A Survey on the Verification of Adversarial Data Planes in Software-Defined Networks

Black, C., & Scott-Hayward, S. (2021). A Survey on the Verification of Adversarial Data Planes in Software-Defined Networks. In *2021 ACM International Workshop on Software Defined Networks & Network Function Virtualization Security: SDN-NFV Sec'21* ACM.

Published in:

2021 ACM International Workshop on Software Defined Networks & Network Function Virtualization Security

Document Version:

Peer reviewed version

Queen's University Belfast - Research Portal:

[Link to publication record in Queen's University Belfast Research Portal](#)

Publisher rights

Copyright 2021 ACM.

This work is made available online in accordance with the publisher's policies. Please refer to any applicable terms of use of the publisher.

General rights

Copyright for the publications made accessible via the Queen's University Belfast Research Portal is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy

The Research Portal is Queen's institutional repository that provides access to Queen's research output. Every effort has been made to ensure that content in the Research Portal does not infringe any person's rights, or applicable UK laws. If you discover content in the Research Portal that you believe breaches copyright or violates any law, please contact openaccess@qub.ac.uk.

A Survey on the Verification of Adversarial Data Planes in Software-Defined Networks

Conor Black

Centre for Secure Information Technologies
Queen's University Belfast
Belfast, N. Ireland
cblack39@qub.ac.uk

Sandra Scott-Hayward

Centre for Secure Information Technologies
Queen's University Belfast
Belfast, N. Ireland
s.scott-hayward@qub.ac.uk

ABSTRACT

As network policies are becoming increasingly nuanced and complex, so too are the mechanisms required to ensure that the network is functioning as intended. In particular, since the dawn of software-defined networking and the shift towards high-level descriptions of intended network policy, traditional tools such as *ping* and *traceroute* have been insufficient to test that complex data plane configurations have been correctly implemented. As a result, novel data plane verification solutions have been proposed that use formal methods to ensure that network policies are adhered to and that the data plane is free of bugs. While the number of these verification solutions continues to grow, only a few are equipped to verify the data plane when a malicious adversary is present. As research continues to expand the remit of data plane functionality, these solutions may become key to securing an increasingly valuable attack target. In this survey, we review the work that has been dedicated to preventing and detecting attacks on data planes in software-defined networks and discuss some of the unsolved problems in this field that must be addressed in future adversarial verification solutions.

KEYWORDS

data plane verification, adversarial data plane, data plane security, software-defined networking (SDN), SDN security

1 INTRODUCTION

The centralised control of network forwarding devices made possible by software-defined networking (SDN) has revolutionised network application development by enabling a shift towards logically centralised applications such as load balancers [43] and firewalls [19] with network-wide visibility. The implementation of network-wide policies (which often required individual configuration of network devices) can now be described centrally at a high level (e.g. by using a network programming language such as NetKAT [1]) and compiled down to data plane (DP) device-level implementations.

These programming languages not only make it straightforward for a network operator to define network-wide behaviour, but the sound equational theory underpinning languages such as NetKAT allow the existence of certain network properties to be proven mathematically. The network operator can prove, for example, that no forwarding loops exist in the network, before the configuration changes are applied. In addition, these languages allow for several programs to be composed together, before their sum effect is checked for correctness properties. This has the potential to significantly reduce the

impact of programs undermining each other's goals by installing contradictory rules in the DP, a recognised problem in SDN [36].

While invariant checking before deployment of network programs is invaluable, bugs in compilers and switch software have the potential to cause errors in forwarding behaviour, even when the network program is correct. This problem has led to the development of runtime DP testing solutions that complement compile-time correctness checking by observing the network's actual behaviour. The format of these solutions varies. For example, BUZZ [16] automatically creates test traffic based upon the intended network policies and runs this test traffic through the network to detect discrepancies between expected and observed traffic. Another method used in DP testing is the collection of so-called postcards, which report the exact path that packets have traversed through the network and the flow rules that each packet has matched along the way. NetSight [18] is an example of a system that does this.

Assuming the effectiveness of the chosen solutions, the combination of compile-time and runtime checking mechanisms should be able to detect a wide range of bugs that may cause DP errors. However, the two main approaches (1) expecting consistent behaviour in the case of packet probes, and (2) trusting the DP devices to self-report the rules that they have applied to packets in the case of DP postcards, rely on an implicit trust of the DP to behave faithfully. As a result, many runtime DP verification solutions are vulnerable to the threat of an adversarial DP, where one or more of the DP devices are under the control of a malicious actor, who not only wishes to alter the network's forwarding behaviour, but actively attempts to conceal these policy breaches from the network operator. To date, a relatively small number of defence mechanisms have been proposed that seek to detect DP forwarding anomalies without assuming the trustworthiness of any individual DP device. We refer to these solutions as *adversarial data plane verification* (ADPV).

While the techniques used by state-of-the-art ADPV solutions vary, the different solutions are often similar in scope, with more recent solutions failing to improve substantially on either the scalability or security guarantees of previous solutions. In addition, with the growing research interest in programmable DPs and the growing array of applications that are being designed to take advantage of DP statefulness, the motivations and capabilities of an adversary with control of a modern DP device are changing rapidly. More recent ADPV solutions have failed to fully reflect this trend.

With this in mind, the purpose of this work is to summarise the state-of-the-art in ADPV, with a focus on contrasting the practicality and security guarantees offered by each solution. In addition, we aim to identify the major open questions within the field of ADPV, and discuss how these may be addressed in future work. The structure

of the paper is as follows: next, we provide a background of current SDN verification solutions in Section 2, before discussing previously identified DP security risks in Section 3. Section 4 presents a detailed review of the state-of-the-art in ADPV and Section 5 identifies the outstanding issues with ADPV mechanisms. Finally, we offer concluding remarks in Section 6.

2 SDN VERIFICATION

The term SDN was coined at Stanford following the publication of the Ethane paper [9], in which the authors demonstrated how the management of the campus network could be dramatically simplified by replacing the control planes (CPs) found on each network switch with a single, logically centralised CP running on a Linux server. This abstraction encouraged a rapid increase in the number of performance and security-enhancing applications, which were able to avoid the constraints of distributed control in traditional networks by locating their more complex, stateful functionality at the CP.

While the CP greatly simplified application development by providing a global view of the DP at a centralised controller, the implementation of DP policy was still complicated by the need for application developers to manage flow rules on individual switches. In response to this, the idea of defining network policy using a domain-specific language, with the device-level implementations abstracted away by compilation of these programs to device-specific implementations (as argued for in [22]), soon took hold. The languages (such as NetKAT [1] and SNAP [2]) that were developed for this purpose allowed for logical reasoning about properties of the network that should always hold true, known as invariants. These invariants include properties such as loop-freedom (that packets don't get stuck in an endless DP forwarding loop) and reachability (that packets from node A can always reach node B).

These verification-friendly programming languages introduced assurance that contradictory policies were never installed in the DP. However, by design, these mechanisms cannot guard against compiler and switch-level bugs, which can cause incorrect DP behaviour despite provably correct policy. To guard against this, DP testing and monitoring mechanisms have been developed as a complement to CP verification. These take a variety of forms, such as the sending of generated probe packets through DP devices (e.g. BUZZ [16]) and the monitoring of packet histories to ensure that they correspond to policy (e.g. NetSight [18]). The verification of both DPs and CPs using formal methods is surveyed extensively in [28].

Early research work on SDN almost universally assumed the use of the OpenFlow (OF) protocol [30] as the standard for communication between the DP and CP. Some verification solutions, such as Veriflow [24], relied specifically on the monitoring of OF messages to extract the network configuration. OF proved popular due to its compatibility with existing network switches, whose capabilities were immutably defined in packet processing hardware such as ASICs. However, as new paradigms such as the Reconfigurable Match Table (RMT) switch architecture [6] showed that switch chips could be made programmable without sacrificing performance, interest in programmable data planes grew.

The P4 language [5] has been adopted as the de facto standard for programming this new generation of programmable switches. P4 offers programmers the ability to define custom protocol parsers, edit

arbitrary packet header fields and define stateful network functions within the DP. This new DP representation has radically changed the landscape for DP verification, as flow tables have been replaced by programs that specify precisely the intended behaviour of the DP device. This has given rise to program verification solutions such as p4v [29], which feeds a first order logic expression of the P4 program into a theorem prover to find inputs that would violate its intended properties. P4-specific DP testing and monitoring frameworks have also been developed, such as p4pktgen [32], which generates probe packets for P4 programs and [25], which annotates P4 programs to efficiently produce postcards with encoded program paths.

3 DATA PLANE SECURITY ISSUES

While SDN has facilitated better DP testing and verification than was practical in traditional networks, it has not eradicated attacks on network DPs. Indeed, the SDN architecture itself has introduced some novel attacks involving the DP. [21] describes how the centralised CP is vulnerable to denial-of-service (DoS) attacks caused by an excessive number of *PACKET_IN* messages from the DP. In [8], a more novel DoS technique is described, where hosts connected to the DP saturate the specific DP links that are also used to transport control traffic. In addition, lack of effective authentication mechanisms (particularly in early SDN implementations) have allowed for host location hijacking attacks [35] and man-in-the-middle attacks between the CP and DP [4]. While still relatively new, some work has focused on vulnerabilities associated specifically with P4 networks. For example, [14] analyses how bugs in P4 programs can be exploited on different switch architectures for purposes such as DoS and information disclosure. [36] surveys many of the threats brought by the SDN architecture, along with solutions that have been proposed to combat these issues.

While the described attacks may all be launched by a malicious host connected to the network DP, other works have recognised the potential threats posed by an attacker that has directly compromised a DP device. [7] describes how a compromised switch can be used to launch a middlebox bypass attack, where the switch appends falsified tags to DP packets to falsely declare that they have passed through a particular middlebox. [11] describes a number of possible attacks that may be launched from a compromised OF switch including packet injection and duplication.

Despite the potential range of attacks that can be launched from a compromised SDN switch, there have been few works showcasing direct switch exploitation. [41] is a prominent example of one such attack. In this attack, the authors show that by fuzzing the OvS software switch, they were able to find two vulnerabilities that they could exploit to obtain a remote shell on the switch. Pickett [33] showcased how the culmination of several vulnerabilities in white-box switches, e.g. lack of secure boot, could be used by an attacker to stay persistent in one of these SDN-ready switches after gaining access via a compromised network administrator.

4 ADVERSARIAL DP VERIFICATION

The key idea behind ADPV is that an attacker's goal is to modify the network behaviour by altering the forwarding behaviour of the switches, and that by examining actual network behaviour against a model of expected behaviour, it is possible to discover the presence

of an attacker-controlled DP device. The distinction between non-adversarial and adversarial DP verification is important, given that our above definition could apply equally to both finding an attacker and finding bugs in the network. The distinguishing feature of ADPV is the added assumption that an attacker may attempt to hide their violations of policy, for example by falsifying packet counters when requested by the CP. This is in contrast to non-adversarial verification where buggy switches are expected to faithfully report their actual behaviour. To account for this, ADPV solutions tend to treat all data originating from the DP with suspicion, employing techniques such as redundancy or cryptography to increase confidence in the legitimacy of the data¹. We refer the reader to [28] for a survey of the state-of-the-art in non-adversarial DP verification.

It is also important to distinguish between ADPV and mechanisms that use DP switches to detect compromised hosts directly attached to these switches. A prominent example of such a mechanism is presented in [3], where a system of trusted OF switches and middle-boxes are used to collect forensic records about the behaviour of the end hosts on the network. In ADPV, these switches are considered attack targets and cannot be trusted.

We also note that other works that we have not included in our discussion have considered the impact of compromised switches in SDNs, but are interested in more restricted threat models than the general capabilities of compromised switches that are considered here. The most prominent example of this is the consideration of middlebox bypass attacks in [7] and [39], where an attacker in control of a compromised switch adds a tag to a packet to indicate falsely that the packet has been processed by a middlebox. These solutions prevent this attack by trusting the middleboxes themselves to verify packet tags cryptographically. While these solutions provide a degree of path confirmation and prevent illegal modification of packet tags, they are not intended to prevent more general violations of network policy, which the solutions we focus on attempt to do.

Table 1 summarises the current state of the art in ADPV by both attack coverage and detection techniques used².

4.1 Verification by Trusted Components

In NetCo [17], the authors attempt to reduce the potential impact of a malicious DP switch by deploying redundant switches to detect anomalies. Each switch/router is replaced by a trusted hub, k switches and a compare function. The hub forwards received packets to all k switches, which perform identical operations on the packet, before sending it to the compare function, which makes a majority decision based upon the outputs of the switches. In this way, malicious behaviour can be detected and prevented as long as the number of compromised switches per hub remains below $k/2$. While the concept of NetCo is sound in its use of redundancy to both detect and prevent attacks, it relies on the hubs and compare devices being uncompromisable, and on limited collusion between switches connected to the same hub.

¹In the case of probing, this distinction is less clear, as different solutions use probing to test for both bugs and attacks. We survey here only solutions with the goal of attack detection, to enable fair critique of their attack coverage.

²In Table 1, Illegal Packet Modification refers to the ability to distinguish legal modification (in line with network policy) from illegal modification and thus does not include solutions that regard any packet modification as a potential attack.

4.2 Verification through Cryptography

The concept behind the use of cryptography in DP verification is that by arming each switch with a secret key that is inaccessible to other switches, it is possible for the CP to verify which switches have truly been traversed by a particular packet. The first solution to leverage this idea in the context of ADPV was SDNSec [34], which uses a combination of source routing and Message Authentication Codes (MACs) to verify packet traversal. MAC-protected forwarding entries are embedded into each packet at the ingress switch, before the packet is forwarded to the core switches in the network. These are reduced to two operations: verifying that the applicable flow rules are genuine, and forwarding the packets to the ports indicated by the flow rules, if so. The egress switches are then responsible for forwarding each packet to the correct end hosts.

REV [45] does not use source routing like SDNSec, but rather uses a compressive MAC that is continuously updated by each switch in a packet's traversal. This is done through the use of a secret key assigned to each switch by the SDN controller. Once a packet reaches the egress switch, the final MAC is reported to the controller, which then verifies its correctness. The use of a compressive MAC in this solution is its main innovation, as it allows for the traversals of several packets to be merged into a single report to the controller, which reduces the volume of traffic destined for the CP.

The major benefit of these solutions is their coverage of all packets in the network, allowing them to detect low volume attacks that may be missed by checking per-flow counters. However, performing cryptographic operations on every packet and increasing packet sizes with tags can significantly limit throughput. In addition, these solutions do not account for network policies that may legitimately edit packet headers, and so would erroneously report anomalies for common network operations such as network address translation.

4.3 Verification using Probe Packets

Chi et al. [11] propose an OF-based mechanism to detect switch misbehaviour using probe packets. In this approach, switches forward controller-injected probe packets matching a pre-identified, suspiciously underutilised flow rule back to the controller. If the packet is not received at the CP, or received from the wrong switch, the presence of an attacker is assumed.

Chiu et al. [12] use crafted probe packets to trigger multiple match rules at once, with the intention of significantly reducing the number of probe packets that need to be sent to cover all installed flow rules. The major limitation of these solutions is that they cover only pre-installed, legitimate flow rules and thus do not cover attacks where an attacker adds their own additional rules.

Finally, Chao et al. [10] propose a detection mechanism that combines probing with the checking of counter values at each switch (similar to [13]), and the encryption of packets traversing a suspect switch. This technique is intended to ensure that an attacker cannot tailor their attacks to manipulate packets with a particular feature, e.g. a particular DNS response. While this work combines detection mechanisms and analyses different attacks, the individual detection mechanisms are seen in isolation in previous works.

Table 1: Comparison of Attack Coverage & Techniques of ADPV Solutions

	NetCo [17]	SDNSec [34]	REV [45]	Chi et al. [11]	Chiu et al. [12]	Chao et al. [10]	FlowMon [20]	FOCES [46]	SPHINX [13]	White Rabbit [38]	Preacher [40]	Wedge Tail [37]	DynaPFV [27]
Attack Coverage													
Inter-Switch Collusion	✓	✓	✓	×	×	✓	×	×	✓	✓	✓	✓	✓
Drops & Redirects	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Packet Injection	✓	✓	✓	×	×	✓	×	✓	✓	✓	✓	✓	✓
Illegal Packet Modification	✓	×	×	×	×	×	×	×	×	×	✓	✓	×
Edge Switch Attacks	✓	×	×	×	×	×	×	×	×	×	×	×	×
Stateful Attacks	✓	×	×	×	×	×	×	×	×	×	×	×	×
Detection Techniques													
Trusted Hardware	✓	×	×	×	×	×	×	×	×	×	×	×	×
Cryptographic Operations	×	✓	✓	×	×	✓	×	×	×	×	×	×	×
Probe Packets	×	×	×	✓	✓	✓	×	×	×	×	×	×	×
Distributed Statistics	×	×	×	×	×	✓	✓	✓	✓	✓	×	×	✓
Packet/Hash Collection	×	×	×	×	×	×	×	×	×	×	✓	✓	✓

4.4 Verification by Distributed Statistics

The most popular approach to ADPV to date is the use of distributed statistics collection across multiple switches. The rationale is that packets of a particular flow should follow a defined path through the network and thus the number of packets seen at each switch should be consistent with the correct path being taken by each of these packets. Otherwise, an attack may be taking place.

FlowMon [20] uses packet counters at each switch port to detect packet dropping and redirection attacks. However, the impact of false statistics reporting is not considered and the use of crude per-port packet counts to detect packet drops does not consider attacks where a device swaps equal numbers of packets between flow rules, maintaining correct packet counters.

This packet swapping attack was subsequently addressed by FOCES [46], which monitors per-flow statistics rather than the per-port statistics, which rendered the attack possible in the presence of FlowMon. In addition, FlowMon uses a network slicing algorithm to improve the performance of statistics processing at the controller.

SPHINX [13] more broadly considers the potential actions of a malicious switch in an OF network; attempting to detect packet dropping, redirection and injection attacks, and attacks on the CP such as the creation of fake links in the network. SPHINX does this by intercepting OF messages between the data and CPs and checking them for anomalies such as requests that indicate Address Resolution Protocol (ARP) spoofing attacks. This is augmented by a byte consistency check, which checks that counter values are preserved in the same fashion as FlowMon.

The authors of WhiteRabbit [38] improve upon the accuracy of byte consistency checks in SPHINX by proposing a time synchronisation protocol that ensures delays in statistics collection at the controller are accounted for in consistency calculations. However, WhiteRabbit and other counter-based solutions rely on quite crude estimates of compromise, relying on the breach of threshold packet counts by attacks and are quite heavily coupled to the OF protocol.

4.5 Verification by Packet or Hash Collection

Verification by packet or hash collection improves upon the effectiveness of statistics collection solutions by gathering and comparing actual data plane packets rather than just statistics about them. This allows for detection of more advanced attacks, such as payload modification. Preacher [40] collects hashed DP packets by assigning random hash values to each switch in the network and instructing switches to forward any packets with matching hashes to the CP. Preacher then compares expected and received hash values, with any discrepancy indicating a potential attack. Given that the memory of each switch is insufficient to store all possible hashes, each switch's hash list is periodically rotated to enable eventual coverage of all hash values. While Preacher is one of the more complete solutions, its major disadvantage is its probabilistic nature, requiring hashes to be located at the right switches at a given time to detect an attack.

WedgeTail [37] operates similarly to Preacher in that it processes packet hashes (or alternatively packet histories provided by NetSight) to detect anomalous forwarding. WedgeTail, however, is light on the details of the overhead of its packet collection mechanism and

claims to be able to detect attacks even when every switch in the network has been compromised, a clear impossibility.

Finally, DynaPFV [27] combines the dynamic collection of full packets at ingress and egress switches with analysis of packet counters. When a *PACKET_IN* is received for a new flow, the verification engine adds flow rules, which instruct the ingress and egress switches to randomly clone a subset of packets of this flow to the control plane. Once these are received, the verifier compares the packets received at ingress and egress to ensure that they are equal. If they are, the verifier reduces the probability of further collections by changing a weight variable, as trust in the network builds. Once each flow rule expires, the verification then collects counter values rather than packets for verification. While this solution is innovative in its ability to reduce overhead whilst maximising detection probability, it does not consider networks where modifications can be legally made to packets, instead regarding all modifications as potential attacks.

5 DISCUSSION

5.1 Performance Impact of Verification

A potential objection to the adoption of ADPV mechanisms within a network is the processing overhead associated with verifying that observed DP behaviour corresponds to network policy. This can generally be divided into two categories: CP overhead and DP overhead, which will be discussed separately in the following subsections.

5.1.1 CP Overhead. CP overhead is generally the result of two activities: calculation of expected network behaviour and processing of reports received from the DP. The impact of these is not always investigated thoroughly, but some solutions do provide some insight. For example, the maximum processing rate achieved by Preacher is just below 3000 samples per second, while FOCES reports computation times of over 5000 seconds when 12,000 flows are used (however, this is reduced by 80% with slicing). This overhead in statistics gathering techniques can be a security issue, particularly when the number of reports sent to the CP is driven by the DP, as end hosts overwhelm the controller with statistics reports. The controller does have the ability to control the sampling rate to reduce controller burden by, for example, reducing the percentage of packets sampled. However, this would reduce the probability of attack detection.

Recalculation of expected network behaviour is critical to the correctness of all ADPV solutions, as DP activity must be verified against the policy that applied at the time of the traversal to avoid false positives. Despite this, many solutions don't effectively analyse the performance impact of rule updates on their solutions. For example, Chiu et al. [12] rigorously investigate the impact of different network topologies and flow rules per switch on their coverage time, finding that switches with over 100 flow rules can require 1 minute to probe exhaustively. It is not clear how the alteration of forwarding rules by the CP over the course of this time may impact the flow coverage, which is carefully calibrated to reduce the required number of probe packets. SPHINX attempts to ensure that it keeps track of the current network state by intercepting *OF_FLOW_MOD* messages before they reach the DP. However, this degrades DP throughput due to the resultant delay in installing new flow rules.

5.1.2 DP Overhead. Given the substantial effort by network operators to increase DP throughput, ADPV solutions that impose

significant DP overhead are unlikely to achieve widespread adoption. While some overhead is inevitable (as evidenced by Table 2, in which we present a comparison of DP overhead between the ADPV mechanisms), some solutions perform better in this regard than others. The cryptographic ADPV mechanisms, SDNSec and REV, for example, are particularly affected by DP overhead, as they both append tags to data plane packets. In addition, the lengths of these tags increase with the packet path length in SDNSec, as it embeds a forwarding rule for each DP switch in every packet.

Some verification mechanisms reduce throughput by adding additional packets to the DP, reducing the capacity for the links to handle regular packets. For example, the probing mechanism proposed by Chao et al. requires up to 25,000 packets to test a single set of flow rules. NetCo requires each packet to be duplicated multiple times and forwarded through the comparison hardware, doubling round trip time in some configurations. Finally, the additional overhead associated with packet hashing and collection solutions can have an impact on the capacity of the data plane to forward packets. This effect is directly measured in DynaPFV, which reports degradation of just over 37% if all packets are sampled. However, the authors do advocate dynamic sampling, which reduces this by over 30%. Impact on throughput is not directly measured in Preacher or WedgeTail. However, a variant of WedgeTail does rely on NetSight [18] to collect packet histories for analysis, which reports a 31% increase the volume of DP traffic if the packet histories are reported in-line.

Apart from direct performance impact on DP traffic, an additional concern with ADPV solutions is the use of scarce switch memory resources through flow rules and counters required for statistics gathering. However, the authors of FlowMon, for example, claim that the counters used by their solution would already be present as they are associated with existing flow rules. On the other hand, the probing mechanism proposed by Chiu et al. can require hundreds of additional flow rules to be installed on switches.

5.2 Probabilistic Detection

Despite the often significant performance penalties associated with ADPV mechanisms, the security guarantees they provide are probabilistic for most detected attacks. A major contributor to this for all solutions is benign packet drops, which are virtually guaranteed in all networks and indistinguishable from malicious packet drops performed by an attacker. This creates a dilemma as to whether or not to flag these drops as potential attacks. The general consensus among ADPV mechanisms has been to use threshold values for drop attacks, which allow a small number of drops to happen without raising an alarm. For counter-based statistics analysis solutions such as FlowMon, a formula for calculating this threshold is explicitly outlined by the authors, while other solutions such as Preacher simply advise disregard of low-volume drops. The authors of FOCES investigate its accuracy under different rates of packet drops, noticing that its detection capability is greatly reduced by packet loss rates of greater than 10%. REV is distinctly affected by packet loss, as it batches packets of a single flow and calculates a cumulative hash on all of them to reduce the number of report packets sent to the controller. The loss of any of these packets will render the hash incorrect, invalidating the entire batch.

Table 2: Comparison of DP Performance Overhead for Different ADPV Solutions

	NetCo [17]	SDNSec [34]	REV [45]	Chi et al. [11]	Chiu et al. [12]	Chao et al. [10]	FlowMon [20]	FOCES [46]	SPHINX [13]	White Rabbit [38]	Preacher [40]	Wedge Tail [37]	DynaPFV [27]
Affected Switches	All	All	All	Single Switch	Flow Chain	All	All	All	All	All	All	All	All
Increased Packet Size	×	✓	✓	×	×	×	×	×	×	×	×	×	×
DP Hashing/ Crypto	×	✓	✓	×	×	✓	×	×	×	×	✓	✓	×
Additional DP Traffic	✓	×	×	✓	✓	✓	×	×	×	×	×	×	×
Requires Flow Rules/Counters	×	Rules at Edge	Rules at Edge	Rules	Rules	Both	Counters	Counters	Counters	Counters	Rules	Rules	Both

The wider impact of packet drops on ADPV solutions is that they inevitably have to sacrifice detection accuracy to achieve usability. For all of the previously mentioned solutions, an attacker could subtly degrade network performance by launching low volume dropping attacks. While this may not be particularly effective when deployed at random, these attacks may be applied to packets with particular characteristics, such as those carrying the data of a specific protocol, potentially significantly degrading that service.

Another probabilistic element of ADPV that reduces detection rates is the inability of the network to simultaneously investigate all flow rules at once. This limitation is particularly prevalent in probing and packet hash gathering solutions, as attempting to probe all flows at once would create a large degree of DP traffic and simultaneously gathering all DP packets would easily overwhelm processing capacity at the CP. The probing solution proposed by Chi et al. relies on a pre-identified malicious flow rule to avoid the need to probe all packets in the network, inevitably reducing detection rates. Chiu et al. probe every flow rule, but this can require up to 2 minutes when each switch has 200 flow rules. In practice, high performance DP switches can have several times more flow rules than that [26]. Not only could this make total flow coverage infeasible, but the greater time between probes for a given flow rule also raises the possibility of a time-based attack, launched when an attacker knows their flow rule of interest is not being probed. The success of such an attack relies on efficient detection of probe packets, but this is often possible as probe packets may identify themselves, e.g. by setting a magic number in VLAN tags as proposed by Chao et al. In Preacher’s case, a very low percentage of packets are sampled by the controller at any given time (just 0.9% and 1.3% in the experiments). Although cycling ensures eventual full coverage, it remains vulnerable to the same time-based attacks as probing mechanisms.

5.3 Reliance on Trusted Components

The Achilles heel of all current ADPV mechanisms is the inability to detect attacks on certain intrinsically trusted components should they be compromised. With the exception of NetCo (which trusts the hubs and comparison hardware), it is the network’s edge switches that are trusted. The necessity of edge switch trust stems from their unique position in the network, being both the first and last switches to forward DP packets, giving them free reign to silently edit or

Figure 1: Illustration of Impact of Compromised Edge Switch on Distributed Statistics ADPV Mechanism

drop these packets. This attack is particularly evident in DynaPFV, as the verification engine relies on packets received from ingress and egress switches to differ for attacks to be detected. An attacker could easily avoid detection by DynaPFV by simply replaying the original packet to the controller while forwarding an altered version to the end host. Fig. 1 shows how a simple packet hash collection ADPV mechanism would be unable to detect such a packet dropping attack. Once the ingress switch drops the packet, it is not received by either the next hop switch or the CP. As a result, both the CP and the remaining DP switches are oblivious to the existence of the packet and thus cannot verify its correct traversal.

This problem cannot be easily overlooked, especially considering that edge switches may be the first to process packets from untrusted end hosts, who have been shown to be able to compromise switches using specially crafted packets [41]. Despite this, no current ADPV solutions offer a potential path to finding a solution and often disregard edge switch attacks for simplicity. While the most obvious solution lies in extending the reach of the network this may not always be practical and simply shifts the problem to an arguably more easily exploitable network component.

5.4 Inadequate Consideration of P4 Data Plane

No ADPV solution surveyed addresses the implications of P4 DPs on verification. This is problematic, as P4 DPs have the potential to greatly increase the required complexity of ADPV solutions, largely due to the support for stateful packet processing in some

P4 programmable switch chips, such as Intel’s Tofino [42]. This statefulness is provided in P4 by array-like data structures called registers, which can store arbitrary data beyond the processing of a single packet. This functionality has had a major impact on the development of applications for SDNs with P4 DPs, as it allows for previously centralised stateful processing to be relocated onto the DP switches and carried out at line rate. This includes the ability to fully offload stateful security policies, as done by the authors of [23], or to partially offload statistics processing as advocated by [31].

These novel DP applications have opened up an entirely new class of attack on a compromised switch, whereby state variables are manipulated to alter stateful DP forwarding behaviour. For example, for applications that require statistics processing at the DP such as [31], the attacker could subtly alter the packet ratios reported to the machine learning engine, potentially causing it to overlook an ongoing DoS attack.

The ability of current ADPV solutions to detect these subtle attacks is weak, with NetCo the only mechanism that could detect them in its current form. It has this ability as it simply replicates the required operations across several switches, which should produce the same result regardless of whether these operations are stateful or not. For all other solutions, some form of extension is required to adapt to stateful forwarding policies. Fig. 2 shows how ADPV solutions that rely on the collection of distributed statistics are impacted by a stateful attack. In this case, the middle switch has a crude stateful policy which causes it to drop packets if the throughput is too high. The attacker, who has compromised this switch, changes the policy to drop packets at a lower threshold rate, causing network degradation. When the network operator observes this packet dropping activity, it is impossible for them to distinguish between legitimate drops in line with their policy and malicious drops, which are caused by an attacker’s alteration of the P4 program (assuming a subtle enough change to the program by the attacker). This impossibility is caused by the fact that the DP switches have the ability to falsify their statistics reports to the CP, which can be changed to match the state variable to the observed forwarding behaviour. One mechanism that may be used to combat this would be to replicate the state across multiple switches, before collecting and analysing it at the CP. However, not only would this be difficult due to differences in observed traffic at each switch, it would also lead to redundant usage of register memories on switches. Other work on stateful DPs [2, 44] has recognised the scarcity of these memories as a limitation on the development of DP applications and has proposed distribution of state amongst as many switches as possible to solve this problem. The issues with DP state are equally applicable to ADPV solutions that use DP cryptography, as the CP relies on a real-time view of networking forwarding policy to check for correctness.

In the case of probing solutions, the state problem also necessitates an overhaul of their functionality. The most obvious impact is the massive increase in possible forwarding configurations in the network, as each flow rule may have its behaviour changed by a state variable. Achieving scalability for non-adversarial stateful DP testing tools is already a major research area, as noted in [28], with FlowTest [15], for example, being unable to scale to networks with more than 5 nodes. Even if a tractable solution were to be found, it would be necessary to send enough probe packets through a switch to trigger the state change to be tested, which would not only increase

Figure 2: Illustration of Impact of P4 DP on Distributed Statistics ADPV Mechanism

DP overhead, but would also interfere with the state used by regular DP packets, if probing was carried out on a live production network.

6 CONCLUSION

As SDN continues to mature, it is critical that there is continued focus on the potential security issues associated with it. Adversarial DP verification mechanisms have so far provided an important consideration of the potential damage posed by an attacker with control of a DP forwarding device in an SDN, and have offered a wide variety of mechanisms to detect and prevent such attacks. However, there is a need for improvements to both the security guarantees and scalability of these solutions. In addition, the lack of consideration of P4 DPs leaves a significant gap in attack coverage, which will become increasingly important as the popularity of P4 grows further. With this work, we highlight the relative capabilities and outstanding issues associated with state-of-the-art adversarial DP verification, enabling researchers to improve upon these solutions where necessary.

REFERENCES

- [1] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. 2014. NetKAT: Semantic foundations for networks. *Acm sigplan notices* 49, 1 (2014), 113–126.
- [2] Mina Tahmasbi Arashloo, Yaron Koral, Michael Greenberg, Jennifer Rexford, and David Walker. 2016. SNAP: Stateful network-wide abstractions for packet processing. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 29–43.
- [3] Adam Bates, Kevin Butler, Andreas Haeberlen, Micah Sherr, and Wenchao Zhou. 2014. Let SDN be your eyes: Secure forensics in data center networks. In *Proceedings of the NDSS workshop on security of emerging network technologies (SENT’14)*.
- [4] Kevin Benton, L Jean Camp, and Chris Small. 2013. OpenFlow vulnerability assessment. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. 151–152.
- [5] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 87–95.
- [6] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 99–110.
- [7] Kai Bu, Yutian Yang, Zixuan Guo, Yuanyuan Yang, Xing Li, and Shigeng Zhang. 2018. FlowCloak: Defeating middlebox-bypass attacks in software-defined networking. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 396–404.
- [8] Jiahao Cao, Qi Li, Renjie Xie, Kun Sun, Guofei Gu, Mingwei Xu, and Yuan Yang. 2019. The crosspath attack: Disrupting the {SDN} control channel via shared links. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 19–36.
- [9] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. 2007. Ethane: Taking control of the enterprise. *ACM*

- SIGCOMM computer communication review* 37, 4 (2007), 1–12.
- [10] Tzu-Wei Chao, Yu-Ming Ke, Bo-Han Chen, Jhu-Lin Chen, Chen Jung Hsieh, Shao-Chuan Lee, and Hsu-Chun Hsiao. 2016. Securing data planes in software-defined networks. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. IEEE, 465–470.
 - [11] Po-Wen Chi, Chien-Ting Kuo, Jing-Wei Guo, and Chin-Laung Lei. 2015. How to detect a compromised SDN switch. In *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 1–6.
 - [12] Yen-Chun Chiu and Po-Ching Lin. 2017. Rapid detection of disobedient forwarding on compromised OpenFlow switches. In *2017 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 672–677.
 - [13] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and Vijay Mann. 2015. SPHINX: Detecting Security Attacks in Software-Defined Networks.. In *Ndss*, Vol. 15. 8–11.
 - [14] Mihai Valentin Dumitru, Dragos Dumitrescu, and Costin Raiciu. 2020. Can we exploit buggy P4 programs?. In *Proceedings of the Symposium on SDN Research*. 62–68.
 - [15] Seyed K Fayaz and Vyas Sekar. 2014. Testing stateful and dynamic data planes with FlowTest. In *Proceedings of the third workshop on Hot topics in software defined networking*. 79–84.
 - [16] Seyed K Fayaz, Tianlong Yu, Yoshiaki Tobioaka, Sagar Chaki, and Vyas Sekar. 2016. {BUZZ}: Testing Context-Dependent Policies in Stateful Networks. In *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*. 275–289.
 - [17] A. Feldmann, P. Heyder, M. Kreutzer, S. Schmid, J. Seifert, H. Shulman, K. Thimmaraju, M. Waidner, and J. Sieberg. 2016. NetCo: Reliable Routing With Unreliable Routers. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*. 128–135.
 - [18] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. 2014. 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)*. 71–85.
 - [19] Hongxin Hu, Wonkyu Han, Gail-Joon Ahn, and Ziming Zhao. 2014. FLOW-GUARD: building robust firewalls for software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking*. 97–102.
 - [20] Andrzej Kamiński and Carol Fung. 2015. Flowmon: Detecting malicious switches in software-defined networks. In *Proceedings of the 2015 Workshop on Automated Decision Making for Active Cyber Defense*. 39–45.
 - [21] Rajat Kandoi and Markku Antikainen. 2015. Denial-of-service attacks in OpenFlow SDN networks. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 1322–1326.
 - [22] Nanxi Kang, Zhenming Liu, Jennifer Rexford, and David Walker. 2013. Optimizing the "one big switch" abstraction in software-defined networks. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. 13–24.
 - [23] Qiao Kang, Lei Xue, Adam Morrison, Yuxin Tang, Ang Chen, and Xiapu Luo. 2020. Programmable In-Network Security for Context-aware {BYOD} Policies. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*. 595–612.
 - [24] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P Brighten Godfrey. 2013. 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)*. 15–27.
 - [25] Suriya Kodeswaran, Mina Tahmasbi Arashloo, Praveen Tammana, and Jennifer Rexford. 2020. Tracking P4 program execution in the data plane. In *Proceedings of the Symposium on SDN Research*. 117–122.
 - [26] Maciej Kuźniar, Peter Perešini, and Dejan Kostić. 2015. What you need to know about SDN flow tables. In *International Conference on Passive and Active Network Measurement*. Springer, 347–359.
 - [27] Q. Li, X. Zou, Q. Huang, J. Zheng, and P. P. C. Lee. 2019. Dynamic Packet Forwarding Verification in SDN. *IEEE Transactions on Dependable and Secure Computing* 16, 6 (2019), 915–929. <https://doi.org/10.1109/TDSC.2018.2810880>
 - [28] Yahui Li, Xia Yin, Zhiliang Wang, Jiangyuan Yao, Xingang Shi, Jianping Wu, Han Zhang, and Qing Wang. 2018. A survey on network verification and testing with formal methods: Approaches and challenges. *IEEE Communications Surveys & Tutorials* 21, 1 (2018), 940–969.
 - [29] Jed Liu, William Hallahan, Cole Schlesinger, Milad Sharif, Jeongkeun Lee, Robert Soulé, Han Wang, Călin Cașcaval, Nick McKeown, and Nate Foster. 2018. P4v: Practical verification for programmable data planes. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 490–503.
 - [30] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review* 38, 2 (2008), 69–74.
 - [31] Francesco Musumeci, Valentina Ionata, Francesco Paolucci, Filippo Cugini, and Massimo Tornatore. 2020. Machine-learning-assisted DDoS attack detection with P4 language. *IEEE International Conference on Communications* (2020).
 - [32] Andres Nötzli, Jehadad Khan, Andy Fingerhut, Clark Barrett, and Peter Athanas. 2018. P4pktgen: Automated test case generation for p4 programs. In *Proceedings of the Symposium on SDN Research*. 1–7.
 - [33] Gregory Pickett. 2015. Staying persistent in software defined networks. *Black Hat Briefings* (2015).
 - [34] Takayuki Sasaki, Christos Pappas, Taehoo Lee, Torsten Hoeffer, and Adrian Perrig. 2016. SDNsec: Forwarding accountability for the SDN data plane. In *2016 25th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 1–10.
 - [35] S. Scott-Hayward and T. Arumugam. 2018. OFMTL-SEC: State-based Security for Software Defined Networks. In *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 1–7. <https://doi.org/10.1109/NFV-SDN.2018.8725686>
 - [36] S. Scott-Hayward, S. Natarajan, and S. Sezer. 2016. A Survey of Security in Software Defined Networks. *IEEE Communications Surveys Tutorials* 18, 1 (2016), 623–654. <https://doi.org/10.1109/COMST.2015.2453114>
 - [37] Arash Shaghghi, Mohamed Ali Kaafar, and Sanjay Jha. 2017. 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*. 849–861.
 - [38] Takahiro Shimizu, Naoya Kitagawa, Kohta Ohshima, and Nariyoshi Yamai. 2019. WhiteRabbit: Scalable software-defined network data-plane verification method through time scheduling. *IEEE Access* 7 (2019), 97296–97306.
 - [39] Nguyen Canh Thang and Minhho Park. 2020. An efficient defense method for compromised switch and middlebox-bypass attacks in service function chaining. *Journal of Communications and Networks* 22, 6 (2020), 493–504.
 - [40] Kashyap Thimmaraju, Liron Schiff, and Stefan Schmid. 2019. Preacher: Network policy checker for adversarial environments. In *2019 38th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 32–3209.
 - [41] Kashyap Thimmaraju, Bhargava Shastry, Tobias Fiebig, Felicitas Hetzelt, Jean-Pierre Seifert, Anja Feldmann, and Stefan Schmid. 2018. Taking control of sdn-based cloud systems via the data plane. In *Proceedings of the Symposium on SDN Research*. 1–15.
 - [42] Barefoot Tofino. 2018. World's fastest P4-programmable Ethernet switch ASICs.
 - [43] Richard Wang, Dana Butnariu, Jennifer Rexford, et al. 2011. OpenFlow-Based Server Load Balancing Gone Wild. *Hot-ICE* 11 (2011), 12–12.
 - [44] Menghao Zhang, Guanyu Li, Shicheng Wang, Chang Liu, Ang Chen, Hongxin Hu, Guofei Gu, Qi Li, Mingwei Xu, and Jianping Wu. 2020. Poseidon: Mitigating Volumetric DDoS Attacks with Programmable Switches. In *Proceedings of NDSS*.
 - [45] Peng Zhang, Hui Wu, Dan Zhang, and Qi Li. 2020. Verifying Rule Enforcement in Software Defined Networks With REV. *IEEE/ACM Transactions on Networking* 28, 2 (2020), 917–929.
 - [46] Peng Zhang, Shimin Xu, Zuoru Yang, Hao Li, Qi Li, Huanzhao Wang, and Chengchen Hu. 2018. Foces: Detecting forwarding anomalies in software defined networks. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 830–840.