstrated that compromising the topology view is possible by exploiting the *link discovery service* and *host tracking service* of a controller (Section 2.2).

The link discovery service is employed to identify active links on the data plane using the following methods. Initially, it directs a switch to broadcast LLDP (Link Layer Discovery Protocol) packets to its neighbors through PACKET_OUT messages. Upon receiving the LLDP packet, the neighbor switch sends a PACKET_IN message to the controller. Subsequently, 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

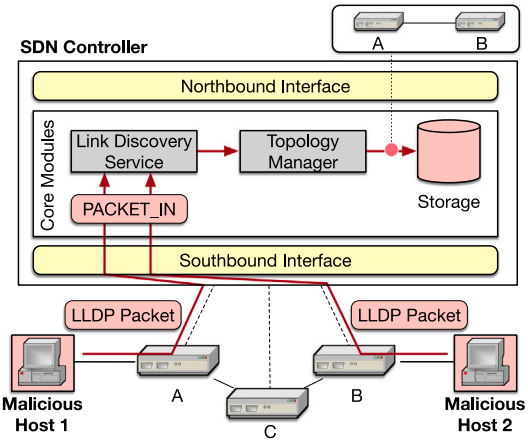


Fig. 9. An illustration of the link fabrication attack [52,67,69] in host-to-controller penetration. The malicious host 1 and 2 establishes a secret tunnel and send shared LLDP packets to the switch A and B, respectively. As a result, the controller misleads that switch A and B are connected.

come from a real switch. This vulnerability allows attackers to manipulate link information from various layers. On the infrastructure layer, SPHNIX [67] and TopoGuard [52] show that an attacker can inject fake link information by relaying LLDP packets between two malicious hosts (Fig. 9). Note that those vulnerabilities were reported to vendors as well (CVE-2015-1610, CVE-2015-1611, CVE-2015-1612). While TopoGuard [52] proposes a defense that distinguishes actual link events based on precondition (e.g., PORT_DOWN or PORT_UP), TopoGuard+ [69] proposes a *port amnesia attack* that bypasses the defense through artificially generating fake PORT_DOWN events.

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 the binding is valid, it gives an attacker a chance to disguise herself with the existing host information.

TopoGuard [52] and SPHNIX [67] 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+ [69] presents a port probing attack that periodically probes victim status and attempts to take the victim's binding when it goes offline. SecureBinder [70] introduces a similar attack, called a *persona hijacking attack* against DHCP.

Finally, it is shown that LLDP packets are attactive targets to trigger exceptional cases on southbound interfaces. Marin et al. [71] propose reverse loop and topology freezing attacks. The former exploits the fact that a controller typically probes an opposite link when receiving an LLDP packet whose LINK_TYPE field is 0×01. 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 originating from the same port. As the link discovery service in Floodlight [40] considers it 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.

5.4.3. Policy evasion

A malicious host can affect controller operation to evade security policies. For instance, the host can abuse a host-to-controller event chain. Ujcich et al. [72] present a cross-plane attack that focuses on "unhandled" events by applications. They show that the malicious

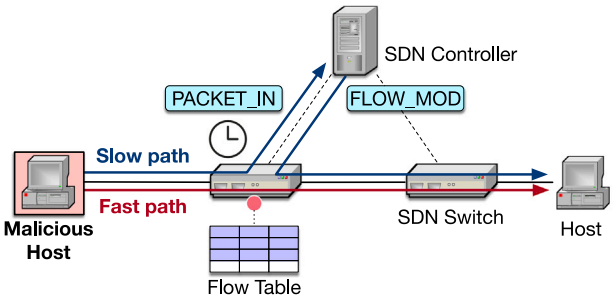


Fig. 10. An illustration of the policy fingerprinting attack [55,56,74,75] in host-to-switch penetration. The malicious host sends a carefully created packet to see if it takes the slow path. If so, the rule associated with the packet is not installed in the switch. Otherwise, the rule is installed in the switch (i.e., fast path).

host can trigger a malformed HOST_UPDATED event containing a broadcast IP address (e.g., 192.168.0.255), which is syntactically valid but meaningless for identifying a host. Subsequently, it makes a controller fail to install a rule that blocks a certain flow as it cannot parse the event. Thus, the attacker can bypass a security policy (CVE-2018-12691). For a similar purpose, the malicious host can send a fake ARP reply that spoofs the other host's MAC address, triggering a HOST_MOVED event. Consequently, the host-related rules are automatically cleaned up, and thus the malicious host can bypass access control rules (CVE-2019-11189). Other similar vulnerabilities were reported as well (i.e., CVE-2019-16297, CVE-2019-16298, CVE-2019-16299, CVE-2019-16300, CVE-2019-16301, CVE-2019-16302).

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

Why is host-to-controller penetration possible? In SDN, a controller possesses a comprehensive overview of the entire network through its logically centralized architecture. This design philosophy offers numerous advantages in effectively and flexibly managing the underlying network devices. However, with switches being reduced to "dumb" devices, there is a vulnerability where a malicious host can inject malformed messages that are not filtered at the data plane. Consequently, switches end up transmitting abnormal messages to the controller. Therefore, the centralized design of SDN can be seen as a double-edged sword. Thus, it is crucial to develop SDN controllers with fault tolerance and the capability to verify updated states. These measures are necessary to mitigate these risks and fully capitalize on the benefits of a centralized architecture.

5.5. Host to switch

Rationale behind the penetration route. In an SDN-enabled network, a host can send packets to its connected switch freely unless there is a blocking rule. By sending well-crafted packets to a switch, a malicious host may trigger bugs in switch implementations or obtain sensitive information. More importantly, this is the shortest route (identical to the app-to-controller); thus it is possible to remain stealthy as there are numerical hosts in the network.

5.5.1. Information leakage

The forwarding mechanism of SDN switches can inadvertently leak valuable information to attackers. For instance, Shin and Gu [21] and Bifulco et al. [73] demonstrate that attackers can discern whether a target switch is SDN-enabled by measuring latency differences caused by table mismatches. In the data plane, a switch needs to contact the SDN controller remotely when there is no matching table entry for incoming packets. The switch sends a PACKET_IN message to the controller, and in response, it receives a FLOW_MOD message instructing flow installation. Incoming packets are held in a queue while waiting for the OpenFlow procedure, and they are subsequently forwarded once the switch installs a flow rule. This moment, known as the *slow path*, results in elevated latency for the first packet, affecting end-user experience.

Through in-depth analysis of timing differences, attackers can fingerprint network and security policies for a target network (Fig. 10). Sonchack et al. [55] demonstrate the possibility of measuring round-trip-times (RTTs) from specific destinations using *packet streams*. Elevated RTTs indicate the involvement of the control plane in packet forwarding, signifying that no installed rule is matched. With this insight, attackers can infer various network policies, including host communication patterns, access control lists, and monitoring rules. Liu et al. [75] propose a more formalized method that models switch flow tables as a Markov model, inferring fine-grained rules among complex flow rules. Yu et al. [74] adopt a similar idea, focusing on switch parameters such as flow table size, cache replacement policy, and load. Achleitner et al. [56] propose flow rule reconstruction techniques using carefully crafted probing packets that spoof specific header fields (e.g., MAC and IP addresses) to determine if they are used as match fields in flow rules. If a destination host replies to the probe despite the spoofed header, attackers can deduce that the field is not used. By eliminating answered headers, attackers can identify an unanswered one, which is the target rule's match field.

Even under the SSL/TLS encryption of control channels, it cannot be fully guaranteed that there is no leakage. Cao et al. [76] demonstrate that attackers can analyze patterns of encrypted control traffic with deep learning 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. [77] 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.

As network size expands, the task of establishing dedicated control channels (referred to as *out-of-band*) between a controller and switches becomes challenging due to physical distances and high costs. In such cases, network operators frequently opt for *in-band* channels, where control-plane and data-plane traffic coexist on the same links. Although the information indicating which links serve as in-band control channels is kept confidential, Cao et al. [78] show that an attacker can identify these links by measuring latency variations.

5.5.2. Main-in-the-middle

Unencrypted control channels are highly susceptible to man-in-the-middle attacks. Unfortunately, SSL/TLS encryption for OpenFlow control channels is rarely implemented in real-world scenarios due to the significant performance degradation it incurs [86]. As a result, if attackers manage to compromise a switch or launch ARP spoofing attacks to intercept control traffic, they gain the ability to eavesdrop on all OpenFlow messages. Notably, Yoon et al. [30] and Jero et al. [61] have demonstrated that attackers positioned as a man-in-the-middle between a controller and a switch can manipulate the action field of FLOW_MOD messages to selectively drop traffic, causing disruptions in the flow of benign network traffic.

Compromised switches can also be abused for man-in-the-middle attacks to bypass middle-box service chains. Bu et al. [79] show that a compromised switch manipulates a packet tag which is used for marking the service chain context of middleboxes. As there is a lack of packet integrity checks in the current SDN, this attack cannot be mitigated.

5.5.3. Denial of service

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 installing unnecessarily many rules. For instance, a malicious host can send randomly spoofed packets to a switch. This attack, also known as *flow table overloading attack*, makes the target switch forward PACKET_INs to a controller, thereby making many flow rules installed [21,64,67,68,80].

Open vSwitch (OVS) [4] is the most popular NFV (Network Function Virtualization), which enables the deployment of high-performance virtual switches. As it is fully compatible with OpenFlow, it is widely used in cloud environments that employ SDN. Whereas the virtualized data plane contributes to broadening OpenFlow deployment, it also expands the attack surfaces of an attacker who looks for a chance to infiltrate inside a cloud. Thimmaraju et al. [51] 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.

Concurrency bugs that trigger race conditions can lead to switch denial-of-service vulnerabilities [81–83]. Consider a scenario where a controller needs to establish bidirectional flow rules using two FLOW_MOD messages before forwarding a requested packet using PACKET_OUT. If BARRIER_REQUEST is not utilized, the OpenFlow messages sent to a specific switch may be processed in a non-deterministic manner. For instance, if a PACKET_OUT message is sent to the switch first, a pending packet may be forwarded before the installation of a flow rule for the opposite path. This can lead to the creation of a forwarding loop and potentially disrupt network operations.

Summary. As discussed, the insufficient implementation of switches is a primary factor that significantly contributes to host-to-switch penetration. This limitation can result in various outcomes, including fingerprinting, man-in-the-middle attacks, and denial of service incidents.

Why is host-to-switch penetration possible? 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.

6. SDN defense classification

In this section, we introduce a classification of countermeasures designed to defend against the aforementioned SDN attacks. Table 4 offers a comprehensive overview of all examined countermeasures, including their associated root causes and components. Furthermore, we include information on the *cost* of each defense, addressing challenges related to their implementation and deployment. Additional insights and findings will be provided at the conclusion of each subsection.

6.1. Application layer

The primary reason that allows a malicious application to perform harmful API invocation is that most controllers have no authentication and authorization measures for applications. Due to this reason, some studies have proposed solutions that can be adopted in the SDN application layer.

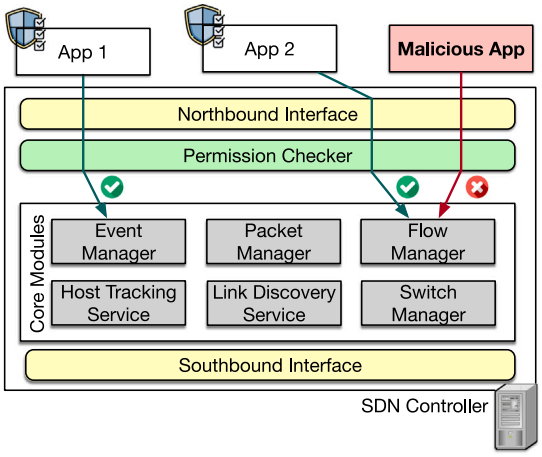


Fig. 11. An illustration of the *permission system* for application-layer defense. The permission checker sits between the northbound interface and core modules to inspect whether applications have suitable permission.

6.1.1. Authentication

A digital certificate signed by trusted entities would help operators trust SDN applications. FortNOX [59] and SE-Floodlight [17] use digital signatures when tracking flow rules driven by specific apps. Rosemary citeshin2014rosemary employs public key infrastructure (PKI) to verify if SDN apps are correctly signed by a developer. These 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 significantly enhance a trusted SDN application ecosystem.

6.1.2. Access control

To prevent unauthorized access to any API of a controller, role-based access control (RBAC) has been proposed [15,17,59,87]. The primary goal is to restrict application behavior by assigning predefined priorities to running SDN applications (see Fig. 11). In this context, SDN applications can be associated with a specific *role*, representing a security level that determines access to certain APIs. Consequently, applications with lower priority cannot invoke security-sensitive northbound APIs, such as `flowruleWrite()` in the ONOS controller, which is used for modifying flow rules.

As modern SDN controllers have grown in complexity, and a multitude of applications have been developed, the RBAC model may prove too broad to handle all cases. For instance, an administrator might want to block `flowruleWrite()` while allowing other functions. To address this, Security-Mode ONOS (SM-ONOS) [15] proposes a northbound interface (NBI) permission model tailored for the ONOS SDN controller [42]. This model introduces a hierarchical structure comprising application-, bundle-, and API-level permissions.

SDNShield [16] presents permission filters similar to Berkeley Packet Filter (BPF) syntax. Administrators can select desired permissions based on a manifest file that includes the permissions used in SDN applications. However, it relies on developers accurately documenting the used permissions in the manifest file. At a certain point, attackers may attempt to specify false permissions to deploy malicious applications. To address this, AEGIS [48] proposes a natural-language-processing (NLP)-based analysis system to compare the actually used permissions of an SDN application with its manifest file.

Finally, some studies have devised a permission model for southbound interfaces (SBI). For instance, SE-Floodlight [17] proposes RBAC for southbound interfaces to restrict the abuse of the OpenFlow protocol.

Table 4
Systematization of SDN defenses. (✓ Root Cause ● Target)

Layer	Type	Defense	Root cause									Target component/interface							Cost
			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	
Application Layer (Section 6.1)	Authentication	Application Authentication [17,18,59]					✓					●							Medium
	Access Control	Role-based Authorization [15,17,59,87]	✓				✓					●	●						Medium
		NBI Permission Model [15,16,48]	✓										●						Medium
		SBI Permission Model [17]		✓								●			●				Medium
	Monitoring	Control-Plane Invariant Verification [17,88–90]									✓	●							Medium
	Program Analysis	Control/Data Flow Analysis [12,45,72,91]									✓	●							Medium
Control Layer (Section 6.2)	Testing	Control Event Blackbox Fuzzing [58]			✓						✓		●						Medium
		Control Message Blackbox Fuzzing [58,61,62]				✓					✓				●				Medium
	Patch/Extension	Multi Controller [39,92,93]							✓					●					High
		Controller Failure Recovery [94,95]							✓					●					Medium
		Controller Sand-boxing [18,87,95]							✓					●					High
Infrastructure Layer (Section 6.3)	Testing	Protocol Conformance Testing [58,96]									✓					●			Medium
	Monitoring	Malicious Switch Detection [97–99]						✓	✓							●			Medium
	Patch/Extension	Protocol Extension [23,68]							✓					●					Medium
		Proactive Rule Installation [65,68]							✓					●	●				Low
		Switch Module Extension [22,23,68,69,100]							✓					●	●				High
		Delay Normalization [55]								✓					●				Low
		Delay Randomization [73,75,78]								✓					●				Low
Cross Layer (Section 6.4)	Authentication	Switch/Host Authentication [52,70,101]						✓							●	●		●	Medium
	Monitoring	Topology Event Verification [52,67,69]			✓	✓									●	●		●	Medium
		Data-Plane Invariant Verification [67,102,103]									✓			●	●	●			Medium
	Program Analysis	Dynamic Instrumentation [54,81–83,104,105]									✓	●		●					Medium
		Provenance Graph Analysis [12,45,106,107]									✓	●		●		●		●	Medium

6.1.3. Monitoring

In SDN architecture, various SDN applications run upon a controller and they can generate many OpenFlow rules. In this context, it is difficult to verify if the rules generated by distinct applications comply with security policies correctly. Thus, the goal of *invariant verification* is to inspect potential violations of those rules.

SE-Floodlight [17] proposes a rule-based conflict analysis (RCA) algorithm that investigates conflicts between a newly created OpenFlow rule and existing ones. For example, a malicious application 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 create a possible rule chain and compare it with existing rules to detect policy violations.

Formal methods are helpful when checking the correctness of the rules. FLOVER [90] and VeriCon [88] model SDN applications as first-order logic to check invariant with Satisfiability Modulo Theories (SMT) solvers. While it may take a long time if there are many invariants required to investigate, it can correctly verify possible corner

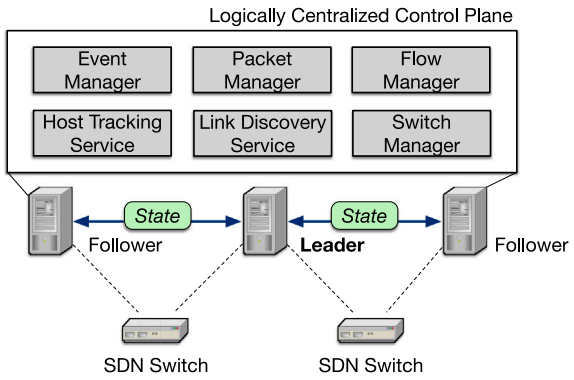


Fig. 12. An illustration of the distributed SDN controllers [39,92,93] for control-layer defense. The *leader* controller replicates states to other *follower* controllers and they synchronize a global view with each other.

cases. NICE [89] 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.

6.1.4. Program analysis

Typical execution flows in an application layer can be represented as a series of API call sequences triggered by an event. 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 problem, operators can employ the *static analysis* that examines the control flow graph (CFG) extracted from an application source code. INDAGO [91] leverages a machine-learning approach to find suspicious patterns of API call chains from malicious apps. It investigates diverse features pertaining to security-sensitive APIs that manipulate the states of SDN controllers. By conducting clustering analysis, it is shown that malicious API chains can be detected with high accuracy. EventScope [72] 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.

Summary. The focus of defense strategies in the application layer 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 techniques. For example, the idea of API-level permissions is borrowed from Android systems [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 a popular SDN controller (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.

6.2. Control layer

As mentioned, controllers are a critical component in SDN, and they must not be vulnerable to shutdown by any attacks. To achieve this, numerous studies have been proposed to test, extend, and improve their robustness and availability.

6.2.1. Testing

Testing, such as *blackbox fuzzing*, has been proven to be useful for finding hidden bugs in software. As these bugs can be abused by attack chains, it is helpful to find and fix them before deployment. DELTA [58] proposes a control event fuzzing module that discovers potential vulnerabilities of northbound interfaces. They are mainly raised from the misimplementation of event processing logic in controller internals. By randomizing inputs or control flow sequences in an application service chain, it is able to find possible bugs pertaining to control events, which can be abused by a malicious application.

Protocol implementation of a controller normally involves large input space due to a variety of protocol messages, and thus it is difficult to find exceptional cases with manual labor. Fuzz-testing can address this challenge by exploring all possible input combinations to find unexpected behavior. DELTA [58], BEADS [61] and ATTAIN [62] 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 specifications. These kinds of black box fuzzing techniques, which can be utilized as security assessment tools at the same time, can classify inherent weaknesses of the target SDN controllers while incredibly reducing manual effort; thus, it would be efficient for operators to measure their own controller's security.

6.2.2. Patch/extension

To mitigate 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 DoS attacks, and even high performance by partitioning a network into several segments. Onix [39] 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 to avoid excessive memory usage per controller instance.

ONOS [92] 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 [93] 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. Note that 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 (see Fig. 12).

As controller instances are augmented, state replication is employed so that slave-controller instances can maintain a consistent view with the master through the east/westbound interface (Section 2.1). Ravana [94] models the process of state replication as a database transaction, and proposes 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 [95] introduces a *cross-layer roll-back* mechanism through managing snapshots. When an application fails, it enables the controller to restore previous states from a saved checkpoint.

The monolithic architecture constitutes a primary factor contributing to the inadequate robustness of SDN controllers. In response to this limitation, researchers have pursued strategies to compartmentalize the execution space, focusing on both applications and core modules. Pioneering initiatives by Rosemary [18] and LegoSDN [95] introduce a

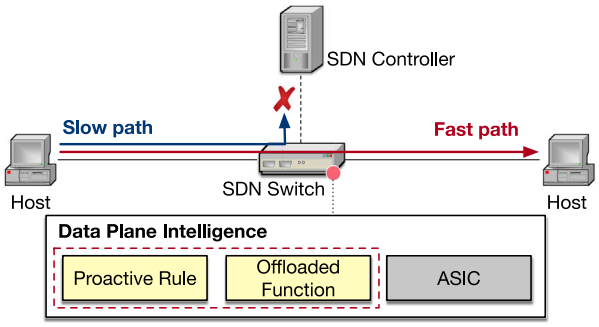


Fig. 13. An illustration of the switch module extension [22,23,68,69,100] for infrastructure-layer defense. The controller offloads performance-sensitive functions and installs proactive rules in the switch. This approach enables flows to take the *fast path*, reducing the controller overhead.

sandbox architecture designed to segregate applications from core modules. This approach effectively detaches application processes from the core module space, facilitating API invocation through Remote Procedure Call (RPC). Furthermore, the work by Rosemary and Barista [87] present a sophisticated *micro-kernel* architecture. This design partitions core modules into isolated entities, thereby enhancing their separation and independence.

Summary. Most defense mechanisms in the control layer focus on addressing the shortage of resource control in current SDN controllers. Many works have proposed extending the control-plane functions to enhance controller robustness and availability.

Why is the control-layer defense important? An SDN controller is often referred to as the "brain" of the network [54]. 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].

6.3. Infrastructure layer

Due to the potential for critical bugs and the risk of malicious behavior in switches, various solutions have been proposed to assess their implementation, monitor behavior, and extend functionalities.

6.3.1. 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 specifications. The root cause of this security hole is that various switch vendors interpret specifications differently. With this fact, there have been several studies to find those implementation holes using protocol implementation testing. SOFT [96] attempts to find implementation inconsistencies among different switch vendors. It utilizes symbolic execution to explore the control flows of switch agents and compares their different outcomes. DELTA [58] uses its value-fuzzer module to inject randomized OpenFlow packets to find abnormal cases from switches.

6.3.2. Monitoring

As discussed, SDN switches are susceptible to compromise, enabling attackers to intercept and drop packets. To mitigate this risk, various techniques have been proposed to detect malicious switches exhibiting anomalous behavior. Kamisinski et al. [97] identify two types of malicious switches—packet droppers and packet swappers (which forward packets to different ports)—and employ anomaly detection to detect these threats. Chi et al.[98] propose an online detection algorithm that generates an artificial packet from a controller to verify if it follows the intended forwarding path. Mohan et al.[99] leverage node-disjoint control paths, exploiting the inconsistency between two control messages if the source switch is compromised. For a more comprehensive survey on this topic, refer to [112].

6.3.3. Patch/extension

Early SDN developers adhered to the design philosophy of 4D [1], leading to SDN switches as simple forwarding devices with limited local-decision capability. The primary issue arises from the verbosity of SDN switches, which must consult a controller when encountering unknown packets (i.e., table-miss). Several solutions have been proposed to alleviate the controller burden by enabling switches to make certain decisions independently when required.

One approach to alleviate the controller burden is the proactive installation of "wildcard" rules. DIFANE [65] introduces the concept of *authority switches* responsible for determining forwarding actions for network partitions, functioning like default gateways. These switches employ wildcard rules that match partial flow spaces of policies. In case of table-misses, ingress switches can consult the authority switches without querying the controller for instructions. DevoFlow [68] also advocates the aggressive use of wildcard rules for uninteresting flows, such as mice flows, shifting most decision-making to the data plane. These approaches offer the advantage of requiring modifications to only a small part of the switch implementation.

Switch modules can be enhanced to support greater intelligence, making them more resilient against saturation attacks (refer to Fig. 13). AVANT-GUARD [23] introduces the *connection migration* technique, which relays traffic only for established TCP sessions. By confirming a new TCP connection request through a syn cookie before reporting it to the control plane, this approach reduces excessive control-plane dependency. Building on this concept, OFX [100] proposes a versatile framework that enables operators to implement various security functions in an OpenFlow switch using diverse APIs. FloodGuard [22] introduces a switch add-on module, known as *data-cache*, which temporarily stores table-miss packets to prevent control-plane saturation during flooding. It employs multi-queues to manage packets per protocol, assuming that attackers typically use a single protocol in flooding attacks (e.g., TCP, ICMP flooding). SWGuard [69] adopts a similar approach, maintaining queues inside a switch based on OpenFlow message types. These queue-based extensions help schedule packets to alleviate the impact on data-to-control-plane messages.

Several works aim to extend protocol capabilities to incorporate additional functions beyond native OpenFlow actions. DevoFlow [68] introduces two actions: *rule cloning* and *local routing*. The former reduces excessive usage of switch TCAM by generating an exact-matching rule from a given wildcard rule, while the latter allows switches to determine multipaths similar to ECMP or automatically reroute a path upon detecting a failed output port. AVANT-GUARD [23] introduces *actuating triggers*, which extends switch functionality to asynchronously report network status without relying on control-plane operations.

The root cause of timing-based side-channel attacks in SDN is the necessity for switches to request instructions from the controller when encountering an unknown packet, leading to exploitable timing differences. To mitigate these attacks, one potential solution is to obfuscate timing delays. For example, Sonchack et al. [55] 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 incorporating random delays into control channels [73,75,77,78].

Summary. Various studies have been conducted on the topic of infrastructure-layer security. However, a majority of these proposals

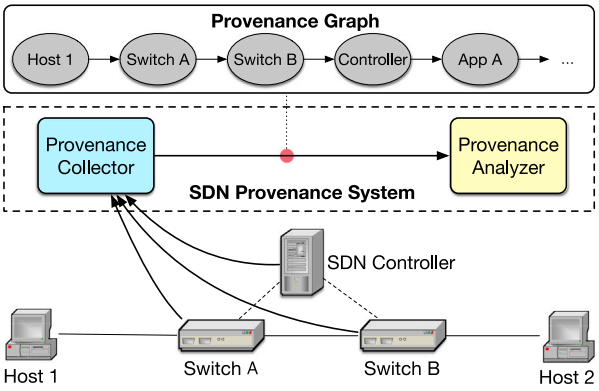


Fig. 14. An illustration of the provenance graph system [12,45,106,107] for cross-layer defense. It collects evidence from multiple SDN layers and analyzes a provenance graph to perform forensic analysis.

concentrate on the verification of enforced rules or the evaluation of protocol correctness. There remains a significant gap in the literature concerning 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.

Is there a trend for the infrastructure-layer defense? In the 2010s, there was an initial proposal to address the dependency on the control plane by extending the functions of SDN switches. This approach aimed to overcome limitations and enhance the capabilities of SDN switches. Consequently, numerous OpenFlow switch extensions were proposed during this period. However, with the emergence of programmable switch architectures like P4, these proposals faced stiff competition and gradually lost prominence. The programmable switch architecture offered more flexibility and programmability, making it a preferred choice over the extension of SDN switch functions.

6.4. Cross layer

As most SDN attacks exploit several vulnerabilities of SDN components, it is necessary to design defenses across SDN layers. This section explores *cross-layer* defenses that require the deployment of multiple countermeasures in SDN architecture.

6.4.1. Monitoring

Current SDN architecture lacks authentication measures for data-plane entities (e.g., switches, hosts), which is opposite to the original design philosophy of SDN [5]. Researchers have proposed solutions that ensure proper authentication of switches and hosts.

TopoGuard [52] presents a link-event authentication that writes a switch signature into an LLDP packet. If another switch checks the signature, it is possible to ensure that the LLDP packet is sent from a trustworthy switch. SecureBinder [70] extends IEEE 802.1x protocol that validates host MAC addresses with certificates signed by CA (Certification Authority). This effectively prevents attackers from spoofing other host identifiers. DFI [101] proposes a more fine-grained access control system that authenticates an end-host with identifiers such as user- and host-names. With the centralized view of an SDN controller, it enforces stateful ACL (Access Control List) rules according to the status of a target host.

TopoGuard [52] proposes a system that filters unacceptable topology events considering the state of switch ports. If a switch receives an LLDP packet from a certain port where PORT_UP event[3] was reported before, the controller can ignore the event because the host is not allowed to generate a topology event (i.e., *link fabrication attack*). To defend the *host identifier spoofing* attack, TopoGuard periodically sends a probe packet to a location where a PORT_DOWN event was detected. This prevents a victim identifier from being hijacked by an attacker during host migration. TopoGuard+ [69] complements the limitation 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 [67] proposes a general framework that intercepts all OpenFlow messages and builds a flow graph that 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.

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 prior studies use *control-message hooking* techniques that capture control messages to check if they correspond to intended network policies. VeriFlow [102] 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 by looking into the required parts of an address space, and pinpoints the violated one. The flow graph proposed by SPHINX [67] can be used to verify network invariants. 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. [103] propose a system that compares control events generated from apps and control messages applied to the data plane. This prevents a malicious application (e.g., rootkit) from installing a false flow rule that violates an operator's network policies.

6.4.2. Program analysis

Application-level race conditions are mainly related to unexpected bugs typically triggered in a dynamic environment, so it is hard to find them from simple unit testing during a development phase. Thus, finding hidden bugs requires a more advanced approach that considers complicated interactions across multi-layers, 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 to facilitate an operator's debugging process. OFRewind [104] is a traffic-replay tool that dynamically records control and data traffic and reproduces them to find bugs when controller operations fail. STS [105] leverages the delta debugging concept that localizes minimum code snippets that are likely to raise exceptional cases. SDNRacer [81,82], BigBug [83], and ConGuard [54] 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 the causal relations of a complex attack chain (see Fig. 14). 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 that dynamically records how an event is propagated from the data plane to the control plane. ProvSDN [45] aims to locate a root cause of cross-application poisoning attacks by backtracking poisoned data that guides a victim application to make a harmful decision. GitFlow [107] takes inspiration from version control systems, such as Git, to create a versioned provenance graph that tracks the evolution

[3] Most SDN controllers use PORT_UP and PORT_DOWN as host-specific events that indicate whether a host-connected link is up or down.

of flow states. In contrast, PicoSDN [106] 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. Besides, the provenance-based defenses also utilize static analysis for pre-processing API call chains before instrumentation [12,45].

Summary. Many countermeasures in SDN require monitoring or modification of multiple SDN components to capture a suitable context.

Why is the cross-layer defense important? Some attacks involve a long penetration route that is composed of multiple SDN components. To prevent this, it is often necessary to monitor events and messages across layers so that a network operator can obtain a holistic view. This is the reason why many provenance-based forensic systems, which collect evidence from multiple components, have been proposed recently.

7. Use cases

In this section, we explore the practical application of our taxonomies and classifications. We elaborate on how practitioners can leverage our survey to bolster the security of SDN. This includes an in-depth analysis of various use cases and case studies, incorporating real-world SDN attack scenarios from a penetration perspective.

7.1. Utilizing penetration routes for proactive defenses

As with the dynamic evolution observed in contemporary malware behavior, attacks on SDN have become increasingly sophisticated, marked by the deployment of diverse techniques and the exploitation of various layers and components within the network. This complexity poses a significant challenge in accurately identifying the root cause and pinpointing vulnerable components during an attack, creating operational difficulties for network operators.

To tackle these challenges, operators can employ *proactive defense* mechanisms, informed by our comprehensive research and analysis. Our survey contributes to this domain by shedding light on recent trends in SDN attacks, notably emphasizing the prevalence of a *bottom-up* penetration route (e.g., host-to-controller). Given these insights, it is crucial for operators to meticulously scrutinize potential implementation flaws, particularly those related to host events on a switch and interactions with the southbound interface of a controller.

7.2. Predicting unexplored penetration routes

Our results provide predictive insights into potential future penetration routes that have remained underexplored. While Section 5 introduces various penetration routes, the *controller-to-controller* route has yet to be investigated in prior research. Section 8.2.3 underscores the prevalence of distributed controllers in large-scale SDN deployments, aimed at enhancing scalability and fault-tolerance. However, the vulnerability inherent in such configurations, particularly the risk of a compromised controller initiating attacks on other controllers, has not been adequately addressed in prior studies.

Moreover, it has been uncovered that the *application-to-host* route remains untapped. We posit that this avenue is viable, given that a malicious application could dispatch a packet to a host through northbound interfaces. For instance, OpenFlow facilitates the PACKET_OUT message, directing a switch to transmit a queued packet [44]. However, by configuring *in_port* to OFPP_CONTROLLER, one can potentially dispatch a CPU-generated packet originating from a controller. Consequently, a malicious controller could transmit a harmful payload, exploiting vulnerabilities in the target host.

7.3. Identifying vulnerable layers and components

Operators can leverage our findings to quantitatively identify vulnerable layers and components within their network. To facilitate this analysis, one can compute the *density*, representing the frequency of each component across all penetration routes. Fig. 15 illustrates the density derived from Table 2, where we tally the occurrences of each component as a source, middle, or target in penetration routes. Notably, the controller emerges as a frequent target due to its pivotal role as the network's core. Concurrently, applications and hosts often serve as the starting points for penetration routes, aligning with their status as primary threat sources. Lastly, switches are commonly exploited as middle nodes, given their close association with the controller.

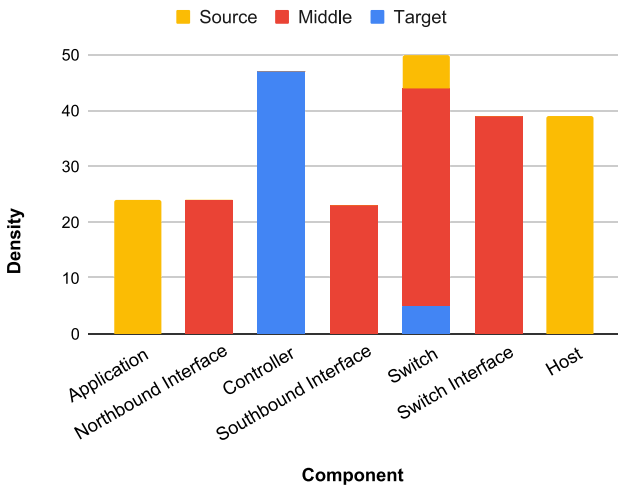


Fig. 15. The density of each component across all penetration routes.

7.4. Correlating penetration routes with shadow CVEs

In response to the escalating concern regarding SDN security, there has been a concerted effort among security researchers to identify vulnerabilities, with a particular focus on communicating their findings to vendors. The Common Vulnerability Exposure (CVE) trend, exemplified in Fig. 16, underscores a discernible rise in the number of identified vulnerabilities within prominent open-source SDN controllers, including ONOS, OpenDaylight, and Floodlight. Recent publications on SDN security meticulously document these vulnerabilities, resulting in the issuance of corresponding CVEs and providing operators with a comprehensive understanding of potential risks. Nevertheless, a significant portion of older papers omit explicit references to CVEs, even when vulnerabilities are reported concurrently, a phenomenon referred to as "shadow CVEs", a term we define to encapsulate instances of unacknowledged vulnerabilities. This oversight has the potential to leave operators with insufficient knowledge. To bridge this gap, we propose aligning presented penetration routes with these shadow CVEs, acknowledging the inherent challenge while emphasizing the invaluable insights such connections offer into the intricacies of security vulnerabilities.

7.5. Real-world case studies

In this section, we present case studies with attack scenarios to illustrate the utilization of surveyed penetration routes and defenses. These cases assume the deployment of ONOS [42], acknowledged as one of the most widely used SDN controllers.

7.5.1. Case study 1 – buffered packet hijacking

The concept of *buffered packet hijacking* [46] illustrates application-to-switch penetration (see Section 5.2), wherein a malicious application illicitly directs a switch to transmit a buffered packet. As previously explained, when a switch receives an unfamiliar packet, it dispatches

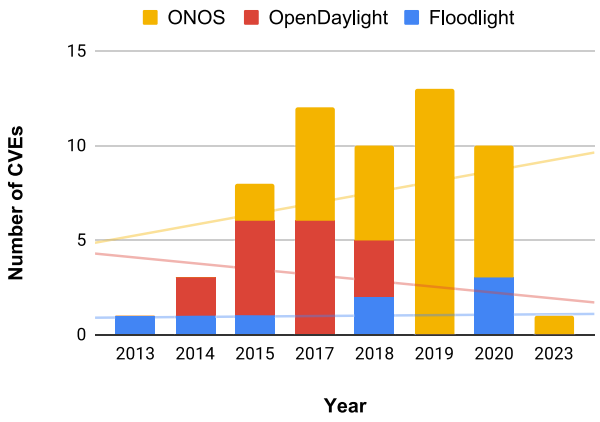


Fig. 16. The number of CVEs reported per year for popular controllers (Note that the lines denote trend lines.). We omit the year where no CVEs are reported (e.g., 2016, 2021, 2022).

a PACKET_IN message to a controller while concurrently buffering the packet (identified by an assigned buffer ID). The essence of this attack involves injecting a FLOW_MOD message that contains a buffer ID identical to the one in the corresponding PACKET_IN message, even without comprehensive knowledge of its complete header. Consequently, the switch forwards the buffered packet, irrespective of the malicious application's lack of responsibility. To execute this attack, the malicious application illegally invokes an interface (e.g., `PacketService`) to retrieve incoming packets. Subsequently, it needs to invoke an interface that generates a rule-installation event (e.g., `FlowRuleService`), spoofing the buffer ID in a matching manner. Finally, the event is transformed into an OpenFlow message (i.e., FLOW_MOD) through a southbound interface (e.g., `OpenFlowRuleProvider`) of the controller, thereby installing a rule on the flow table of the switch.

According to Table 2, the root causes of buffered packet hijacking are (i) lack of NBI (Northbound Interface) authorization, (ii) lack of control event integrity, and (iii) implementation flaws. Additionally, this application-to-switch penetration involves multiple layers. Operators can select suitable defenses for each root cause and layer, as outlined in Table 4. First, operators can employ the NBI permission model, exemplified by Security-Mode ONOS (refer to Section 6.1.2), to prevent unauthorized applications from illicitly accessing the northbound interfaces. Also, the use of control-message blackbox fuzzing tools (e.g., DELTA [58], BEADS [61]) can help pinpoint the code where an event with a spoofed buffer ID is bypassed, even when its header differs from that on the switch. Operators can then apply patches to the code to mitigate this vulnerability. Lastly, data-plane invariant verification, such as VeriFlow [102], proves valuable in scrutinizing policy disparities between the controller and switches. An operator could discover that the rule installed on a switch is not intended by the controller. In summary, by understanding the penetration route, operators can strategically deploy necessary defenses.

7.5.2. Case study 2 – cross-plane attack

The *cross-plane attack* [72] exemplifies host-to-controller penetration, as detailed in Section 5.4.3. In this scenario, a malicious host dispatches a dummy packet with a broadcast IP address, syntactically correct but invalid for normal host identification. This packet triggers the switch to send a PACKET_IN message to the controller. Subsequently, if the controller has not previously discovered this host, ONOS's `HostManager` generates a HOST_UPDATE event. However, this event lacks associated host information due to the invalid IP address, causing another security application (e.g., `acl` in ONOS) to fail in installing a blocking rule. The issue arises as the host is erroneously considered approved by `acl`, allowing subsequent packets to bypass security measures and violating established security policies.

The root causes of this attack include (i) the lack of event integrity checks, (ii) the lack of message integrity checks, and (iii) insufficient host/switch authentication (refer to Table 2). Given these factors and the multi-layered nature of the penetration, an operator could employ two cross-layer defenses outlined in Table 4. First, host authentication mechanisms, such as SecureBinder [61], can aid in identifying only authenticated hosts, preventing malicious hosts from injecting unauthorized packets. Second, topology event verification tools (e.g., TopoGuard [52], TopoGuard+ [69]) can assist in recognizing invalid messages and events, particularly those associated with host information (e.g., broadcast IP addresses).

7.5.3. Case study 3 – flow table overloading

Flow table overloading [21,64,67,68,80] constitutes host-to-switch penetration, wherein a malicious host inundates a switch with a massive number of packets (see Section 5.5.3). The objective is to overwhelm the flow table of the switch by triggering numerous table mismatches, leading to the generation of PACKET_IN and FLOW_MOD messages, and the installation of an excessive number of flow rules on the switch. Given the limited availability of the switch's TCAM, this attack exhausts most of the switch resources, impeding the installation of other essential rules.

While the attack follows a straightforward penetration route between a host and a switch, it also impacts the controller since flow rules are installed by the controller via FLOW_MODs (i.e., lack of controller resource control in Table 2). Consequently, the attack introduces overhead on the controller as well. To mitigate this, the solutions addressing controller resources, such as malicious switch detection [97–99] and proactive rule installation [65,68], can be beneficial. The former enables the detection of abnormal switches that generate numerous PACKET_INs, while the latter involves the proactive installation of a minimal set of rules to prevent table mismatches.

7.5.4. Implication

An in-depth exploration of these case studies highlights the critical importance of understanding penetration routes for enhancing network security within the SDN environment. By carefully mapping out how malicious entities might infiltrate and exploit system vulnerabilities, operators gain invaluable foresight. This strategic insight empowers them to take preemptive measures, deploying a range of defense mechanisms to effectively thwart potential security breaches before they occur. The penetration route serves as a guiding blueprint for operators to deploy targeted, layered defensive measures that align precisely with identified vulnerabilities, optimizing resource allocation and bolstering system resilience. As exemplified in the presented case studies, the practical application of penetration route analysis has played a pivotal role in safeguarding the integrity, confidentiality, and availability of the SDN ecosystem, preempting a wide spectrum of sophisticated attacks from Buffered Packet Hijacking to Flow Table Overloading. This underscores its efficacy as an indispensable component of modern network security strategy.

8. Future research directions

Prior to concluding our remarks, we illuminate the prospective research directions for each SDN layer.

8.1. Application layer

8.1.1. Designing fingerprinting-resistant SDN

As discussed in Section 5.5.1, fingerprinting internal information of SDN poses significant security challenges. Whereas some potential defenses have been discussed in Refs. [73,76,77], none of them yet have proposed a clear solution. Here, we suggest the following two approaches to make SDN robust to fingerprinting attacks.

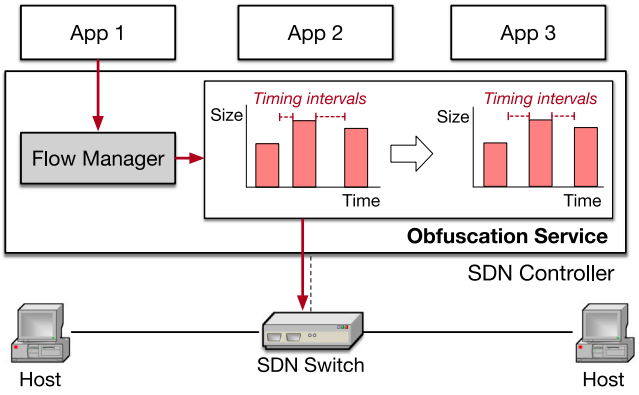


Fig. 17. An illustration of an obfuscation service inside an SDN controller. It obfuscates the timing intervals of control packets for installing flow rules so that an attacker cannot learn traffic patterns.

The first approach aims to incorporate a robust and adaptive *obfuscation service* within the application layer of SDN. Functioning like a proxy, it directs the output messages of various applications as they operate routinely. In this context, the service may employ the insertion of consistent timing intervals on the control channel, effectively preventing attackers from inferring confidential information by learning timing patterns [76,77]. Fig. 17 illustrates an example of this service. However, it is crucial to recognize that adopting this approach entails a trade-off between performance and security. As a result, the obfuscation service may establish a priority hierarchy among applications within the SDN application layer.

The second approach seeks to create SDN applications that possess inherent resilience against fingerprinting attacks. This involves implementing a coordinated and consistent response strategy to conceal any discernible patterns. By standardizing how applications respond to particular network requests or anomalies, it becomes more challenging for attackers to deduce specific application behaviors or detect the presence of certain policies [55,56,73–75]. Specifically, this approach may involve the incorporation of *dummy* messages exchanged between a controller and switches. However, it is evident that performance overhead is inevitable; hence, appropriate optimization techniques should be introduced.

In conclusion, the proposed strategies aim to enhance the security posture of SDNs against internal information fingerprinting. By obfuscating traffic patterns and providing fingerprinting-resistant applications, it is possible to significantly diminish the effectiveness of network fingerprinting techniques. Future research should focus on developing and evaluating these approaches in real-world scenarios, assessing their efficacy in protecting against sophisticated network reconnaissance methods.

8.1.2. Solving policy inconsistencies in large-scale SDN

As explored in Section 5.1.2, the absence of state synchronization between a controller and switches gives rise to *policy inconsistencies*. While ad-hoc remedies have been suggested for small-scale SDN deployments, the burgeoning expansion of the networking landscape introduces challenges in identifying policy inconsistencies in practical scenarios. This challenge becomes especially pronounced for graph-based detection mechanisms; as network topologies expand, the intricacy of detection escalates, potentially making it impracticable.

In order to enhance the identification of inconsistencies within extensive SDN environments, the utilization of *tagging* emerges as a viable solution. Inconsistencies are discerned through a comparative analysis of tags, which are updated as the packet follows the anticipated path in the control plane, against those updated subsequent to the authentic movement of the packet. Prior investigations, as outlined in a recent study [113], have evidenced the minimal overhead incurred by this methodology within experimental settings. Consequently, the logical progression involves the application of this approach to large-scale SDN within practical environments. Fig. 18 illustrates exemplifies the tag-based policy verification system in AS-level network topology.

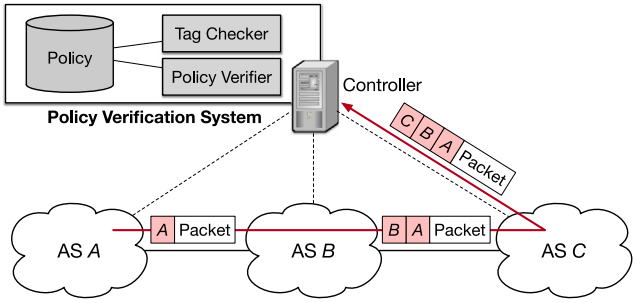


Fig. 18. An illustration of a policy verification system in AS-level network topology. It checks whether the packet is correctly sent from AS A to AS C by inspecting its tag.

As such, detecting policy inconsistencies is crucial, but proactively preventing them is even more important. Inconsistencies may arise due to issues related to the trustworthiness and standardization of network applications. For instance, a vendor might offer a network application that conflicts with other applications provided by different vendors, potentially leading to policy conflicts and resultant inconsistencies. Additionally, the open programmability of these systems allows for rule modifications, further increasing the risk of inconsistency. To mitigate these issues, more stringent *policy management APIs* should be established and rigorously validated.

8.2. Control layer

8.2.1. Implementing zero trust paradigm in SDN

Within the framework of a zero-trust model, the prevailing security protocols at the application layer bear similarities to a traditional perimeter-based defense system. For instance, the current SDN permission model, as illustrated by Security-Mode ONOS [15], primarily evaluates whether a specific application holds the essential permissions at a particular layer. If the application indeed possesses the requisite permissions, the controller places trust in all ensuing actions of the application. Furthermore, messages emanating from a switch are deemed trustworthy once it establishes a connection with a controller. This mirrors the methodology employed by a perimeter-based defense system.

Thus, in forthcoming research, it is imperative to integrate the zero-trust paradigm into SDN, scrutinizing all layers for events without relying on inherent trust. Architecturally, the SDN controller, with its comprehensive oversight of application and network events, emerges as an optimal locus for such implementation. A notable challenge lies in effectively monitoring and verifying the multitude of events originating from expansive SDN environments, particularly in large-scale settings like WANs and data centers. Consequently, the forthcoming research focus should pivot towards designing a monitoring system adept at capturing all pertinent events while ensuring optimal performance in the face of the inherent scalability of practical SDN environments.

8.2.2. Finding vulnerabilities in O-RAN architecture

A current trend in the application of SDN involves Open Radio Access Network (O-RAN) [114], an emerging architectural framework designed to enable the programmable management of cellular networks. O-RAN has incorporated various software technologies, including SDN and containers, to enhance flexibility, garnering significant attention in recent times. Nevertheless, akin to the challenges posed by initial SDN architectures, O-RAN introduces potential vulnerabilities that demand careful examination. Therefore, it is imperative to assess potential threats and formulate appropriate security measures.

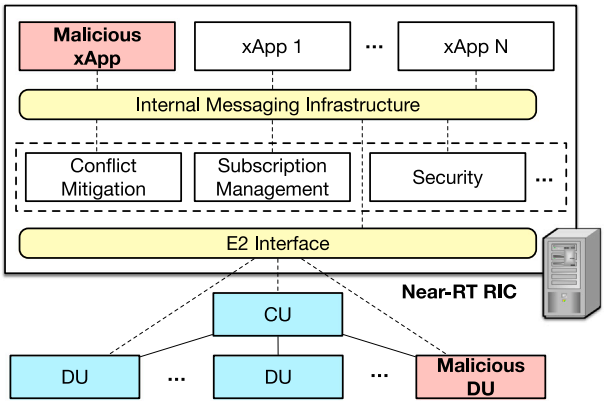


Fig. 19. An illustration of potential threats in the Near-RT RIC; a malicious xAPP can execute harmful actions, or a malevolent RAN node (e.g., a malicious DU) may engage in detrimental behaviors.

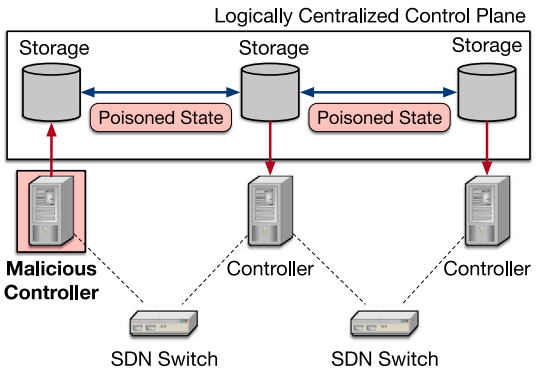


Fig. 20. An illustration of a potential threat in distributed SDN controllers. A poisoned state injected by the malicious controller can be propagated to other controllers without validation.

Within the O-RAN framework, the Near-Real Time RAN Intelligent Controller (Near-RT RIC) operates as a logically centralized entity, resembling an SDN controller, with oversight of both Control Units (CUs) and Distributed Units (DUs). Connectivity between RAN nodes and the Near-RT RIC takes place through an *E2 interface*, mirroring the southbound interface in SDN. It is essential to highlight the adaptability of the E2 interface to various RAN node vendors, introducing increased complexity during implementation. Due to its recent integration into the control layer, there is a potential for an unforeseen penetration route, such as a bottom-up penetration route by a malicious DU, as illustrated in Fig. 19, resembling the switch-to-controller penetration in SDN. Furthermore, the Near-RT RIC hosts multiple xApps akin to SDN applications, where a malicious xApp can potentially access the E2 interface through the internal messaging infrastructure. Significantly, vulnerabilities in the Near-RT RIC are of utmost concern, given its mandated time scale of at most $\leqslant$ 1000 ms. This underscores the urgency and significance of addressing security concerns to ensure the robustness and reliability of the O-RAN framework.

To mitigate the risk of unforeseen vulnerabilities, the use of *fuzzing* is recommended. Specifically, the E2 interface comprises two protocols: E2AP (E2 Application Protocol) and E2SM (E2 Service Model). Although standards for both protocols have been established by the O-RAN Alliance [115,116], variations in implementation are possible across different vendors, potentially leading to flaws. As a preventive measure, one can utilize generation-based fuzzing, generating inputs based on the protocol specifications to systematically assess and uncover potential vulnerabilities.

8.2.3. Finding vulnerabilities in distributed controllers

With the significant growth of modern networking environments, it has become apparent that relying on a single SDN controller is insufficient to efficiently orchestrate the enormous underlying traffic. As a solution, distributed SDN architectures have emerged, aiming to alleviate the burdensome overload on a single centralized controller and ensure its resiliency through a fault-tolerant system. However, while vulnerabilities within a single controller environment have been extensively studied, the security implications of distributed SDN architectures have not received thorough investigation. Therefore, it is crucial to delve into the security issues that may arise from the distributed SDN architecture, in order to identify and address potential vulnerabilities and strengthen the overall security of such environments.

Furthermore, modern distributed SDN architectures have undergone constant evolution. As an example, the ONOS project [42] recently separated its underlying distributed storage into an independent project named Atomix [117]. This separation allows operators to design clusters that incorporate controllers and storage with customizable configurations, providing flexibility for various use cases that require distributed systems. However, it is important to recognize that this newly emerged structure may introduce new attack surfaces, presenting potential vulnerabilities for attackers to exploit. Therefore, careful consideration and analysis of the security implications are necessary to ensure the robustness and resilience of these distributed SDN architectures.

An example of an unresolved issue in SDN is the presence of a malicious application that can potentially impact control-layer operations, even when employing multi-controller or distributed controller setups. This issue arises from the prevailing design of distributed controllers, which follows a physically distributed but logically centralized architecture. In this architecture, states made by a single controller are replicated across all controllers, creating a vulnerability where the entire system can be compromised if a malicious controller successfully manipulates these states (see Fig. 20). Therefore, there is a need to address this security concern and explore alternative architectural designs or mechanisms to mitigate the risk of such malicious control-layer operations.

8.2.4. Microservices and SDN security

Distributed deployments of SDN controllers have enhanced scalability but involve unnecessary replication of the entire SDN controller functionality. To tackle this, a newer approach, the microservice-based SDN controller [118,119], has emerged. Leveraging microservices enables the construction of a system composed of small, diverse components. This architectural approach facilitates the separation of SDN functions and allows for on-demand scaling of resource-intensive tasks without necessitating full-scale replication.

However, the partitioning of internal SDN functions into microservices introduces a series of new security considerations, due to the architectural characteristics of microservices. These considerations extend to concerns related to inter-service communication, necessitating the implementation of robust authentication and authorization mechanisms to manage access between microservices, which are SDN functions partitioned into smaller and more fine-grained components. Also, the security of APIs that facilitate communication between such microservices becomes of paramount importance, as any vulnerabilities in these interfaces could potentially be exploited by malicious entities.

Additionally, akin to the approach employed by distributed SDN controllers, which have integrated distributed storage solutions such as Atomix to uphold group management and data consistency, replicating these capabilities within microservices becomes a critical imperative, especially when microservices require concurrent access to the same networking resources. Furthermore, in scenarios where multiple microservices actively engage in transactions or data updates, ensuring the preservation of transactional consistency takes center stage.

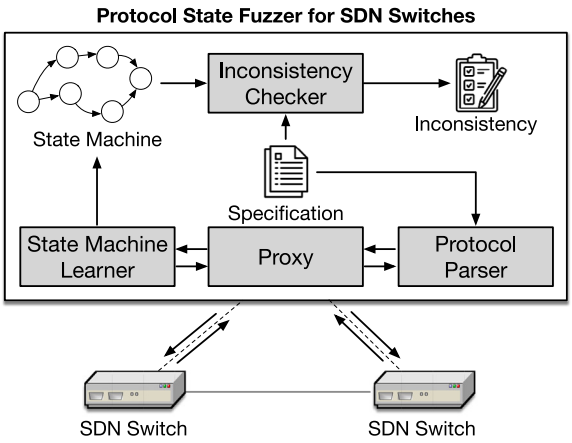


Fig. 21. An illustration of a protocol state fuzzer for SDN switches.

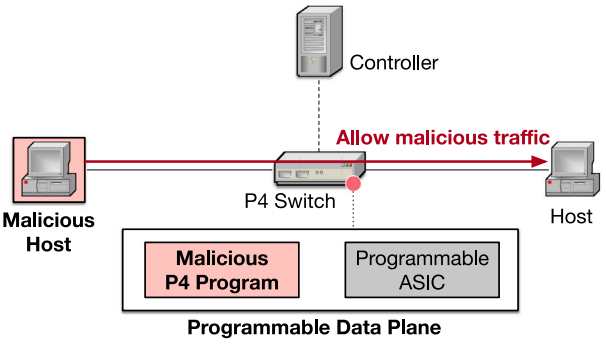


Fig. 22. An illustration of a potential threat in the programmable data plane (e.g., P4 switches). A buggy program may allow malicious traffic to bypass access control.

This necessitates the implementation of robust distributed transaction mechanisms, particularly within the dynamic and scaling container environment inherent to microservices.

8.3. Infrastructure layer

8.3.1. Verifying implementation inconsistency in SDN switches

OpenFlow serves as the de facto standard protocol facilitating communication between controllers and switches. Since the publication of its official specification (v1.0.0) in 2009, the protocol has evolved significantly, incorporating numerous new features and meeting various requirements, culminating in the latest version (v1.5.0) [44]. However, this evolution has introduced complexities, potentially resulting in inconsistencies between the specification and actual OpenFlow implementations. Notably, we contend that SDN switches might harbor many implementation inconsistencies, as the scrutiny directed at SDN controllers has overshadowed attention to SDN switches. Moreover, the proliferation of switch vendors further amplifies the diversity in switch implementations, contributing to potential challenges in maintaining consistency across the ecosystem.

To tackle this challenge, it is imperative to conduct thorough testing of switch implementations to ensure their adherence to specifications. However, this task is inherently demanding due to the intricacies of the OpenFlow specification, which significantly expands the search space. To address this complexity, we propose the adoption of *protocol state fuzzing* as a testing methodology. This approach identifies undesired states in switch implementations by learning a protocol state machine. It achieves this by systematically sending a series of inputs to a target switch and analyzing the corresponding outputs. Subsequently, the learned state machine is compared against the official specification to identify any disparities. By employing this technique, we can streamline the search space, enhancing the efficiency of pinpointing discrepancies between a switch implementation and the specified standards. In Fig. 21, we propose a design of a protocol state fuzzer for SDN switches, which adopts this technique.

8.3.2. Finding vulnerabilities in programmable switches

Programmable switches have emerged as the next revolutionary paradigm for innovative networking research, extending beyond the scope of SDN. This system empowers network operators to define the behavior of network devices at a granular level, enabling precise control over how each packet is processed. An exemplary technology in this domain is P4 [120], which has garnered support from both academia and industry due to its capacity to customize packet processing logic while maintaining line-rate performance. This unprecedented level of control and customization facilitates the creation of networks tailored to specific requirements, leading to improved efficiency and enhanced overall performance. The programmable data plane holds immense potential for driving further advancements in networking technology and propelling the field into new dimensions of innovation and optimization.

However, despite the increasing adoption of programmable data planes, their security aspect has not received sufficient attention. This lack of focus on security could inadvertently lead to an expanded attack surface, primarily due to the following reasons:

First, the introduction of significant complexity through P4's capability to define every aspect of packet processing could pose a security risk. For example, the Intel Tofino ASIC implements the TNA (Tofino Native Architecture), a custom P4 architecture that supports advanced capabilities necessary for implementing complex software programs [121]. This granular level of control afforded by P4 may potentially lead to the presence of bugs and vulnerabilities, which attackers could exploit to disrupt the network or gain unauthorized access to sensitive data. The fine-grained customization allowed by P4, while empowering network operators, also requires careful attention to ensure robust security measures are in place.

Second, while P4 facilitates the rapid development and deployment of new network functions, this advantage may inadvertently encourage the use of unverified or insecure code, as it lacks robust testing and verification frameworks [122]. As a consequence, the potential presence of unvetted code in the programmable data plane could introduce vulnerabilities that attackers could exploit to compromise network policies or exploit sensitive information (Fig. 22). Thus, ensuring proper testing and verification of code within the programmable data plane becomes crucial to mitigate such security risks and maintain the overall resilience of the network.

Third, P4's philosophy of user-centric control over the data plane implies that security heavily relies on user behavior. Consequently, insecure practices by users can potentially expose the network to various types of attacks. The level of customization and control offered by P4 puts a significant responsibility on network operators to implement secure configurations and adhere to best security practices. Failure to do so could create vulnerabilities and avenues for exploitation by malicious actors. Therefore, it is crucial to raise awareness among users about the security implications of their actions and promote a security-first mindset when designing and operating networks using programmable data planes.

Lastly, the dynamic nature of programmable data planes can pose challenges to conventional network monitoring systems, making real-time detection of attacks particularly difficult [123,124]. The fine-grained customization and constant changes in packet processing logic create complexities for traditional monitoring tools to accurately detect and respond to emerging threats. As a result, attackers may exploit this gap in real-time detection to carry out sophisticated attacks that evade traditional security measures. Addressing this issue requires the development of advanced monitoring and detection mechanisms that

can adapt to the dynamic nature of programmable data planes and effectively identify potential security breaches in real time, thereby ensuring a robust defense against emerging threats.

9. Conclusion

In this paper, we have conducted a comprehensive examination of SDN security, encompassing a review of both attacks and defenses. Our contribution includes the introduction of novel taxonomies, specifically focusing on penetration routes and root causes. We have categorized existing attacks based on penetration routes and defenses according to their respective layers. Through a detailed exploration of motivations, approaches, and fundamental security issues, we have identified critical areas that merit increased attention.

While numerous researchers have dedicated efforts to analyzing potential vulnerabilities and devising practical defenses in the realm of SDN, our conclusion emphasizes the need for heightened consideration of SDN controller security. The secure communication between the control and data planes emerges as a pivotal aspect that requires more in-depth investigation.

Furthermore, recognizing the ongoing trend of "softwarization" extending to the data plane, we advocate for an intensified exploration of security issues in this layer by security researchers. By shedding light on the existing landscape of SDN security and delineating potential future research directions, we aim to catalyze further advancements in safeguarding SDN architectures.

CRediT authorship contribution statement

Jinwoo Kim: Writing – review & editing, Writing – original draft, Investigation, Conceptualization. **Minjae Seo:** Writing – review & editing, Writing – original draft, Investigation, Conceptualization. **Seungsoo Lee:** Writing – review & editing. **Jaehyun Nam:** Writing – review & editing. **Vinod Yegneswaran:** Writing – review & editing. **Phillip Porras:** Writing – review & editing. **Guofei Gu:** Writing – review & editing. **Seungwon Shin:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgment

We express our sincere gratitude to the anonymous reviewers of Elsevier Computer Networks for their invaluable comments and feedback. The present research has been conducted by the Research Grant of Kwangwoon University in 2022. Also, this work was partially supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2022-00166401).

References

[1] A. Greenberg, G. Hjalmtysson, D.A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, H. Zhang, A clean slate 4D approach to network control and management, ACM SIGCOMM Comput. Commun. Rev. (2005).

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: Enabling innovation in campus networks, ACM SIGCOMM Comput. Commun. Rev. (2008).

[3] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, 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, S. Shenker, Ethane: Taking control of the enterprise, ACM SIGCOMM Comput. Commun. Rev. (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 Comput. Commun. Rev. (2013).

[7] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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] D. Kreutz, F.M. Ramos, 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.

[25] S. Scott-Hayward, S. Natarajan, S. Sezer, A survey of security in software defined networks, IEEE Commun. Surv. Tutor. (2015).

[26] I. Ahmad, S. Namal, M. Ylianttila, A. Gurtov, Security in software defined networks: A survey, IEEE Commun. Surv. Tutor. (2015).

[27] I. Alsmadi, D. Xu, Security of software defined networks: A survey, Comput. Secur. 53 (2015) 79–108.

[28] Q. Yan, F.R. Yu, Q. Gong, 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 Commun. Surv. Tutor. 18 (1) (2015) 602–622.

[29] S. Khan, A. Gani, A.W.A. Wahab, M. Guizani, M.K. Khan, Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art, IEEE Commun. Surv. Tutor. 19 (1) (2016) 303–324.

[30] C. Yoon, S. Lee, H. Kang, T. Park, S. Shin, V. Yegneswaran, P. Porras, G. Gu, Flow wars: Systemizing the attack surface and defenses in software-defined networks, IEEE/ACM Trans. Netw. (2017).

[31] A. Shaghaghi, M.A. Kaafar, R. Buyya, S. Jha, Software-Defined Network (SDN) data plane security: Issues, solutions, and future directions, in: Handbook of Computer Networks and Cyber Security: Principles and Paradigms, Springer, 2020, pp. 341–387.

[32] J.C.C. Chica, J.C. Imbachi, J.F.B. Vega, Security in SDN: A comprehensive survey, J. Netw. Comput. Appl. 159 (2020) 102595.

[33] B. Rauf, H. Abbas, M. Usman, T.A. Zia, W. Iqbal, Y. Abbas, H. Afzal, Application threats to exploit northbound interface vulnerabilities in software defined networks, ACM Comput. Surv. 54 (6) (2021) 1–36.

[34] M.B. Jimenez, D. Fernandez, J.E. Rivadeneira, L. Bellido, A. Cardenas, A survey of the main security issues and solutions for the SDN architecture, IEEE Access 9 (2021) 122016–122038.

[35] M. Rahouti, K. Xiong, Y. Xin, S.K. Jagatheesaperumal, M. Ayyash, M. Shaheed, SDN security review: Threat taxonomy, implications, and open challenges, IEEE Access 10 (2022) 45820–45854.

[36] A. Melis, A. Al Sadi, D. Berardi, F. Callegati, M. Prandini, A systematic literature review of offensive and defensive security solutions with software defined network, IEEE Access (2023).

[37] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker, NOX: Towards an operating system for networks, ACM SIGCOMM Comput. Commun. Rev. (2008).

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

[39] 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.

[40] Floodlight controller, 2023, https://github.com/floodlight/floodlight.

[41] D. Erickson, The beacon OpenFlow controller, in: Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, 2013.

[42] ONOS Github repository, 2022, https://github.com/opennetworkinglab/onos.

[43] OpenDaylight Github repository, 2023, https://github.com/opendaylight/.

[44] OpenFlow switch specification v1.5.1, 2014, https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[45] B.E. Ujcich, S. Jero, A. Edmundson, Q. Wang, R. Skowyra, J. Landry, A. Bates, W.H. Sanders, C. Nita-Rotaru, H. Okhravi, Cross-app poisoning in software-defined networking, in: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, ACM, 2018.

[46] J. Cao, R. Xie, K. Sun, Q. Li, G. Gu, M. Xu, When match fields do not need to match: Buffered packet hijacking in SDN, in: Proceedings of the Network and Distributed System Security Symposium, Internet Society, 2020.

[47] Y. Zhou, X. Jiang, Dissecting android malware: Characterization and evolution, in: Proceedings of the IEEE Symposium on Security and Privacy, IEEE, 2012.

[48] H. Kang, S. Shin, V. Yegneswaran, S. Ghosh, 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.

[49] S. Lee, S. Woo, J. Kim, V. Yegneswaran, P. Porras, S. Shin, AudiSDN: Automated detection of network policy inconsistencies in software-defined networks, in: Proceedings of the IEEE Conference on Computer Communications, IEEE, 2020.

[50] L. Felix, Router exploitation, 2009, Black Hat Briefings USA.

[51] K. Thimmaraju, B. Shastry, T. Fiebig, F. Hetzelt, J.-P. Seifert, A. Feldmann, S. Schmid, Taking control of SDN-based cloud systems via the data plane, in: Proceedings of the Symposium on SDN Research, 2018.

[52] S. Hong, L. Xu, H. Wang, 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.

[53] B. Agborubere, 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.

[54] L. Xu, J. Huang, S. Hong, J. Zhang, G. Gu, Attacking the brain: Races in the SDN control plane, in: Proceedings of the USENIX Security Symposium, USENIX, 2017.

[55] J. Sonchack, A. Dubey, A.J. Aviv, J.M. Smith, E. Keller, Timing-based reconnaissance and defense in software-defined networks, in: Proceedings of the Annual Conference on Computer Security Applications, 2016.

[56] S. Achleitner, T. La Porta, T. Jaeger, P. McDaniel, Adversarial network forensics in software defined networking, in: Proceedings of the Symposium on SDN Research, ACM, 2017.

[57] V.H. Dixit, A. Doupé, Y. Shoshitaishvili, Z. Zhao, 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.

[58] S. Lee, C. Yoon, C. Lee, S. Shin, V. Yegneswaran, 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.

[59] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, G. Gu, A security enforcement kernel for OpenFlow networks, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, ACM, 2012.

[60] F. Xiao, J. Zhang, J. Huang, G. Gu, D. Wu, 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.

[61] S. Jero, X. Bu, C. Nita-Rotaru, H. Okhravi, R. Skowyra, 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.

[62] B.E. Ujcich, U. Thakore, 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.

[63] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, R. Smeliansky, Advanced study of SDN/OpenFlow controllers, in: Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia, 2013.

[64] M. Zhang, G. Li, L. Xu, J. Bi, G. Gu, 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.

[65] M. Yu, J. Rexford, M.J. Freedman, J. Wang, Scalable flow-based networking with DIFANE, in: Proceedings of the ACM Special Interest Group on Data Communication, ACM, 2010.

[66] G. Shang, P. Zhe, X. Bin, H. Aiqun, 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.

[67] M. Dhawan, R. Poddar, K. Mahajan, V. Mann, SPHINX: Detecting security attacks in software-defined networks, in: Proceedings of the Network and Distributed System Security Symposium, Internet Society, 2015.

[68] A.R. Curtis, J.C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, DevoFlow: Scaling flow management for high-performance networks, in: Proceedings of Conference of the ACM Special Interest Group on Data Communication, 2011.

[69] R. Skowyra, L. Xu, G. Gu, V. Dedhia, T. Hobson, H. Okhravi, 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.

[70] S. Jero, W. Koch, R. Skowyra, H. Okhravi, C. Nita-Rotaru, D. Bigelow, Identifier binding attacks and defenses in software-defined networks, in: Proceedings of the USENIX Security Symposium, USENIX, 2017.

[71] E. Marin, N. Bucciol, 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.

[72] B.E. Ujcich, S. Jero, R. Skowyra, S.R. Gomez, A. Bates, W.H. Sanders, 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.

[73] R. Bifulco, H. Cui, G.O. Karame, F. Klaedtke, Fingerprinting software-defined networks, in: Proceedings of the International Conference on Network Protocols, IEEE, 2015.

[74] M. Yu, T. He, P. McDaniel, 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.

[75] S. Liu, M.K. Reiter, V. Sekar, Flow reconnaissance via timing attacks on SDN switches, in: Proceedings of the International Conference on Distributed Computing Systems, IEEE, 2017.

[76] J. Cao, Z. Yang, K. Sun, Q. Li, M. Xu, P. Han, Fingerprinting SDN applications via encrypted control traffic, in: Proceedings of the International Symposium on Research in Attacks, Intrusions and Defenses, 2019.

[77] M. Seo, J. Kim, E. Marin, M. You, T. Park, S. Lee, S. Shin, J. Kim, Heimdallr: Fingerprinting SD-WAN control-plane architecture via encrypted control traffic, in: Annual Computer Security Applications Conference, 2022, pp. 949–963.

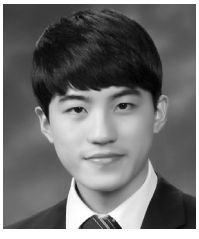
[78] J. Cao, Q. Li, R. Xie, K. Sun, G. Gu, M. Xu, Y. Yang, The CrossPath attack: Disrupting the SDN control channel via shared links, in: Proceedgins of the Security Symposium, USENIX, 2019.

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

[80] T.A. Pascoal, I.E. Fonseca, V. Nigam, Slow denial-of-service attacks on software defined networks, Comput. Netw. 173 (2020) 107223.

[81] J. Miserez, P. Bielik, A. El-Hassany, L. Vanbever, 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.

[82] A. El-Hassany, J. Miserez, P. Bielik, L. Vanbever, M. Vechev, SDNRacer: Concurrency analysis for software-defined networks, in: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, ACM, 2016.
[83] R. May, A. El-Hassany, L. Vanbever, M. Vechev, BigBug: Practical concurrency analysis for SDN, in: Proceedings of the Symposium on SDN Research, ACM, 2017.
[84] ONOS reactive forwarding application, 2023, https://github.com/opennetworkinglab/onos/blob/master/apps/fwd/src/main/java/org/onosproject/fwd/ReactiveForwarding.java.
[85] Unverified commits: Are you unknowingly trusting attackers' code? 2023, https://checkmarx.com/blog/unverified-commits-are-you-unknowingly-trusting-attackers-code/.
[86] R. Durner, W. Kellerer, The cost of security in the SDN control plane, in: Proceedings of the ACM CoNEXT 2015-Student Workshop, ACM, 2015.
[87] J. Nam, H. Jo, Y. Kim, P. Porras, V. Yegneswaran, S. Shin, Barista: An event-centric NOS composition framework for software-defined networks, in: Proceedings of the IEEE Conference on Computer Communications, IEEE, 2018.
[88] T. Ball, N. Bjørner, A. Gember, S. Itzhaky, A. Karbyshev, M. Sagiv, M. Schapira, 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.
[89] M. Canini, D. Venzano, P. Perešíni, D. Kostić, J. Rexford, A NICE way to test openflow applications, in: Proceedings of the USENIX Symposium on Networked Systems Design and Implementation, USENIX, 2012.
[90] S. Son, S. Shin, V. Yegneswaran, P. Porras, G. Gu, Model checking invariant security properties in OpenFlow, in: Proceedings of the IEEE International Conference on Communications, IEEE, 2013.
[91] C. Lee, C. Yoon, S. Shin, S.K. Cha, INDAGO: A new framework for detecting malicious SDN applications, in: Proceedings of the IEEE International Conference on Network Protocols, IEEE, 2018.
[92] 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.
[93] J. Medved, R. Varga, A. Tkacik, 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.
[94] N. Katta, H. Zhang, M. Freedman, J. Rexford, Ravana: Controller fault-tolerance in software-defined networking, in: Proceedings of the ACM SIGCOMM Symposium on Software Defined Networking Research, ACM, 2015.
[95] B. Chandrasekaran, B. Tschaen, T. Benson, Isolating and tolerating SDN application failures with legosdn, in: Proceedings of the Symposium on SDN Research, ACM, 2016.
[96] M. Kuzniar, P. Peresini, M. Canini, D. Venzano, D. Kostic, A SOFT way for openflow switch interoperability testing, in: Proceedings of the International Conference on Emerging Networking Experiments and Technologies, ACM, 2012.
[97] A. Kamisiński, 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.
[98] P.-W. Chi, C.-T. Kuo, J.-W. Guo, 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.
[99] P.M. Mohan, T. Truong-Huu, 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.
[100] J. Sonchack, J.M. Smith, A.J. Aviv, E. Keller, Enabling practical software-defined networking security applications with OFX, in: Proceedings of the Network and Distributed System Security Symposium, vol. 16, 2016.
[101] S.R. Gomez, S. Jero, R. Skowyra, J. Martin, P. Sullivan, D. Bigelow, Z. Ellenbogen, B.C. Ward, H. Okhravi, 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.
[102] A. Khurshid, X. Zou, W. Zhou, M. Caesar, 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.
[103] C. Röpke, 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.
[104] A. Wundsam, D. Levin, S. Seetharaman, A. Feldmann, OFRewind: Enabling record and replay troubleshooting for networks, in: Proceedings of the USENIX Annual Technical Conference, USENIX, 2011.
[105] 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.
[106] B.E. Ujcich, S. Jero, R. Skowyra, A. Bates, W.H. Sanders, H. Okhravi, Causal analysis for software-defined networking attacks, in: USENIX Security Symposium, 2021, pp. 3183–3200.
[107] A. Dwaraki, S. Seetharaman, S. Natarajan, 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.
[108] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L.P. Cox, J. Jung, P. McDaniel, A.N. Sheth, TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones, ACM Trans. Comput. Syst. (2014).
[109] V.B. Livshits, 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, 2023, https://wiki.onosproject.org/display/ONOS/Security-Mode+ONOS.
[111] CORD: Central office re-architected as a datacenter, 2023, https://opennetworking.org/cord/.
[112] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, M. Conti, A survey on the security of stateful SDN data planes, IEEE Commun. Surv. Tutor. 19 (3) (2017) 1701–1725.
[113] K. Lei, G. Lin, M. Zhang, K. Li, Q. Li, X. Jing, P. Wang, Measuring the consistency between data and control plane in SDN, IEEE/ACM Trans. Netw. 31 (2) (2022) 511–525.
[114] ORAN alliance, 2023, https://www.o-ran.org/.
[115] O. Alliance, O-RAN near-real-time RAN intelligent controller, E2 application protocol (E2AP) 2. 02, 2022.
[116] O. Alliance, O-RAN near-real-time RAN intelligent controller E2 service model (E2SM), ran function network interface (ni) 1.0, 2020, Technical Specification.
[117] Atomix:A reactive Java framework for building fault-tolerant distributed systems, 2023, https://atomix.io.
[118] Next generation architecture of ONOS, 2019, https://docs.onosproject.org/.
[119] S.T. Arzo, D. Scotece, R. Bassoli, D. Barattini, F. Granelli, L. Foschini, F.H. Fitzek, MSN: A playground framework for design and evaluation of microservices-based SDN controller, J. Netw. Syst. Manage. 30 (2022) 1–31.
[120] 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 Comput. Commun. Rev. (2014).
[121] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, M. Menth, A survey on data plane programming with P4: Fundamentals, advances, and applied research, J. Netw. Comput. Appl. 212 (2023) 103561.
[122] K. Birnfeld, D.C. da Silva, W. Cordeiro, B.B.N. de França, P4 switch code data flow analysis: Towards stronger verification of forwarding plane software, in: NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium, IEEE, 2020, pp. 1–8.
[123] Q. Huang, P.P. Lee, Y. Bao, Sketchlearn: Relieving user burdens in approximate measurement with automated statistical inference, in: Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, 2018, pp. 576–590.
[124] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, V. Braverman, One sketch to rule them all: Rethinking network flow monitoring with univmon, in: Proceedings of the 2016 ACM SIGCOMM Conference, 2016, pp. 101–114.



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 ETRI, 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 systems.

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 at Department of Computer Engineering, Dankook University, South Korea. He received his Ph.D. and M.S. degree in School of Computing (Information Security) from KAIST and his B.S. degree in Computer Science and Engineering from Sogang University in Korea. His research interests focus on networked systems and security. He is especially interested in performance and security issues in cloud and edge computing systems, including SDN/NFV, IoT, containers.

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.