Software-Defined Network (SDN) Data Plane Security: Issues, Solutions and Future Directions

Arash Shaghaghi [1,3] · **Mohamed Ali Kaafar** [2] · **Rajkumar Buyya** [3] · **Sanjay Jha** [1]

Abstract Software-Defined Network (SDN) radically changes the network architecture by decoupling the network logic from the underlying forwarding devices. This architectural change rejuvenates the network-layer granting centralized management and re-programmability of the networks. From a security perspective, SDN separates security concerns into control and data plane, and this architectural recomposition brings up exciting opportunities and challenges. The overall perception is that SDN capabilities will ultimately result in improved security. However, in its raw form, SDN could potentially make networks more vulnerable to attacks and harder to protect. In this paper, we focus on identifying challenges faced in securing the data plane of SDN - one of the least explored but most critical components of this technology. We formalize this problem space, identify potential attack scenarios while highlighting possible vulnerabilities and establish a set of requirements and challenges to protect the data plane of SDNs. Moreover, we undertake a survey of existing solutions with respect to the identified threats, identifying their limitations and offer future research directions.

Arash Shaghaghi, E-mail: a.shaghaghi@unsw.edu.au
Mohamed Ali Kaafar, E-mail: dali.kaafar@data61.csiro.au
Rajkumar Buyya, E-mail: rbuyya@unimelb.edu.au
Sanjay Jha, E-mail: sanjay.jha@unsw.edu.au

[1] School of Computer Science and Engineering,
The University of New South Wales (UNSW), Australia
[2] Data61, CSIRO, Australia
[3] Cloud Computing and Distributed Systems (CLOUDS) Lab, School of Computing and Information Systems,
The University of Melbourne, Australia

1 Introduction

Traditional IP network devices are purpose-built and application-specific with integrated circuits and chips designed to achieve high throughputs. This 'hardware-centric' network model requires network operators to configure each device separately through low-level vendor-specific commands. Hence, in heterogeneous networks, network configuration is tedious, and automatic reconfiguration and response are virtually impossible. Moreover, the data and control plane bundling reduces flexibility, hinders innovation and slows down the evolution of networking infrastructure. In fact, as shown by recent studies, in the long run, traditional networks are incapable of coping with the increasing demand and continuous expansion in the number of devices and applications brought through advances in Cloud Computing, Internet-of-Things (IoT), and Cyber-Physical Systems [35, 45, 49, 54].

Software-Defined Network (SDN) is a new network paradigm that decouples the control from data plane in networking devices. This architectural recomposition places the 'brain' of the network on a specialized central controller, enabling centralized management and global view of the network. The data plane is composed of 'dummy' devices, forwarding packets based on rules specified remotely. These rules may be specified by the application running atop of the controller and triggered according to packet-level extracted information.

SDN's layered architecture follows the 'separation-of-concerns' principle [24], which is a fundamental security engineering requirement and is missing in today's Internet architecture. Hence, in theory, SDN lays a solid ground for improving the security of networks, and tremendous efforts have already been made to leverage the capabilities of SDN to enhance security for both

network providers and users. The SDN security literature is split into research aiming to secure software-defined network platform itself, solutions attempting to enhance existing network security services (e.g. firewalls) and proposals on creating new security services. In this paper, we take the first direction and our focus is on the security threats associated with data plane of SDNs.

The roadmap ahead is as follows: we start by presenting an overview of SDN architecture and capabilities, where we highlight the latest advances and trends in SDN data plane research (§2). Thereafter, present a taxonomy of attacks against SDNs based on their scope and impacts (§3). To the best of our knowledge, this categorization does not exist in the literature, which we believe is fundamental for better understanding of different attack impacts in SDNs and develop practical countermeasures. In §4, we formalize the problem of securing SDN's data plane, identify the attack vectors and highlight the underlying vulnerabilities. Afterwards, we establish a set of requirements to protect SDNs against compromised forwarding devices and accordingly, survey existing practical solutions. Finally, we present a set of future research directions in §5 and conclude the paper in §6.

2 Software-Defined Network (SDN)

Traditionally, computer networks have been divided into three planes of functionality namely, the management plane, the control plane, and the data plane. In a nutshell, network policies are devised at the management plane and passed to control plane for enforcement and executed at the data plane. Hence, the data plane refers to the network forwarding devices that forward the packets, the control plane represents the protocols used to configure the forwarding tables, and management plane includes the set of software services used to configure the control functionality (e.g., SNMP, NETCONF, etc.). Traditional IP networks, follow a 'hardware-centric' model where the control and data plane are developed and embedded in the same device by the device vendor. The resulting outcome has been quite effective in terms of network performance and resilience. Nevertheless, this architecture is very resistant to change, slow in adopting innovations and quite complicated to setup, troubleshoot and manage.

Software-Defined Network (SDN) has emerged with its largest special envoy being the loose coupling between the control and data plane. Hence, SDN moves away from a vertical integration of network components to a horizontal one and adds distinctive separate functioning layers for policy definition, enforcement and im-

plementation. We present an overview of SDN architecture and its main components in §2.1. Here, we are mostly concerned with SDN's data plane, and in §2.2 we include a more detailed revision of this layer, where we also discuss the recent trends with stateful data planes.

2.1 Architecture and Main Components

Software-Defined Network framework facilitates networks programmability and grants the ability to manage, amend and control the network behavior dynamically via open interfaces. It enables centralized control of data plane forwarding devices independent of technology used to connect the devices while maintaining live and centralized network-wide view of all the data path elements. SDN enables long-awaited features such as on-demand resource allocation, self-service provisioning and truly virtualized networking through its intelligent orchestration and provisioning system. The high-level reference SDN architecture promoted by the Open Networking Foundation (ONF) is shown in Figure 1. The architecture is composed of three main layers namely, the Data Plane, Control Plane and Application Plane. Each layer has its own specific functions and the components that are always present in an SDN deployment include the Southbound API, SDN Controller (or Network Operating System), Northbound API and network applications. In the following, we present a succinct overview for each of these components through a bottom-up approach. Understanding the core properties of these components play a role when designing solutions to secure the data plane of SDNs.

2.1.1 Data Plane

The data plane is composed of networking equipments such as switches and routers specialized in packet forwarding. However, unlike traditional networks, these are just simple forwarding elements with no embedded intelligence to take autonomous decisions. These devices communicate through standard OpenFlow interfaces with the controller - which ensures configuration and communication compatibility and interoperability among different devices. An OpenFlow enabled forwarding device has a forwarding table, which is constitute of three parts: 1) Rule matching; 2) Actions to be executed for matching packets; and 3) Counters for matching packet statistics. The rule matching fields include Switch port, Source MAC, Destination MAC, Ethernet Type, VLAN ID, Source IP, Destination IP, TCP Source Port, TCP Destination Port. A flow rule may be defined a combination of these fields. The most
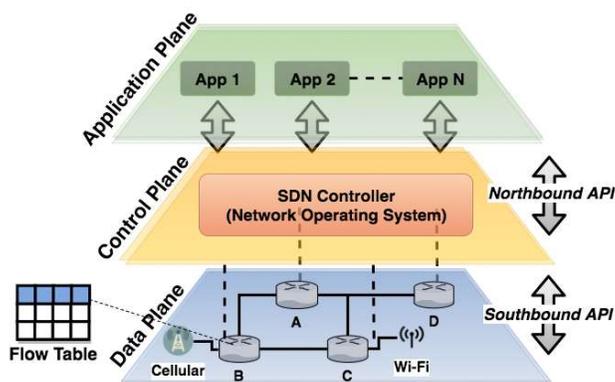
Fig. 1: SDN Architecture

common actions include 1) Forward the packet to outgoing port(s); 2) Encapsulate and forward to controller; 3) Drop; 4) Enqueue and 5) Modify Field. The most common case is to install a default rule to instruct the switch to forward the packet to the controller for a decision.

SDN has enabled the introduction of Software switches, which are deemed to be a promising solutions for data centers and virtualized network infrastructures [7, 9]. These have been very attractive to data center networks with the most dominant examples being Open vSwitch [6], Switch Light [4], ,Pica8 [8], Pantou [2] and XorPlus [10].

2.1.2 Southbound API

The southbound API is one of the most critical components of an SDN system, which bridges in-between forwarding devices and the control plane. Currently, OpenFlow is the most accepted standard for southbound standard - there are several alternatives to OpenFlow such as POF [65], OpFlex [64] or OpenState [23] but have not seen adoption yet. The OpenFlow protocol provides three main type of information to the Network Operating System (NOS): 1) Packet-In message whenever a forwarding device does not have a matching flow rule for a packet or there is an explicit rule for a packet specifying this; 2) Event-based messages whenever a link or port change is triggered; 3) Flow statistics generated by the forwarding devices.

2.1.3 SDN Controller

The 'brain' of the network, which generates network configurations based on the policies defined by the network operator. It abstracts the lower-level details and makes them available to the application plane through essential services (e.g., network topology, state, device discovery, and etc.) and common APIs to developers.

A diverse set of controllers are available each with their own design and architecture. One of the most prevailing factors in differentiating available controllers is as to whether the controller has a centralized or distributed architecture. With a centralized controller, a single entity is responsible to manage all of the forwarding devices. This architecture has two main limitations: 1) Single point of failure threat; and 2) Scaling limitations. The best known centralized controllers that can achieve the level of throughput required by data center networks include NOX-MT [68], Maestro [55], Beacon [31], Ryu NOS [3] and Floodlight [4]. These NOSs employ multi-threaded designs deploying parallel multicore architectures and achieve processing capabilities such as to deal with up to 12 million flows per second [31]. Distributed controllers such as ONOS [22], Onix [43], HyperFlow [67], PANE [33] and DISCO [59] provide much better scaling support and are much more resilient to logical and physical failures by spreading independent controllers across different network segments. Distributed controllers are equipped with East and Westbound APIs allowing controllers to exchange data, algorithms for consistency models and monitoring information. SDNi [70] is an attempt to standardize the East and Westbound interface.

A typical SDN controller provides a set of core functionalities including topology manager, stat manager, device manager, notification manager, shortest path forwarding and security mechanisms. The first four components are self-descriptive in a networked environment and the security mechanisms provide services such as isolation and security enforcement between services and application (e.g., rules generation by low priority services do not overwrite rules created by high priority applications).

2.1.4 Northbound API

The northbound API is a software ecosystem similar to the Southbound API placed in-between the application and control planes. Controllers offer quite a broad variety of northbound APIs such as ad hoc and RESTful APIS, multilevel programming interfaces and file systems. However, as of today, there is no standardization for the northbound API yet. In fact, existing controllers such as Floodlight, Onix, and OpenDaylight [51] implement their own northbound API with different specifications and programming languages.

2.1.5 Application Plane

The network applications dictate the behavior of the forwarding devices. The applications submit high-level policies to the control plane, which is responsible for enforcing these policies by implementing them as flow rules on network forwarding devices. Many different SDN applications have already been developed, and the the current focus is to have an App Store support for SDNs [45], where customers can dynamically download and install network apps. Most SDN applications fall into one of the following five categories: traffic engineering, mobility, and wireless, measurement and monitoring, security and dependability and data center networking. We will provide an overview of security and dependability application in §3, and we refer the interested reader to [45] for a detailed survey of the existing network applications.

2.2 Latest Advances in SDN Data Plane

The original OpenFlow standard was too restrictive and various proposals have emerged adding extra flexibility to it. They can be categorized in three different directions including A) adding multiple flow tables to forwarding devices, B) improving the match rule flexibility and C) stateful data planes. While our focus is on the third-case (i.e., stateful SDN data planes), we include a summary of research in the first two cases as well.

2.2.1 Adding multiple flow tables to forwarding devices

The single flow table model proposed in original OpenFlow would cause two main problems. First, it would force the developer to deploy OpenFlow rules that combine the Cartesian product of all the required matches. Other than complexity, the exponential increase in the deployed rules would be an extremely inefficient use of flow table capacity, which is typically implemented through the constrained TCAM memory of forwarding devices. Secondly, many applications would benefit from two-stage processing model [34], where packets are first tagged based on some packet characteristics and then matched with rules.

An OpenFlow forwarding devices supporting multiple flow tables (introduced since OpenFlow 1.0 [37]), consist of flow tables composed as a pipeline, where a path through a sequence of flow tables defines how packets should be handled. Proposals have emerged developing advanced hardware solutions enabling a more flexible multiple matching. For instance, Reconfigurable Match Table (RMT) [25], permits flexible mapping, an arbitrary number of match tables having different widths and depths and even supporting matching rules using parameters computers from previous matches.

2.2.2 Improving the match rule flexibility:

The first OpenFlow version just matched 12 fields. However, greater matching flexibility was recognized as different network applications at different layers may require the capability to associate actions to different types of matches. Hence, the recent OpenFlow versions add support for more than 40 fields including finer-grained fields such as TCP flags.

2.2.3 Stateful Data Planes:

In order to reduce the switch-to-controller signaling and the associated latency issues, recently there have been proposals to introduce some very specific stateful operations into the data plane forwarding devices. One of the most motivational use-cases for this emerging trend is link failure. With the original OpenFlow specification, when a data link fails, the forwarding device should seek instructions from the controller to setup rectifying rules (e.g., forward all traffic to a backup link). In this case, there is a small interval of time when all packets will be lost, and in large networks, the resulting number can be quite large. For instance, for a 100ms response time from the controller on 10 Gbps link would cause up to 30 million packets to be lost. Recent versions of OpenFlow specification introduce optional support for some stateful capabilities - e.g., fast failover, select group type, and synchronized tables for learning type functionalities are available in OpenFlow version 1.5.1 [1].

Up to this date, three proposals have emerged for stateful SDN data planes, which go beyond the basic features introduced with OpenFlow and that require architectural upgrades to be deployed. In general, stateful SDN data plane proposals have three basic principles

in common: 1) retaining state information of the flows within forwarding devices, 2) support programmable packet-level state transition in forwarding devices, 3) granting the forwarding devices the permission to update forwarding states based on flow's local state information without requiring them to seek instructions from the SDN controller.

OpenState [23] introduces programmability to SDN's data plane through a special case of eXtended Finite State Machines (XSFM). Each forwarding device keeps two separate tables: State Table and XFSM Table (see Figure 2). The former, stores the current state of the flow based on packets received, which are relevant to that flow. The latter, however, is used to define the rules based on packet's received information. XFSM is modeled as (S, I, O, T), where S is a finite set of states including the start state S_0; I is a finite set of events (inputs); O is a finite set of actions (outputs); and T is a rule (state of transition) that maps ⟨state, event⟩ to ⟨state, action⟩. With OpenState, packet processing in a forwarding device is a three-step process. First, the packet's current state is retrieved from the state table and appended to the packet as a metadata field. If, however, the state table lookup retrieves no match, then the forwarding device assigns 'default' state to the incoming packet. Thereafter, the forwarding queries the XFSM table to find the matching rule with ⟨state, event⟩ pair, executed the associated action and updates the state field of the packet as per the 'next-state' field, which is pre-defined in the XFSM table. Finally, the forwarding device's state table is updated based on the 'next-state' value retrieved in the previous step for the corresponding flow.

Similar to OpenState, *FAST [53]* also stores pre-installed state machines inside each forwarding device. In FAST, however, each forwarding device could have several instances of the state machine and each is dedicated to a special application. Moreover, instead of two tables used in OpenState, the data plane implements four tables including 1) State Machine Filter, 2) State Table, 3) State Transition Table, and 4) Action Table (see Figure 2). There is one state machine filter table for all the instances of the state machine, while each state machine has its own state table, state transition table, and action table. The state machine filter table is used to select the corresponding state machine related to a packet. The state table is a hash table mapping each packet header to a flow state, where each state is stored inside a variable along with its current value - simply put, the state table stores the state information of flows. The forward device uses the state transition table to identify the 'next-state' of a packet based on its current state and packet fields. Lastly, the action table

specifies the actions that the forwarding device should execute on the packet based on the packet header and its new state.

FAST also upgrades the control plane of an SDN by introducing two new components: 1) a compiler and 2) forwarding device agents. The former being an offline component responsible to translate the state machines into forwarding device agents. The latter, however, are online components, which 1) manage the sate machines inside the forwarding devices, 2) perform certain local computations based on the updates received from the forwarding device, 3) manage memory restrictions for confined switches through partial implementation of the state machine inside the forwarding device, and 4) handle communication between the forwarding device and the controllers while updating the controller about the local status of the switch.

Another proposal is *SDPA [71]*, which is composed of three main tables including State Table, State Transition Table, and Action Table (see Figure 2), as well as a state processing module called Forwarding Processor (FP). The state table has three field values: Match, State, and Instruction. The 'Match' value could be any combination of the header fields of the packet, 'State' value is the flow's current state, and 'Instruction' value may be specified either for a 'state' or 'a packet'. Here, the SDN controller communicates with the FP module and initiates the state tables. Moreover, the controller maintains full control and updated information of the state tables via communications with FP either through periodic or specific event updates. With SDPA, when the first packet of a flow is received, the forwarding device sends the packet to the controller, which determines the state table that the corresponding flow state should be stored in. The rest of the packets pertaining to that flow will be processed locally inside the forwarding device without the controller's intervention.

Several different type of applications have already been built atop of the aforementioned stateful data plane solutions including stateful failure recovery [], HULA [38], port knocking [23], SDN tunneling stateful detection [20] and UDP flooding stateful mitigation[20]. While reviewing each of these solutions is out of scope in this work, we refer the interested reader to [28] for a survey.

3 SDN Security Analysis

3.1 SDN's Vulnerabilities

Compared to traditional networks, five characteristics of a software-defined network can have the most impact on its security given that they potentially add to
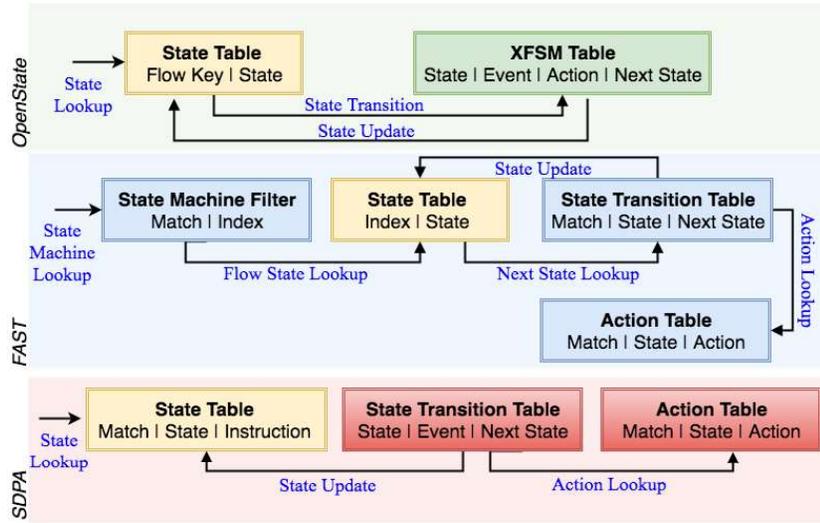
Fig. 2: Tables used in OpenState, FAST and SDPA architectures. Each table is represented as a rectangle containing the table name and the corresponding table columns separated with '|'.

the number of vulnerabilities. As illustrated in Figure 3, these characteristics include a centralized controller, open programmable interfaces, forwarding device management protocol, third-party network services and virtualized logical networks. We provide a succinct summary of these characteristics before analyzing the security of SDNs.

Centralized Controller: As discussed in §2.1, SDN provides a logically centralized control plane to the network. The controller of the network maintains a global view of the network and programs forwarding devices as per the policies defined at the application layer. While initially controllers were developed as single devices, recently there has been a shift of trend to distributed controllers with the goal of adjusting to scalability and reliability requirements of real-world scenarios. In this case, each set of forwarding devices is assigned to a specific instance of controllers and the controllers, follow a Master/Slave deployment model.

Open Programmable Interfaces: In SDNs, there are three main programmable interfaces: A) application plane to control plane, B) east and westbound in the control plane and C) control plane to data plane. Compared to traditional networks, these open interfaces are what make an SDN programmable. A, which also known as Northbound API (see §2.1-IV), enables SDN applications to submit policies to the control plane of the network - e.g., REST APIs. B, is an interface allowing communication among different inter-connected controllers, which may or may not be running in the same domain. C is the southbound API (see §2.1.II), which is the most developed and discussed interface in SDNs up to this

date. OpenFlow is the agreed standard for the controller to data plane communications, which allows a controller to program a forwarding device irrespective of the underlying hardware or software in the controller or the forwarding device.

Forwarding Device Management Protocol: The forwarding device management protocol along with OpenFlow enables configuration and management of programmable forwarding devices. For instance, the OF-Config protocol may be used to configure an OpenFlow enabled device as well as multiple logical forwarding devices that may be initiated on top of that device. Another example of this protocol includes the OVSDB [58].

Third-party Network Services: As an operating system, an SDN controller supports the installation and execution of third-party network services. This allows easy customization, development and innovation, and reduced costs of proprietary services. Third-party services may communicate with a controller either via internal APIs or open Northbound APIs. Moreover, depending on the controller used, applications may be compiled as part of the controller module (e.g., NOX and POX) or may be instantiated at run-time (e.g., OpenDayLight).

Virtualized Logical Networks: Network Function Virtualization (NFV), created by a consortium of service providers, is tightly coupled with software-defined networks but it does not depend on SDN for its existence. In a nutshell, NFV virtualizes network services, which were previously hardware-based. It focuses on optimiz-
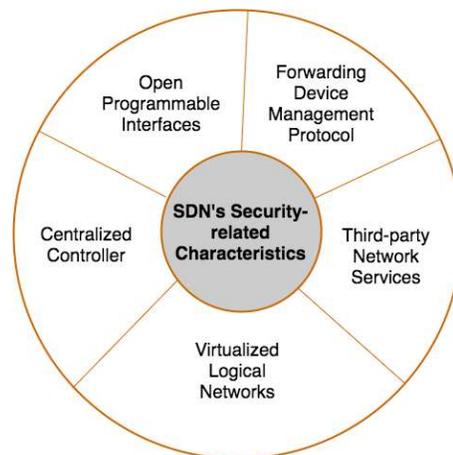
Fig. 3: SDN's Five Main Security-related Characteristics Posing Security Issues

ing network services themselves by decoupling network functions (e.g., DNS, caching, and etc).

While independent, the combination of SDN and NFV leads to a greater value and potential. In fact, in many cases, SDN is linked to server virtualization by enabling multiple logical forwarding devices being instantiated over a shared physical device. This potential has already been explored in the literature, and various proposals have emerged [35].

3.2 Attack Scopes, Vectors and Impacts

Compared to traditional networks, the SDN architecture separates the definition and storage of network policies from their enforcement and implementation. Accordingly, we categories attacks targeting SDN's five main components (see 2.1) as per their impact on network's policy, enforcement, and implementation. Figure 4 illustrates the three main attack types and their relation to SDN's main components. Here, we are focusing on direct associations and potentially, the main motivations of an adversary when targeting any of the SDN's core components. Indeed, attacks against each of the five main components may indirectly fit into all of the three different attack scopes.

3.3 Implementation Attacks

Attacks targeting the Southbound API and Data Plane components of a software-defined network are categorized under 'Implementation Attack'. Figure 5 shows a taxonomy of the main threat vectors associated with this type of attack.

Three different attacks may be used to compromised a data plane including Device Attack, Protocol Attack, and Side Channel Attack. A *Device Attack* refers to all those attacks, where the adversary aims to exploit software or hardware vulnerabilities of an SDN-capable switch to compromise SDN's data plane. For example, an attacker may target software bugs (e.g., firmware attacks) or hardware features (e.g., TCAM memory) of a forwarding device. Similarly, a *Protocol Attack* refers to attacks targeting the data plane of an SDN by exploiting network protocol vulnerabilities in the forwarding devices (e.g., BGP attacks). A *Side Channel Attack* in this context refers to the case where an attacker may deduce the forwarding policy of the network just by analyzing the performance metrics of a forwarding device. For example, an input buffer may be used to identify rules, and by analyzing the packet processing times, an attacker could identify the forwarding policy [60].

There are four main attacks against the Southbound API of an SDN including Interaction, Eavesdrop, Availability and TCP attacks. While with an *Eavesdrop Attack*, the attacker aims to learn about information exchanged between the control and data plane as part of a larger attack plot, in an *Interception Attack* the attacker's goal is to corrupt the network behavior by modifying the messages being exchanged. The *Availability Attack* refers to Denial of Service (DoS) attacks, where the Southbound API is flooded with requests causing the network policy implementation to fail.

3.4 Enforcement Attacks

'Enforcement Attacks' aim to prevent a software-defined network to properly instruct when, where and how the policies should be enforced in the network. Hence, at-
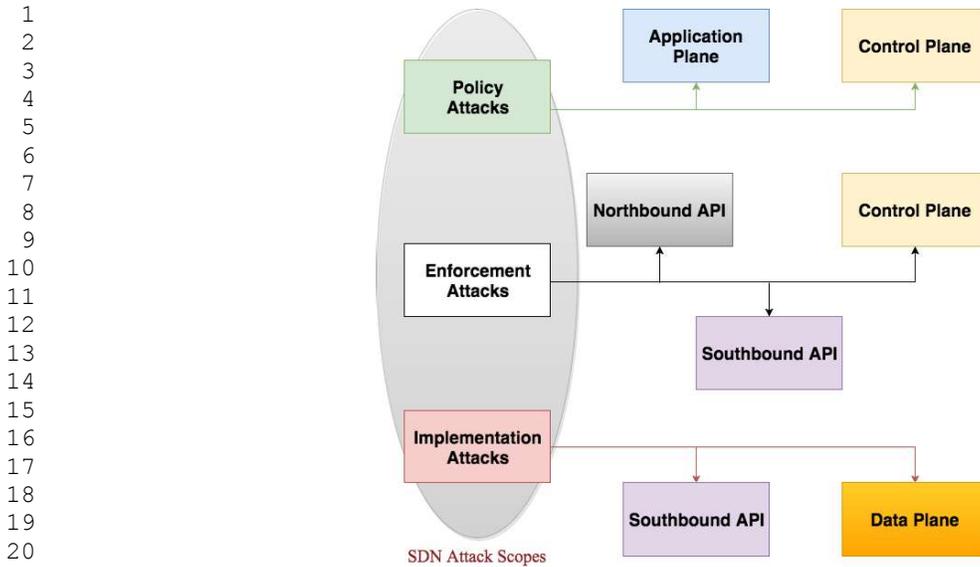
Fig. 4: The Relation Between SDN's Five Main Components and the Attack Scopes

tacks targeting the Control Plane, Southbound API and Northbound API may be associated with attacks targeting policy enforcement. Figure 6 illustrates a taxonomy of different attack vectors targeting the enforcement of network policies.

Earlier in §3.3, we denoted different attacks against Southbound API. These attacks may also have an adverse impact when it comes to policy enforcement in an SDN as well. For instance, using a Man-in-the-Middle attack (MITM) attack, an adversary may alter message exchanges such as $Packet\_In$[1] message or $Flow\text{-}mod$[2] and tamper with the controller's understanding of the requirements of the data plane. As well as malicious attacks, the lack of well-defined standards and constant changes in SDN's Southbound API could lead to unwanted, yet malicious involvement in policy enforcement process. For instance, an improperly configured message exchange could lead to invalid or conflicting instructions being set or distributed in the data plane. Similar to SDN's Southbound API, the Northbound API is susceptible to Interception, Eavesdrop and Availability Attacks. While the nature of both attacks is similar, there are a few key differences: 1) An attacker targeting the Northbound API requires higher-level of access to the system and is potentially sitting on the application plane. There may be cases that the applications do not run on the same device (see [63] for

example) and in that case the attack complexity may be reduced as to Southbound API (e.g. where the adversary targets the communication link); 2) The impacts of a compromised Northbound API are potentially larger given that information exchanged between the control and application plane affect network-wide policies. Unlike Southbound API, where OpenFlow is adopted as the standard, the Northbound API lacks any standardization. Specifically, each controller has different specifications for the Northbound API, and this leads to insecure developments. Moreover, a poorly designed Northbound API could also be exploited by malicious applications to manipulate the behavior of other applications through eviction of active sessions, tampering with control message exchanges, and etc.

The third set of attack vectors against policy enforcement originate from the control plane – potentially being the most critical threat against SDNs. Attacks targeting SDN's control plane may be classified into three types: Manipulation Attack, Availability Attack, and Software Hack. *Manipulation Attack* refers to any attempt by an adversary to subvert the controllers understanding of the data plane, which ultimately leads to 'improper' decision making. For instance, an LLDP (Link Layer Discovery Protocol) related as $Packet\_In$ messages may be used to created fabricated links and network topologies. Similarly, an ARP (Address Resolution Protocol) packet relayed as a $Packet\_In$ message could adversely affect the view of the controller. An SDN controller is hosted on a commodity server and may be subject to *Software Hacks* as any other application. For instance, altering a system variable such as

---

[1] A $Packet\_In$ message is sent by a forwarding devices to the controller when a packet does not match any of its flow rules.

[2] A $Flow\text{-}mod$ message allows the controller to modify the state of an OpenFlow switch.
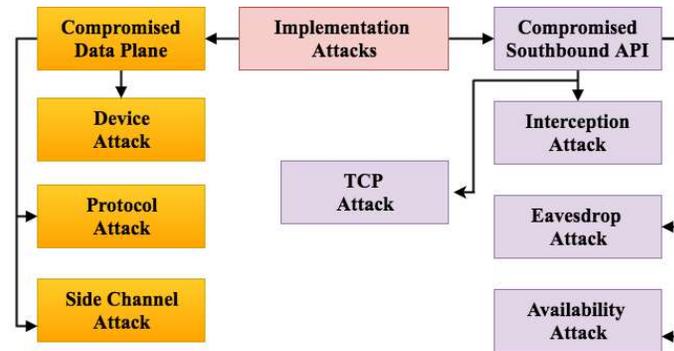
Fig. 5: Taxonomy of Attack Vectors Related to 'Implementation Attacks'

time may effectively turn the controller offline. In the case of *Availability Attack*, the adversary aims to make the controller unavailable for a certain period of time for part or all of the network. One way to achieve this is for an attacker to flood the controller with *Packet_In* messages - given that these are not authenticated.

Scalability and configuration conflicts are also vulnerabilities that may be exploited by opportunistic adversaries. An SDN controller is responsible for all decisions in an SDN. Evidently, a single controller will not scale well for large and dense network with a 10 Gbps link network. As discussed in [18], this may be used to deliver attacks such as saturation and single point of failure. Finally, the combination of a single-domain multiple controllers, multi-tenant controllers and multiple OpenFlow architectures may lead to configuration conflicts.

3.5 Policy Attack

'Policy Attacks' refer to the threats targeting SDN's ability to define and store proper network policies. As illustrated in Figure 7, an attacker aiming to target network's policy level aims for compromising SDN's control and application planes. By compromising the controller, an attacker could alter the information shared with applications about the network and compromise their decision making. Potentially, this type of attack may be part of a larger stealthy plot to compromise the network status in the long run or to avoid detection by an intrusion detection system – given that with a compromised controller the attacker has almost full access to the network. In alternative scenarios, the adversary's access to the controller may be restricted to certain functions or period of time motivating an attack to the application plane. Evidently, an 'honest' configuration conflict in the control plane could also be exploited by an adversary to deliver a 'Policy Attack'.

SDN allows the installation of third party apps and as mentioned in §2.1.V, the current goal is to setup an App Store ecosystem for this. Similar to mobile devices, third-party application support adds to the threat vectors against SDNs and ensuring the security and trustworthiness of apps is challenging. For instance, nested applications have been identified to pose dangerous threats [52, 69]. Generally, attacks targeting SDN's application plane may be categorized into: Storage, Control Message, Resource, Authorization, Authentication and Availability Attacks.

Storage Attack: SDN applications are granted access privilege to shared storage. This access may be exploited to manipulate the internal database targeting the network behavior.

Control Message Attack: Control messages exchanged between the control and data plane are fundamental for functioning of an SDN. An arbitrary issued control message by an application might be catastrophic. For instance, a malicious application may take down the network by sending control messages modifying or clearing the flow table entries of switches.

Resource Attack: Malicious applications may exhaust expensive and critical system resources including memory and CPU thereby seriously affect the performance of legitimate applications and the controller itself. Moreover, a malicious SDN application may execute system exit command and dismiss controller instances.

Access Control Attacks: A common feature among Storage, Control Message and Resource Attacks is access violation. In fact, there is a very limited control in terms of authentication, authorization and accountability in current controllers. We classify all attacks violating required access authorization as Access Control Attack.
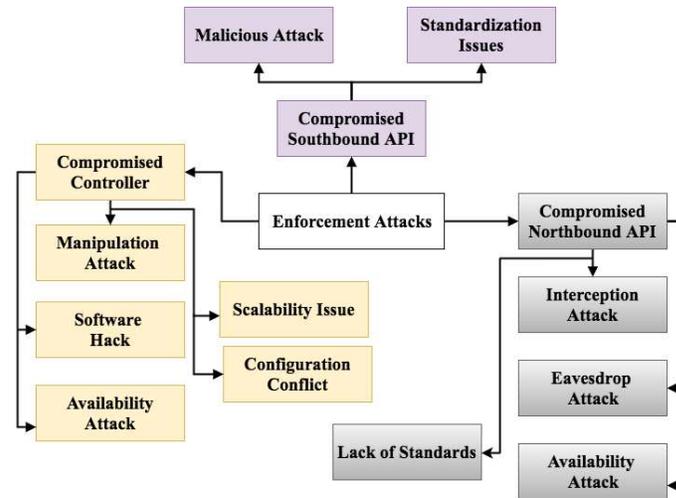
Fig. 6: Taxonomy of Attack Vectors Related to 'Enforcement Attacks'
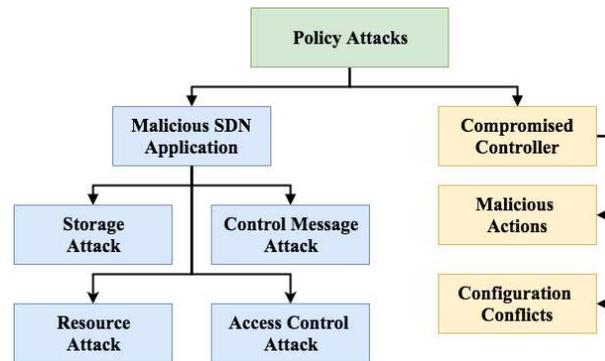
Fig. 7: Taxonomy of Attack Vectors Related to 'Policy Attacks'

4 Securing the Data Plane of SDNs

In general, SDN security domain may be split into four main directions. First, research aiming to import existing security services such as firewalls into SDN. Second, proposals on how to upgrade existing services by leveraging the SDN capabilities. These two directions consisted most of the research in the first few years after SDN's introduction. The recent trend, however, has shifted towards developing innovative security services, which were not feasible to implement before SDN. For instance, using network capabilities to secure Internet-of-Things (IoT) devices (e.g., [56]), smart grid (e.g., [30]), clouds [26] or having inter-layer access control systems [63]. The fourth direction is research aiming to secure the SDN platform itself, which is a critical requirement and has the most direct impact on SDN's adoption. In general, the focus on the security of a technology itself is very much driven by the adoption rate as many threats are discovered and addressed as cus-

tomers deploy it. Fortunately, major industry players such as Google and HP [45] have adopted SDN, and this has boosted research in this area.

Several comprehensive surveys have been published summarizing the ongoing efforts in this area including [19, 45, 60, 61]. Here, focusing on SDN's security, we provide an alternative categorization of the SDN security literature (see Figure 8). Out of the four research directions mentioned earlier, all research falling into any of the first three categories is classified as *SDN-based Security Services*. We categories research aiming to secure SDN itself into three groups including research aiming to protect A) SDN's five main components, B) its core features and C) implementations. SDN's core features include centralized management and programmability and various proposals have been developed to ensure these are protected [60]. SDN's implementation includes security different controller platforms, securing the OpenFlow protocol design and im-

plementations, OpenFlow-enabled devices and software forwarding devices such as Open vSwitch. An alternative way of decomposing the literature is according to the SDN components that proposals aim to secure. In §3.2, we discussed different attack vectors targeting SDN's main components as well as their impacts. However, our literature evaluation indicates that one of the least explored areas of SDN security literature is securing its data plane. In fact, even with the latest proposals, there seems to be an oversight regarding the malicious forwarding devices that may exist in SDN data plane [62].

4.1 Why It Matters?

Network forwarding devices have been a very attractive target for attackers. In fact, given the large amount of information that may be exposed through compromised forwarding devices, resourceful adversaries including intelligence agencies have for long aimed to setup backdoors on them. For instance, Edward Snowden uncovered massive investments by NSA to enable large scale surveillance through core network infrastructure [13, 14]. More recently, the 'Vault 7: CIA Hacking Tools' revelations by WikiLeaks [17] disclosed that CIA had actively exploited a common vulnerability in 318 different Cisco routers to carry out surveillance attacks globally. There have also been WikiLeak's revelations on NSA' upgrading labs tampering with forwarding devices before they are released to the market [12]. However, the attack surface against forwarding devices is not limited to resourceful adversaries. Software and hardware vulnerabilities of the devices [15, 21, 27, 47] and vulnerable implementations of network protocols enable attackers to compromise forwarding devices. For instance, as reported in CVE-2014-9295 [16], a novice hacker could execute arbitrary code on routers simply through crafted packets targeting a specific function of the device [11].

A compromised forwarding devices may be used to drop or slow down, clone or deviate, inject or forge network traffic to launch attacks targeting the network operator and its users. For instance, the documents disclosed under 'Vault 7' revelations indicate that compromised routers may have been used for activities such as data collection, exfiltration (e.g. Operation Aurora [5]), manipulation and modification (insertion of HTML code on webpages) and cover tunneling. A compromised routing system may be also used to bypass firewalls and intrusion prevention systems [32], violating isolation requirements in multi-tenant data centers [40], infiltrate VPNs [46] and more.

As discussed in [42, 44], compromised forwarding devices may be used to take down an entire software-defined network. Compared to traditional networks, this shows a compromised forwarding device poses a much higher risk to SDNs. Moreover, as discussed in [29] and [62], SDN also adds to the complications in protecting networks against compromised forwarding devices. This is mainly associated with the following four factors.

- *Incompatibility of the existing solutions:* With the removal of intelligence from the forwarding devices, the defense mechanisms used for traditional networks no longer work in SDNs. In fact, in order to import traditional defenses into SDNs, we would need a fundamental redesign of OpenFlow protocol [50].
- *Unverified reliance of the control plane to the data plane:* SDN controllers rely on *Packet_In* messages for their view of the network. However, as discussed in §3.5, these messages are not authenticated nor verified. A malicious forwarding device may send forged spoofed messages to subvert the controller view of the network – even with TLS authentication in place. The same vulnerability enables a compromised forwarding device the capability to overload the controller with requests launching a Denial of Service (DoS) attack.
- *Software Switches:* Programmable soft-switches such as Open vSwitches run on top of end host servers. Hence, compared to physical switches, soft-switches are more susceptible to attacks with a comparatively larger attack surface.
- *Stateful SDN forwarding devices:* We have already reviewed stateful SDN switches in §2.2.C. In general, adding some intelligence and authority to the data plane has performance advantages such as lower latency response to network events and improved fault tolerance through continuation of basic network operations under failing controllers. Furthermore, well-standardized protocols such as for encryption, MAC learning and codec control message (CCM) exchanges also require some intelligence at data plane. However, these proposals revive some of the vulnerabilities of traditional networks under SDNs.

4.2 Security Implications of Stateful Data Planes

In §3.3, we reviewed attacks targeting the data plane of SDNs and categorized possible attacks into Device Attack, Protocol Attack and Side Channel Attack. Here, we analyze the security implications of the most recent research direction in SDN data planes (i.e. stateful SDN data planes). Our main motivation is to showcase how
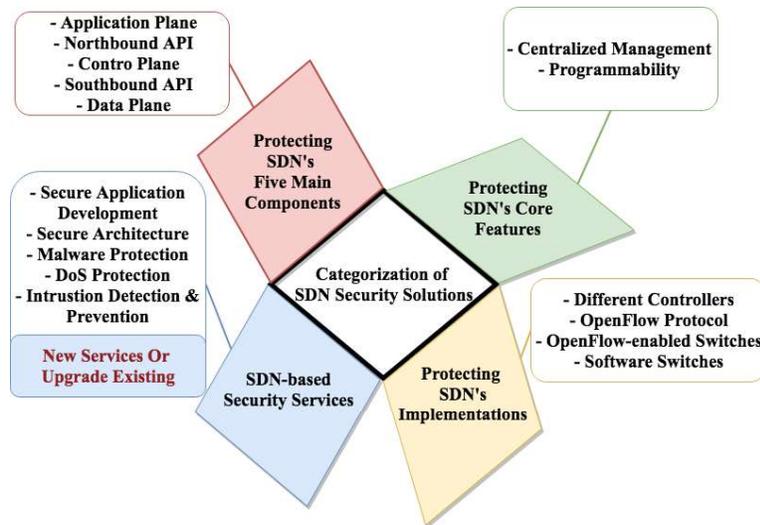
Fig. 8: Categorization of SDN Security Solutions

constant evolution of this technology affect its security and highlight the importance of ongoing research at all levels and for all components of SDNs (see §3).

Compared to standard SDN data planes, three main types of vulnerabilities are added with stateful SDN data planes:

— Unbounded Flow State Memory Allocation: As discussed in §2.2.C, in order to make data planes programmable, each forwarding device must be equipped with memory space to keep track of the state transitions generated by the incoming flows. An attacker may take advantage of the large-in-memory space required for each forwarding device and exhaust the memory of the device.
— Lack of Authentication Mechanisms in the Data Plane: If control independent functionalities were to be implemented in stateful data planes, then this would require the use of probe message between forwarding devices, or information passing between switches and 'piggyback' inside regular traffic packets [28]. Securing inter-forwarding device communications is an important issue, which has almost been ignored in the literature so far. In fact, an attacker may inject fake event/packet into the network impersonating an honest device. Moreover, if the connections are not secured, an attacker may alter the information exchanged between the forwarding devices and change the specific flow states. An attacker could setup fake scenarios, where a link failure has occurred and degrade the network performance.
— Lack of a Central State Management: State inconsistency is an issue of concern for all distributed

systems including inter-linked stateful data planes. However, this is more worrisome in this case given that there exists no central entity to manage the synchronization of states inside the forwarding devices. Specifically, as state transition is triggered after packets are received, an attacker has the capability to force state transitions pushing the network into an inconsistent state.

On the other hand, the architectural changes introduced with stateful data planes reduce some of the attacks that were possible in traditional SDNs:

— Enforcement Attacks: We defined Enforcement Attacks in §3.4. Stateful data planes reduce the required communication between control and data plane. This improves the resilience of SDN against Availability Attacks by improving its scalability. As discussed, an attacker could saturate the controller's resources by flooding a large number of spoofed *Packet_In* message targeting both the communication link and controller's resources.
— Implementation Attacks: Stateful SDNs mitigate the following vulnerabilities by design: A) flow information leakage and B) exhaustible TCAM used for flow tables. In A, the attacker learns about the network configuration by passively listening to messages exchanged between the control and data plane focusing on timing information. With stateful data planes, the forwarding devices can be programmed to handle incoming flow without the need to contract the controller. Hence, flow information leakage vulnerability is to a large extend less relevant

with stateful data plane deployments. TCAM memory exhaustion is also mitigated by stateful SDN data plane [28].

4.3 Solution Requirements

As discussed in §4.1, securing the data plane of SDNs is even more challenging than traditional network. Adding to this, with ongoing improvements and changes in different SDN components designing a solution to protect networks against compromised data plane forwarding devices is a challenging task. In fact, we believe along with attempts to secure each different piece of data planes, we need a solution capable of automatically detecting compromised forwarding devices and protect networks against them. Hence, any such detection should be based on the forwarding device's main functionality (i.e., packet forwarding) rather than any protocol, software or attack. We posit the 'must-have' features of a working solution to include:

— *Scanning Methodology:* For efficiency and to reduce the detection time, the protection mechanism must systematically and autonomously prioritize forwarding devices for inspection.
— *Distinguish Malicious Actions:* The protection mechanism must be able to distinguish between the specific malicious actions (e.g. packet drop, fabrication, delay, etc.) so that it can be intelligently responded to.
— *Locate Malicious Forwarding Devices:* To effectively protect the network, the protection mechanism must be able to localize the malicious device in the network.
— *Intelligent Response to Threats:* The protection mechanism must be programmable and allow an administrator to customize the response as per the high level network policy and requirements such as Quality of Service. In fact, the data plane is a critical component of the network infrastructure and the proposed solution must not disrupt the network performance during its inspection and analysis stage.
— *Support Stateful Data Plane:* In general, the protection mechanism must be designed with the latest advances in data plane solutions. As discussed in §4.2, stateful data planes are an increasing area of focus and bring along a set of opportunities and challenges.

One of the critical requirements when developing a protection mechanism is to have a clear description of the adversary model. We posit the following adversarial model when developing a working solution:

A resourceful adversary who may have taken full control over one, or all, of the forwarding devices. This is, in fact, the strongest possible adversary that may exist at the SDN data plane. The attacker may drop, replay, misroute, delay, reorder and even generate packets (includes both packet modification and fabrication), in random or selective manner over all or part of the traffic. These capabilities grant the adversary the capability to launch attacks against the network hosts, other forwarding devices or the control plane. For example, executing a Denial of Service (DoS) attack against the control plane by replaying or spoofing *Packet_In* messages.

5 Existing Solutions

Existing proposals in SDN data plane security has been known to suffer from inaccurate adversarial model [62]. This limitation directly impacts their adoption and limits any real impact when SDNs are deployed. For instance, solutions proposed in [29, 39–41, 48] assume all, or the majority, of the forwarding devices are trustworthy. Let us look at the specific example of SPHINX [29]: authors discuss an attack exhausting the TCAM memory of a switch that will cause a switch dropping packets over a period of time. As devastating as this may be, this device cannot be used to execute attacks requiring packet modification or misrouting. In fact, to the best of our knowledge, the only solution that closely matches the intuitive working requirements discussed in §4.3 is WedgeTail [63]. To the best of our knowledge, among existing solutions, only SPHINX and WedgeTail have practical implementations and are evaluated under realistic conditions. In the following, we provide an overview of these two solutions.

5.1 SPHINX

Proposed in 2015, SPHINX is a framework to detect attacks on network topology and data plane forwarding. SPHINX is one the very few solutions to secure SDN's data plane that does not assume forwarding devices are trusted. It detects and mitigates attacks originated from malicious forwarding devices through 1) abstracting the network operations with incremental flow graphs and 2) pre-defined security policies specified by its administrator. It also checks for flow consistency throughout a flow path using a similarity index metric, where this metric must be similar for 'good' switches on the path. SPHINX architecture is shown in Figure 10 – the image is imported from author's published paper.

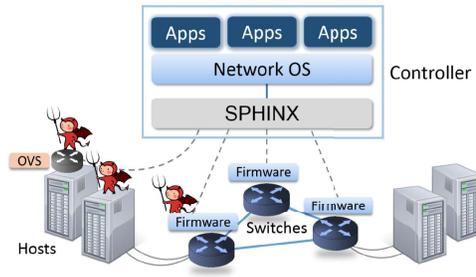Fig. 9: Required Features For a Working Solution

Fig. 10: SPHINX Architecture

SPHINX leverages the novel abstraction of flow graphs, which closely approximate the actual network operations, to (a) enable incremental validation of all network updates and constraints, thereby verifying network properties in realtime, and (b) detect both known and potentially unknown security threats to network topology and data plane forwarding without compromising on performance. It analyzes specific OpenFlow control messages to learn new network behavior and metadata for both topological and forwarding state, and builds flow graphs for each traffic flow observed in the network. It continuously updates and monitors these flow graphs for permissible changes, and raises alerts if it identifies deviant behavior. SPHINX leverages custom algorithms that incrementally process network updates to determine in realtime if the updates causing deviant behavior should be allowed or not. It also provides a light-weight policy engine that enables administrators to specify expressive policies over network resources and detect security violations.

5.2 WedgeTail

WedgeTail[3] is a controller-agnostic Intrusion Prevention System (IPS) designed to 'hunt' for forwarding devices failing to process packets as expected. It be-

gins by mapping forwarding devices as points within a geometric space and regarding packets as 'random walkers' among them. WedgeTail tracks packet paths when traversing the network and generates their corresponding trajectories. Thereafter, in order to detect malicious forwarding devices, locate them and identify the specific malicious actions (e.g., packet drop, fabrication, etc.), it compares the actual packet trajectories with the expected ones (i.e., those allowed according to the network policies). If and when a malicious forwarding device is detected, WedgeTail responds as per the administrator-defined policies. For example, an instant isolation policy may be composed of two actions. First, the potentially malicious device is instructed to reset all the flow rules and then, it is re-inspected at various intervals by re-iterating the same packet(s) originally raising suspicion. If the malicious behavior is persistent, the forwarding device may be isolated from the network.

In order to increase the probability of finding malicious devices, WedgeTail begins by prioritizing forwarding devices for inspection. It adopts Unsupervised Trajectory Sampling [57] to cluster forwarding devices into groups of varying priority based on the cumulative frequency of occurrence in packet trajectories - i.e. all of the trajectory database was analyzed. Wedgetail intercepts OpenFlow messages exchanged between the control and data plane and maintains a virtual replica of the network. It uses this virtual replica to compute the expected packet trajectories removing any trust on forwarding devices for this. The expected packet tra-

---

[3] Recently, WedgeTail 2.0 has also been developed with increased accuracy and performance rate as well as enhanced capabilities. This work is submitted for publication and a draft version is available at https://arxiv.org/abs/1708.05477.

Fig. 11: An abstract representation of WedgeTail over an ISP Network. The red devices are malicious and have been compromised by Attacker. The paths shown in green represent the expected paths for a packet send through forwarding device $fd(a)$ on port p_i to forwarding device $fd(c1)$ on port p_j at Customer Network. The trajectory shown in red depicts a path the same packet took, which is not allowed as per the controller policies.

jectories are computed using Header Space Analysis (HSA) [40] framework. In order to compute the actual packet trajectories, WedgeTail relies on NetSight [36] and queries for the packet history as it traverses the network. However, when NetSight is not available (e.g. in small-sized networks), the packets may be tracked using a custom packet tracking mechanism proposed in the paper.

As shown in Figure 11, WedgeTail is composed of two main parts namely, Detection Engine and Response Engine. The Detection Engine is responsible to retrieve the actual and expected packet trajectories, create the scanning regions and implement the attack detection algorithms. Accordingly, whenever a compromised device is detected, the Response Engine submits policies to the controller to protect the network.

WedgeTail has been tested under simulated environments and with a network composed of more than 300 forwarding devices and 630,000 trajectories it has been capable of locating malicious forwarding devices and identify their specific malicious action in about 90 minutes. For a relatively large network detection of such malicious entity hidden in lowest layer of network infrastructure without defining any policies or manual intervention is quite satisfactory.

5.3 Limitations of SPHINX and WedgeTail

We argue the following three factors as the main limitation of SPHINX. First, the system does not tolerate Byzantine forwarding faults. In other words, it does not assume malicious forwarding device may behave arbitrarily and therefore, SPHINX is not designed to detect the specific malicious actions performed such as packet drop, fabrication and delay. Second, the detection mechanism mainly relies on the policies defined by its administrator. In fact, the flow-graph compo-

nent does not validate forwarding device actions against the controller policies but compared to their behavior through time. Hence, if the forwarding device(s) has been malicious since day one it will never be detected or when there are radical network configuration changes SPHINX will have large false positives. Third, SPHINX does not include a scanning regime and has no prioritization when inspecting the data plane for threats. Arguably, an important factor required for optimizing the detection performance in this context. Last but not least, SPHINX requires the majority of forwarding devices to be trustworthy. Although this assumption is realistic, a better solution solution will have to be independent of it.

Essentially, WedgeTail is built on top of the limitations its developers had identified in SPHINX. WedgeTail relies on network snapshots to compute the expected packet behaviors. However, maintaining valid snapshots of large real-world networks may lead to major scalability and performance limitations. Moreover, WedgeTail's accuracy in detecting attacks over different use-cases and scenarios such as virtualization and VM migrations has not yet been evaluated. Finally, WedgeTail's compatibility with distributed controllers has not been addressed and given the shift of the community to such controllers this may be a major barrier in its adoption.

6 Future Research Directions

As discussed in §4, SDN data plane security research has been relatively dormant with few proposals emerging only recently. Given the critical role of data plane in the implementation of network policies (see §2.1), extreme value for attackers (see §4.1), increased opportunities for attackers to target them (see §3.2) and ongoing evolvement (see §4.2), we regard data plane se-

curity as a major challenge for a secure SDN network. Specifically, the fact that compromised forwarding devices can potentially take down an entire SDN system further highlights the importance of the current oversight.

We predict, however, this status will radically change with the increased adoption of SDNs. Specifically, given the potential attacks that a compromised forwarding device may deliver against different technologies that are imported to SDNs such as Software-Defined Clouds [26], we perceive the importance of SDN data plane security to be much more of a focus.

As discussed in §5, the existing solutions suffer from limitations that hinder their real-world application. We expect future research to propose more advanced and practical protection mechanisms on the ground of existing limitations. Specifically, none of the existing solutions have any consideration for the most recent proposals including stateful data planes. Hence, one of the immediate future required extensions is to evaluate the applicability of existing solution for stateful forwarding devices and propose solutions to ensure if any such device is fully, or partially, compromised it can be detected. Another direction of research that we expect to be very active in the future is designing secure hardware for SDN-enabled forwarding devices based the latest software advances and requirements.

Finally, similar to [62], we believe SDN capabilities provide the required ground to develop solutions that secure this technology independent of the underlying software, hardware and protocol. In fact, we believe this is the most reliable approach towards ensuring secure deployment of this open, dynamic and evolving technology.

7 Summary and Conclusion

In this paper, we presented a taxonomy of attacks against Software-Defined Networks based on their scope and impacts. Specifically, we discussed how an adversary can leverage vulnerabilities of different SDN components to target network policies, their enforcement, and implementation. Moreover, we argued the importance of security SDN data planes, surveyed the latest advances in data plane development and analyzed the security implications of stateful data planes. Thereafter, we established a set of requirements for a working solution and reviewed the existing solutions with respect to these requirements. Finally, we provided a set of suggestions for future research directions, where we hope to attract researchers' attention to a relatively dormant yet critical aspect of securing SDNs.

References

1. OpenFlow Switch Specification 1.5. 1(Protocol version 0x06), December, 2014.
2. Y. Yiakoumis, J. Schulz-Zander, and J. Zhu, "Pantou: OpenFlow 1.0 for OpenWRT". http://www.openflow.org/wk/index.php/Open_Flow1.0_forOpenWRT, 2011.
3. Nippon Telegraph and Telephone Corporation, RYU network operating system. http://osrg.github.com/ryu, 2012.
4. Big Switch Networks, Project Floodlight. http://www.projectfloodlight.org, 2013.
5. Chinese hackers who breached Google gained access to sensitive data, U.S. officials say. https://goo.gl/QrP2iV, 2013.
6. Open vSwitch. http://vswitch.org, 2013.
7. OpenStack and network virtualization. http://blogs.vmware.com/vmware/2013/04/openstack-and-network-virtualization.html, 2013.
8. Pica8 Open Networking, Pica8?s OS for Open Switches. http://www.pica8.org/open-switching/open-switching-overview.php, 2013.
9. VMware's network virtualization poses huge threat to data center switch fabric vendors. https://goo.gl/T2qDkL, 2013.
10. A. Shang, J. Liao, and L. Du, Pica8 Xorplus. http://sourceforge.net/projects/xorplus, 2014.
11. NIST: CVE-2014-9295 Detail. https://nvd.nist.gov/vuln/detail/CVE-2014-9295, 2014.
12. Photos of an NSA upgrade factory show Cisco router getting implant. https://goo.gl/KNH6gD, 2014.
13. Snowden: The NSA planted backdoors in Cisco products, Infoworld. https://goo.gl/fZMxkn, 2014.
14. NSA Preps America for Future Battle, Spiegel. https://goo.gl/PXMXeG, 2015.
15. SYNful Knock - A Cisco router implant - Part I. https://fireeye.com/blog/threat-research/2015/09/synful_knock_-_acis.html, 2015.
16. NIST: CVE-2014-9295 Detail. https://nvd.nist.gov/vuln/detail/CVE-2014-9295,

2017.

17. Vault 7: CIA Hacking Tools Revealed. https://wikileaks.org/ciav7p1, 2017.

18. A. Akhunzada, A. Gani, N. B. Anuar, A. Abdelaziz, M. K. Khan, A. Hayat, and S. U. Khan. Secure and dependable software defined networks. *Journal of Network and Computer Applications*, 61:199–221, 2016.

19. S. T. Ali, V. Sivaraman, A. Radford, and S. Jha. A survey of securing networks using software defined networking. *IEEE transactions on reliability*, 64(3):1086–1097, 2015.

20. M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker. Snap: Stateful network-wide abstractions for packet processing. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 29–43. ACM, 2016.

21. F. Assolini. The tale of one thousand and one dsl modems, kaspersky lab, 2012.

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

23. G. Bianchi, M. Bonola, A. Capone, and C. Cascone. Openstate: programming platform-independent stateful openflow applications inside the switch. *ACM SIGCOMM Computer Communication Review*, 44(2):44–51, 2014.

24. M. A. Bishop. *The art and science of computer security*. Addison-Wesley Longman Publishing Co., Inc., 2002.

25. P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 99–110. ACM, 2013.

26. R. Buyya, R. N. Calheiros, J. Son, A. V. Dastjerdi, and Y. Yoon. Software-defined cloud computing: Architectural elements and open challenges. In *Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on*, pages 1–12. IEEE, 2014.

27. D. Chasaki and T. Wolf. Attacks and defenses in the data plane of networks. *IEEE Transactions on dependable and secure computing*, 9(6):798–810, 2012.

28. T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti. A survey on the security of stateful sdn data planes. *IEEE Communications Surveys & Tutorials*, 2017.

29. M. Dhawan, R. Poddar, K. Mahajan, and V. Mann. Sphinx: Detecting security attacks in software-defined networks. In *NDSS*, 2015.

30. X. Dong, H. Lin, R. Tan, R. K. Iyer, and Z. Kalbarczyk. Software-defined networking for smart grid resilience: Opportunities and challenges. In *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, CPSS '15, pages 61–68, New York, NY, USA, 2015. ACM.

31. D. Erickson. The beacon openflow controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 13–18. ACM, 2013.

32. A. Feldmann, P. Heyder, M. Kreutzer, S. Schmid, J.-P. Seifert, H. Shulman, K. Thimmaraju, M. Waidner, and J. Sieberg. Netco: Reliable routing with unreliable routers. In *Dependable Systems and Networks Workshop, 2016 46th Annual IEEE/IFIP International Conference on*, pages 128–135. IEEE, 2016.

33. A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi. Participatory networking: An api for application control of sdns. In *ACM SIGCOMM computer communication review*, volume 43, pages 327–338. ACM, 2013.

34. O. N. Fundation. The benefits of multiple flow tables and ttps. Technical report, ONF Technical Report, 2015.

35. P. Goransson, C. Black, and T. Culver. *Software defined networks: a comprehensive approach*. Morgan Kaufmann, 2016.

36. N. Handigol, B. Heller, V. Jeyakumar, D. Mazières, and N. McKeown. I know what your packet did last hop: Using packet histories to troubleshoot networks. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 71–85, 2014.

37. B. Heller. Openflow switch specification, version 1.0. 0. *Wire. December*, 2009.

38. N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford. Hula: Scalable load balancing using programmable data planes. In *Proceedings of the Symposium on SDN Research*, page 10. ACM, 2016.

39. P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte. Real time network policy checking using header space analysis. In *Presented as part of the 10th USENIX Symposium*

on Networked Systems Design and Implementation (NSDI 13), pages 99–111, 2013.

40. P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 113–126, 2012.

41. A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey. Veriflow: Verifying network-wide invariants in real time. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 15–27, 2013.

42. R. Klöti, V. Kotronis, and P. Smith. Openflow: A security analysis. In *21st IEEE International Conference on Network Protocols (ICNP)*, pages 1–6. IEEE, 2013.

43. 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 *OSDI*, volume 10, pages 1–6, 2010.

44. D. Kreutz, F. Ramos, and P. Verissimo. Towards secure and dependable software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 55–60. ACM, 2013.

45. D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

46. S. Lee, T. Wong, and H. S. Kim. Secure split assignment trajectory sampling: A malicious router detection system. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pages 333–342. IEEE, 2006.

47. F. Lindner. Cisco ios router exploitation. *Black Hat USA*, 2009.

48. H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. Godfrey, and S. T. King. Debugging the data plane with anteater. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 290–301. ACM, 2011.

49. N. McKeown. Software-defined networking. *INFOCOM keynote talk*, 17(2):30–32, 2009.

50. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

51. J. Medved, R. Varga, A. Tkacik, and K. Gray. Opendaylight: Towards a model-driven sdn controller architecture. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, pages 1–6. IEEE, 2014.

52. C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker, et al. Composing software defined networks. In *NSDI*, volume 13, pages 1–13, 2013.

53. M. Moshref, A. Bhargava, A. Gupta, M. Yu, and R. Govindan. Flow-level state transition as a new switch primitive for sdn. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 61–66. ACM, 2014.

54. T. D. Nadeau and K. Gray. *SDN: Software Defined Networks: An Authoritative Review of Network Programmability Technologies.* " O'Reilly Media, Inc.", 2013.

55. E. Ng, Z. Cai, and A. Cox. Maestro: A system for scalable openflow control. *Rice University, Houston, TX, USA, TSEN Maestro-Techn. Rep, TR10-08*, 2010.

56. M. Nobakht, V. Sivaraman, and R. Boreli. A host-based intrusion detection and mitigation framework for smart home iot using openflow. In *Availability, Reliability and Security (ARES), 2016 11th International Conference on*, pages 147–156. IEEE, 2016.

57. N. Pelekis, I. Kopanakis, C. Panagiotakis, and Y. Theodoridis. Unsupervised trajectory sampling. In *Machine learning and knowledge discovery in databases*, pages 17–33. Springer, 2010.

58. B. Pfaff and B. Davie. The Open vSwitch database management protocol, Internet Engineering Task Force, RFC 7047 (Informational). http://vswitch.org, 2013.

59. K. Phemius, M. Bouet, and J. Leguay. Disco: Distributed multi-domain sdn controllers. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–4. IEEE, 2014.

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

61. S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao. Are we ready for sdn? implementation challenges for software-defined networks. *IEEE Communications Magazine*, 51(7):36–43, 2013.

62. A. Shaghaghi, M. A. Kaafar, and S. Jha. Wedgetail: An intrusion prevention system for the data plane of software defined networks. In *Proceedings of the 2017 ACM on Asia Conference*

on Computer and Communications Security, ASIA CCS '17, pages 849–861, New York, NY, USA, 2017. ACM.

63. A. Shaghaghi, M. A. Kaafar, S. Scott-Hayward, S. S. Kanhere, and S. Jha. Towards policy enforcement point of (peps). In *Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on*, pages 50–55. IEEE, 2016.

64. M. Smith, M. Dvorkin, Y. Laribi, V. Pandey, P. Garg, and N. Weidenbacher. Opflex control protocol. *IETF*, 2014.

65. H. Song. Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 127–132. ACM, 2013.

66. K. Thimmaraju, L. Schiff, and S. Schmid. Outsmarting network security with SDN teleportation. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*, pages 563–578, 2017.

67. A. Tootoonchian and Y. Ganjali. Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, pages 3–3, 2010.

68. A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. On controller performance in software-defined networks. *Hot-ICE*, 12:1–6, 2012.

69. T. Tsou, H. Yin, H. Xie, and D. Lopez. Use cases for alto with software defined networks. 2012.

70. H. Yin, H. Xie, T. Tsou, D. Lopez, P. Aranda, and R. Sidi. Sdni: A message exchange protocol for software defined networks (sdns) across multiple domains. *IETF draft, work in progress*, 2012.

71. S. Zhu, J. Bi, C. Sun, C. Wu, and H. Hu. Sdpa: Enhancing stateful forwarding for software-defined networking. In *Network Protocols (ICNP), 2015 IEEE 23rd International Conference on*, pages 323–333. IEEE, 2015.

Arash Shaghaghi is a PhD Candidate at UNSW Sydney where he is also affiliated with Data61, CSIRO. Previously, he completed an MSc in Information Security at University College London (UCL) and a BSc in Information Technology at Heriot-Watt University of Scotland. Arash has been consistently ranked as top student in his studies and received multiple awards and scholarships in the past few years. Arash research interests include Software Defined Network (SDN), SDN Security, and Access Control Systems. Arash's cyberhome: https://www.cse.unsw.edu.au/~arashs/

Mohamed Ali Kaafar is the group leader of the networks research group at CSIRO Data61, Australia. He was previously a research leader NICTA, Australia, and a research scientist at INRIA RhoneAlpes, France. His research interests include cyber Security, privacy preserving technologies, and networks measurement.

Rajkumar Buyya a is a Redmond Barry Distinguished Professor of Computer Science and Software Engineering; and Director of the Cloud Computing and Distributed Systems Laboratory at the University of Melbourne, Australia. He is also serving as the founding CEO of Manjrasoft Pty Ltd., a spin-off company of the University, commercializing its innovations in Grid and Cloud Computing. He has authored over 525 publications and seven text books including 'Mastering Cloud Computing' published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. He is one of the highest cited authors in computer science and software engineering worldwide. Recently, Dr. Buyya is recognized as '2016 Web of Science Highly Cited Researcher' by Thomson Reuters. For further information on Dr. Buyya, please visit his cyberhome: http://www.buyya.com.

Sanjay Jha received the Ph.D. degree from the University of Technology, Sydney, Australia. He is currently a Professor and the Director of the Cyber Security and Privacy Laboratory and the Head of the Networked Systems and Security Group, School of Computer Science and Engineering, the UNSW Australia. He has authored or co-authored over 100 articles in high-quality journals and conferences. His research activities include a wide range of topics in wireless networking and security. He has served on program committees of several conferences. He is a Senior Member of the IEEE Computer Society.