Securing Data Planes in Software-Defined Networks

Tzu-Wei Chao[†] Yu-Ming Ke[†] Bo-Han Chen[†] Jhu-Lin Chen[†] Chen Jung Hsieh[†] Shao-Chuan Lee[†] Hsu-Chun Hsiao[†‡]
[†]National Taiwan University [‡]Academia Sinica

***Abstract*—Ensuring correct data-plane operations is an integral part of securing software-defined networks (SDN). This paper explores practical solutions for localizing and mitigating malicious switches that disobey installed flow rules. Our main insight is that the flexible and proactive nature of SDNs enables efficient defense against realistic adversaries who can collude and report falsified information in addition to manipulating packet forwarding decisions. In contrast, previous proposals either assume a simple threat model or require expensive cryptographic operations even during peacetime. To systematically explore the design space, we study three complementary techniques for data-plane security, that is, active probing, statistics checking, and packet obfuscation. This paper presents the initial design and implementation of our data-plane security system and highlight potential research challenges.**

I. Introduction

Ensuring correct data-plane operations is challenging even without malicious devices (e.g., routers or switches) in the networks. The presence of malicious devices exacerbates the problem; they can behave stealthily to evade detection and even incriminate benign entities by reporting falsified information. As more and more vulnerable routers and switches are exploited to launch attacks (such as denial of service or hijacking traffic), it is imperative to study how to secure data planes against compromised in-network devices.

Unfortunately, previous solutions to detect or mitigate malicious in-network devices either assume a simple threat model [1]–[4] or incur substantial overhead even during peacetime [5]–[7]. For example, network diagnostic tools such as ATPG [2] often assume a benign environment in which failures occur persistently and thus may incorrectly attribute benign entities in the face of sophisticated attacks (e.g., colluding). On the other hand, while cryptographic-based verification protocols such as OPT [7] can enforce path compliance despite strong adversaries, they may be infeasible for universal deployment due to their high overhead (e.g., pairwise key setup, per-packet cryptographic operations, and increased packet size).

The software-defined networking (SDN) paradigm can facilitate *flexible and proactive* defense, thereby enabling efficient solutions against strong adversaries. Given SDN's programmability and centralized control, our key insights are two-fold. (1) Different defense mechanisms can be flexibly deployed in response to different attack types and severity levels, thereby minimizing wasted resources. The same functionality often requires complex reconfiguration in traditional networks. (2) New optimizations are possible because defense mechanisms in SDN can dynamically change the network configuration (e.g., forwarding rules and topology) to their advantage.

978-1-4673-9486-4/16/$31.00 © 2016 IEEE

With these insights, we design and implement a data-plane security system supporting **fault localization** and **fault mitigation** against malicious SDN switches. Fault localization aims to identify malicious switches, while fault mitigation aims to improve resilience against malicious switches if identification fails. We consider a realistic threat model in which malicious switches can selectively drop, inject, modify, or misdirect traffic. They can also collude with each other and report bogus information to incriminate benign entities. To systematically mitigate such a wide spectrum of threats, we explore three complementary techniques to secure data planes and study how they can be tailored in SDN settings:

- **Active probing** is a fault localization technique that validates the correctness of flow rule execution by sending test packets. A straightforward yet expensive solution is sending one test packet per flow rule. Thus, the main challenge is to minimize the number of test packets while ensuring accurate and rapid detection.
- **Statistics checking** is another fault localization technique that detects malicious switches by checking consistency of flow statistics. A key challenge of statistics-based checking is how to deal with malicious switches that report misleading information to incriminate benign ones.
- **Packet obfuscation** is a fault mitigation technique that, by encrypting packet contents and/or headers, ensures continuous network operations despite malicious switches discriminating against certain flows. Challenges include efficient key management and encryption/decryption.

In the rest of this paper, we define the threat model (§II), elaborate our proposed solutions and highlight potential research challenges (§III), present preliminary results that support our observation (§IV), and review related work (§V).

II. Problem Definition

Our goal is to secure packet forwarding against malicious switches that disobey flow rules. To be practical, a fault localization/mitigation scheme should minimize performance overhead and locate malicious switches with high probability and low delay. This section reviews relevant SDN data plane operations and defines the threat model in detail.

A. Background: SDN/OpenFlow Data Plane

In SDN, the controller handles how packets are forwarded by inserting and deleting *flow rules* on switches. Specifically, a switch maintains one or more *flow tables*. Each flow entry (or flow rule) in a flow table contains *match field*, *actions*, *counters*, *priority*, and *timeout*. When an incoming packet matches a match field, the switch performs the corresponding actions on the packet and updates the corresponding counters. The priority field serves as a tie breaker if multiple rules are matched. The timeout field specifies when the rule will expire. In addition, the controller knows the complete network topology and can request switches for counter values.

B. Threat Model

It has been shown that SDN switches can be compromised [8] and are attractive targets for hacking [9]. We assume an attacker that controls a subset of SDN switches called *malicious switches*. The attacker has full control over all communications going through and all information stored on these malicious switches. Malicious switches can *collude* with each other and *misbehave selectively* with respect to flows and time. Flow-based selective attacks discriminate against certain flows, while time-based selective attacks strike with a certain probability in each time slot. It is worth noting that no prior solution can efficiently mitigate such a strong adversary model.

Specifically, we classify potential threats regarding three fundamental functionalities of SDN switches:

Threats against packet handling. We consider two primary malicious actions—drop and inject—against packet handling, because every attack against data plane integrity can be reduced to a sequence of packet drops and injections [6].[1]

Threats against rule matching. Similarly, the attacker can inject or delete installed flow rules on compromised switches, and the combination of injections and deletions can implement various attacks.[2] In addition, as switches are required to report expired flow rules to the controller, malicious switches can create inconsistency between the controller's view and the actual network state by lying about such timeout events.

Threats against statistics reporting. Malicious switches can report falsified counter values, deluding the controller into making wrong decisions, such as blocking benign switches.

We assume that the controller is trusted. Attacks against controllers are orthogonal to this paper and may be mitigated; for example, using threshold-based cryptography [10]. Denial of service by flooding is another major, unresolved threat to SDN data planes but is outside the scope of this paper. Although the attacker can also use malicious switches to eavesdrop on packet contents, thus resulting in privacy breaches, we focus on attacks against integrity rather than confidentiality.

III. System Description

Our work utilizes the programmability and centralized control of SDN networks, and thus is able to test network functionality *actively* according to the controller's knowledge. By contrast, in a traditional network design, problems are detected *passively* through checking statistics on switches.

[1]E.g., misdirection can be instantiated via dropping and then injecting the same packet to a different interface. Header modification can be instantiated via dropping and then injecting a new packet to the same interface.

[2]E.g., the attacker can manipulate a flow's priority or add a rule to drop packets from a specific IP address.

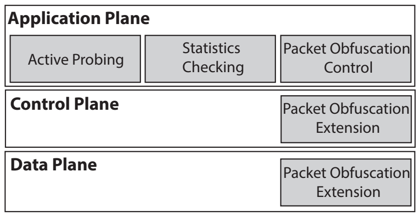
Application Plane
Active Probing
Statistics Checking
Packet Obfuscation Control
Control Plane
Packet Obfuscation Extension
Data Plane
Packet Obfuscation Extension

Fig. 1. Our proposed system w.r.t. the layers of SDN architecture.

Our proposed system deals with the defined problem in two aspects: (1) fault localization by sending test packets and inserting test flow rules, and (2) fault mitigation via packet content obfuscation.

For fault localization, we describe two techniques to locate compromised switches:

(1) We generate a set of test packets from the controller to traverse all flow rules on controlled switches in the network, verifying if the rules are consistent with the controller's knowledge (§III-A). By actively testing the switches, we are able to detect incorrect packet handling by adversaries. However, injected flow rules may not always be discovered with this technique.
(2) We insert test flow rules to switches and collect statistics to identify neighborhoods that report inconsistent results (§III-B). The flow rules on a switch are designed to match the expected behavior of its neighbors. With collected flow counters, we verify the integrity of each switch by comparing counters provided by the switch itself and its neighbors. This method can locate the root cause of incorrect packet handling and unintended flow rules down to the smallest neighborhood (i.e., adjacent switches) but may be unable to determine which one in the neighborhood is misbehaving.

If compromised switches cannot be located with high certainty, or if not every switch is under control, we propose a new flow action that obscures/unobscures packet content. It mitigates malicious switches that perform unintended actions against packets containing a certain header or payload (§III-C).

Figure 1 shows how the proposed system interacts with the standard SDN layered architecture. Both testing methods are implemented at the application level, while the packet obfuscation mechanism requires additional support in the control and data planes. Table I compares the three techniques regarding the types of threats they are capable of detecting/mitigating.

TABLE I
Comparison of three proposed techniques regarding capabilities of detection/mitigation.

Threat Types	Active Probing	Statistics Checking	Packet Obfuscation
Packet Handling	*	*	
Rule Matching		*	*
Statistics Reporting		*	

A. Active Probing: Fault Localization by Sending Test Packets

At a high level, the active probing application consists of two main steps, **test packet generation** and **fault localization**, as Figure 2 illustrates. Before explaining them in detail, we highlight the main technical challenges in each step:

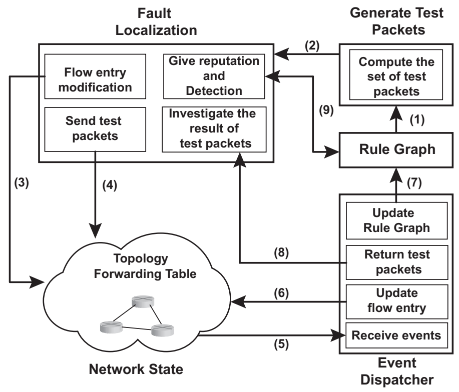

Fig. 2. Overview of the fault localization with test packets.

• **How to efficiently compute a minimum set of test packets.** In test packet generation, the controller computes a minimum set of test packets (P_{min}) that traverse every rule in the network, thereby enabling lightweight monitoring of the entire network. Intuitively, one might attempt to solve this by applying well-established algorithms (e.g., finding a minimum path cover, MPC [11]–[13]). However, as we will explain later, existing algorithms cannot be directly applied due to additional constraints in our setting; thus, we define the *minimum legal path cover (MLPC)* problem, prove that finding P_{min} can be reduced to solving MLPC, and propose an algorithm to solve it in $O(n^3)$ time.

• **How to balance accuracy and efficiency in fault localization.** Test packets are sent periodically. If the test packets do not return as expected, the controller marks the corresponding path as suspected, and begins to pinpoint faults by sending additional test packets. However, narrowing down the suspected region is non-trivial. Suppose the controller spots two suspected paths intersecting each other. Without further information, it would be hard to tell whether the switch at the intersection is malicious or there are two (or more) malicious switches on the paths. We can reduce false positives and false negatives at the cost of sending more packets. We explore the tradeoff by studying a divide-and-conquer algorithm, and leave optimization as a future work.

1) Test Packet Generation: The controller maintains a *rule graph* [1] representing the relationships between flow entries. Given the topology and each switch's flow rules, each vertex in the rule graph represents one flow entry, and there is an edge from vertex r_1 to r_2 if there exists some packet that matches rule r_1 and then rule r_2 consecutively. Since the rule graph encodes only pairwise relationships, there may be no packet that can match every rule on a given path in the rule graph. This explains why finding P_{min} is not the same as finding a minimum path cover in the rule graph.[3]

We say that a path is a *legal path* in the rule graph if a packet can traverse it. We define the **Minimum Legal Path Cover (MLPC)** problem to be finding the minimum number of legal paths such that every vertex belongs to at least one legal path. To this end, generating P_{min} is equivalent to finding a MLPC in the rule graph.

[3]For example, consider a line topology of three switches in which switch w_i has rule r_i. The match field of r_1, r_2 and r_3 are $src\_ip = 10.0.0.1$, $dst\_ip = 10.0.0.2$ and $src\_ip = 10.0.0.3$, respectively. Then the rule graph consists of three nodes and two edges (r_1, r_2) and (r_2, r_3). However, no packet can go through $r_1 \rightarrow r_2 \rightarrow r_3$ because $(src\_ip = 10.0.0.1) \wedge (src\_ip = 10.0.0.3) = \emptyset$.

```
Algorithm 1: Test Packet Generation based on MLPC
  Input  : Rule graph $G$.
  Output: Minimum set of test packets $P_{min}$.
1 $V \longleftarrow$ topological_sorting($G$);
2 $G' \longleftarrow$ transitive_closure($G$);
3 $M \longleftarrow$ Hopcroft-Karp($G', V$) w/ legal augmenting paths;
4 $P_{min} \longleftarrow$ trace_headers($M$);
5 return $P_{min}$;
```

We propose Algorithm 1 to solve MLPC in $O(n^3)$, where n is the number of rules. Our algorithm extends an algorithm [11]–[13] that finds a minimum path cover (MPC). Due to space limitations, we briefly sketch the construction and proof as follows:
First we perform topological sorting of the rule graph. Given a topological order, a modified Hopcroft-Karp algorithm is applied to efficiently find legal augmenting paths. The Berge's Theorem shows a legal path cover C is minimum if and only if no legal augmenting path on C can be extended. Given a minimum legal path cover C_{min}, we compute the header space H_i (e.g., $src\_ip = 10.0.x.x$) for each path ℓ_i in C_{min}. That is, any packet with a header within H_i can traverse ℓ_i. Finally, P_{min} is generated by selecting one packet per H_i. The core observation regarding this proof is that a similar algorithmic construction of MPC works for finding MLPC in the sense that MLPC has a stricter definition of paths than MPC.

2) Fault Localization: For the purpose of fault localization, test packets need to be returned to the controller. To do this, the controller appends an additional test flow rule at the end of each test path. Ideally, to avoid affecting normal packets, the appended test flow rule and test packet should be sufficiently unique such that the test flow entry can only be matched by the corresponding test packet. Our implementation achieves this by putting a unique value in unused header fields (e.g., VLAN ID). Moreover, the last rule r on each test path should be handled as follows to ensure that r can be tested without affecting normal operations: (1) duplicate r to the next flow table, (2) add the test flow entry that is set to the highest priority on the next flow table, (3) modify the action of r on the original table to `goto` the next table.

If a test packet is dropped (i.e., do not return before timeout) or modified, the controller decreases the reputation of the corresponding path. When the reputation value drops below a certain threshold, the path is marked as suspected. To determine the root cause (which switch is malicious), the controller slices the path into two sub-paths and performs the same procedure recursively until the length of the suspected path is 1. Algorithm 2 shows this procedure.

B. Statistics Checking: Fault Localization by Cross-analyzing

In the OpenFlow protocol, switches record information about processed packets, including the number and size of

```
Algorithm 2: Localize the fault with test packets
Input  : Set of test packets $P$.
Output: Faulty or compromised switch(es).
1  while not localize the fault do
2      send test packets $P$;
3      wait and receive the set of test packets $P'$;
4      for each test packet $p \in P$ and correspond $p' \in P'$ do
5          $path \longleftarrow$ the path traversed by $p$;
6          if $p \notin P'$ or $p \neq p'$ then
7              decrease the reputation of $p$;
8          if the reputation of $p < threshold$ then
9              if length of $path$ = 1 then
10                 localize the fault;
11             else
12                 slice_path($path$, $p$);
```

processed (received, forwarded and dropped) packets, the number of packets that match some rules in a flow table, etc. By requesting these statistical data, the controller can have an overall understanding of the network.

In the presence of malicious switches reporting fake information, the controller requires extra information to validate reports and identify malicious switches. To achieve this, we propose that each switch can perform "neighborhood watch" to monitor the behavior of neighboring switches, thereby ensuring **traffic conservation** for each switch. If there are x packets entering a switch and y packets are dropped or ended at the switch, then there should be $x - y$ packets leaving the switch. This can be done at a finer granularity such as per-flow or per-interface conservation.

At a high level, the controller inserts extra *counter rules* to switches so that it can identify compromised switches by comparing counter values among neighboring switches. The sole function of a counter rule is to count packets and it does not change the original behavior of forwarding.

Technically, our system consists of two parts: **adding counter rules** and **checking counter values**. For adding counter rules, the main challenge is ensuring that these newly-added counter rules will not interfere with normal forwarding. For checking counter values, the difficulty lies in how to pinpoint misbehaving switches when they can lie and collude.

1) Adding Counter Rules: We leverage a new OpenFlow feature—the support of multiple flow tables—to store auxiliary counter rules on switches so that counting can be done transparently without affecting normal forwarding.

As a proof of concept, our prototype system works as follows. The controller inserts a new table $from\_table$ and several new flow tables to_x_table before and after the original flow tables, correspondingly. x represents a switch ID, and every action (including default actions) on to_x_table would be "forward to x". When a packet arrives at a switch, it will go through $from\_table$, original flow tables, and to_x_table in order before being forwarded to next switch.

Let $N(s)$ denotes the set of neighbors of a switch s. $O(r)$ denotes a set of output switches, which are next-hop switches

```
Algorithm 3: Add counter rules
Input: A rule $r$ on a switch $s$
1  $r' \longleftarrow r$;
2  for $s' \in O(r)$ do
3      if $s' \in O(r)$ then
4          action($r'$) $\longleftarrow$ go to original flow table of $s'$;
5      else if $s' \in I(r)$ then
6          action($r'$) $\longleftarrow$ forward to s;
7          add $r'$ at $to\_s\_table$ on $s'$;
8          action($r'$) $\longleftarrow$ drop;
9      match_field($r'$) $\longleftarrow$ match_field($r'$) + 'in_port'=$s$;
10     add $r'$ at $from\_table$ on $s'$;
```

```
Algorithm 4: Validate the malicious switch
Input  : Target switch $s$
Output: If statistic is consistent
1  for each $s' \in N(s)$ do
2      request $from\_table$ and $to\_s\_table$ on $s'$;
3  for each rule $r$ on switch $s$ do
4      if $r$ is dropping rule then
5          if $C(r)$ in every from_table $\neq 0$ then
6              return $false$;
7      else if $r$ is forwarding rule then
8          if $|\sum_{s' \in O(r)} C(r) - \sum_{s' \in I(r)} C(r)| > \theta$ then
9              return $false$;
10 return $true$;
```

of a forwarding rule r. If r is a dropping rule (i.e., whose action is to drop packets), $O(r)$ becomes an empty set. $I(r)$ denotes the difference of $N(s)$ and $O(r)$. Algorithm 3 shows the procedure of adding counter rules. Notice that we need to add counter rules for every rule on every switch to obtain enough information for counter checking.

2) Checking Counter Values: Under our threat model, the switch that the controller wants to check might return incorrect data to avoid detection. Hence, instead of requesting statistics from the target switch s directly, the controller will simultaneously request statistics from $s' \in N(s)$, determining whether there are compromised switches by validating the consistency of the returned data. Let $C(r)$ denotes the auxiliary counter of r, and θ denotes a tolerance threshold for packet loss and latency. The validation procedure is depicted in Algorithm 4.

We set a reputation value for each switch. The reputation of the target switch and adjacent switches decrease when an inconsistency occurs. When the reputation value of a switch drops below a threshold, the switch is potentially compromised. It is important to note that this statistics-checking method is suitable for localizing faults but may fail to accurately identify malicious switches. In some cases, we can at best localize the fault to one link but we cannot determine which one of the two switches is malicious. Also, if a benign switch is surrounded by a coalition of malicious switches, its reputation may decrease faster than others. Hence, this method should be combined with the other two methods to reduce false positives and negatives.

For future work, we would like to extend our design to detect other malicious actions in addition to traffic misdirection and dropping. Another unsolved challenge is to reduce the number of auxiliary counter rules. In our algorithm, the number of auxiliary rules increases with the number of adjacent switches, and thus adding counter rules may become a performance bottleneck and exhaust TCAM memory on switches. We plan to investigate probabilistic approaches for better performance.

C. Packet Obfuscation to Prevent Rule-based Misbehavior

A compromised switch is capable of performing unintended actions to packets that match certain criteria, such as packets with specific source/destination addresses or those containing a certain string in the payload. An attacker can thus manipulate sensitive traffic, or even perform man-in-the-middle attacks.

A simple yet effective method of mitigating adversaries is to obfuscate packet headers and payload when sensitive packets enter an untrusted region, and recover the packets before they reach their destinations.

Based on the OpenFlow architecture, we propose a packet obfuscation scheme by means of encryption. We extend the original OpenFlow specification by adding two new flow actions: `encrypt` and `decrypt`. By utilizing modern cryptographic algorithms we are able to prevent the adversary from recovering packet content without access to the secret key.

Encryption method. Modern cryptographic algorithms can be categorized into two types: *symmetric-key* and *asymmetric-key* cryptography. The performance of modern public-key algorithms (e.g., RSA) is hardly comparable to symmetric-key algorithms, which is a major concern in practical usage. Thus, we consider applying symmetric-key cryptography. Switches performing corresponding `encrypt` and `decrypt` actions are assigned a pre-shared key for both operations. Performance is especially guaranteed if hardware support is available, such as Intel's AES-NI instruction set for the x86 architecture. The pre-shared key should be changed constantly to prevent the exploitation of weaker algorithms (e.g., attacks to the RC4 cipher); therefore, a key synchronization mechanism between switches is required for better securely.

Encryption range. Encryption allows us to hide certain routing information, but the packet still needs to be correctly routed to the destination. Applications utilizing this technique should be aware that obfuscated information will be unavailable for successive routing until packets are navigated to switches that match rules and unobfuscate them.

In our design, the `encrypt` and `decrypt` flow actions include a range field deciding whether to encrypt the Layer 2 (L2) payload or Layer 3 (L3) payload:

- By encrypting the L3 payload, L4 (e.g., TCP/UDP) information such as source/destination ports will be unavailable.
- By encrypting the L2 payload, L3 information including source/destination IP addresses, as well as L4 information will be hidden. In this case only traditional switching with MAC addresses will be available.

The proposed method provides protection transparent to end devices, and is more powerful than other approaches such as VPNs, which are unable to protect routing information. Our implementation in progress utilizes an AES-256-CBC cipher with a static pre-shared key field included in both `encrypt` and `decrypt` flow actions.

IV. Preliminary Results

Our fault localization and mitigation system is implemented on top of Ryu and Open vSwitch in accordance with OpenFlow 1.3. This section reports our evaluation methodology and preliminary evaluation of the active probing method (§ III-A).

Methodology. To evaluate our data-plane security system in realistic and diverse settings, we develop an automatic tool to synthesize topologies and flow entries based on real datasets. The tool then creates a virtual SDN network using Mininet. Our topology is sampled from the router-level ISP topologies measured by Rocketfuel [14]. As we are unaware of any publicly available flow entry datasets, we use the all-pair K-th shortest path algorithm to compute routing paths given the synthesized topology. For each switch on a path, we insert a flow entry that matches the destination's IP prefix, such that packets travelling to the prefix can traverse the path.

Preliminary results. Table II summarizes the results of the test packet generation under different settings. The maximum path length represents the maximum number of rules traversed by one packet. The results confirm that our solution can significantly reduce the number of required test packets to cover every rule on the network, and the compression ratio is roughly inversely proportional to the maximum path length.

TABLE II
Results of test packet generation.

# of switches	10	30	30	79
# of wires	15	54	54	147
Max path len.	5	5	8	8
# of rules	4764	82740	33637	205713
# of test packets	954	15098	4203	24456

Figure 3 shows the detection delay of fault localization vs. the number of malicious switches under four settings (Table II). Detection delay is defined as the time between sending the first test packet and catching all malicious switches. Given the number of malicious switches, we determined which switches and rules were malicious uniformly at random in each run of the experiment. The minimum detection delay is 6.19 seconds and the maximum is 89.25 seconds in all experiments. The controller can localize malicious switches within 6.19-8.37 seconds in the smallest topology, within 12.28-35.65 seconds in the medium topology and within 68.27-89.25 seconds in the largest topology. The detection delay is roughly proportional to the number of test packets being sent.

V. Related Work

We review prior work in fault localization and mitigation.

Several network diagnostic tools are proposed for finding network misconfiguration (e.g., loops and black holes) [2],

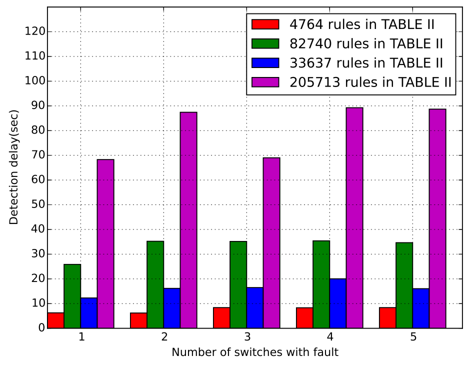

Fig. 3. Detection delay of fault localization.

[15], [16]. For example, ATPG [2] aims to locate faults by sending test packets that traverse the entire network. VeriFlow [16] efficiently verifies network-wide invariants by dynamically tracking flow updates. However, prior works often focus on reliability rather than security and thus can be bypassed by sophisticated adversaries.

Cryptographic-based verification protocols can localize faults and enforce path compliance against strong adversaries [5]–[7] but often incur high overhead and require significant changes to the current network architecture. For example, ShortMAC [6] requires that every router on the forwarding path add a small authenticator to each packet. DynaFL [5] requires that each router record a sketch of every packet and compare all the sketches in a neighborhood to eventually detect malicious links. Our active-probing and statistics-checking approaches leverage SDN's features to reach similar goals without changing the network architecture.

Chi et al. [4] propose a test-packet-based method that randomly selects one rule for testing in each epoch. However, this is inefficient and may be circumvented by colluding switches. FortNox [17] extends the Nox controller to check rule contradiction on a single switch against malicious OpenFlow applications, which is orthogonal to our work.

Several studies provide traffic monitoring to support anomaly detection and network forensic analysis in SDN [18]–[21]. However, they often assume trusted switches and focus on monitoring rather than locating where problems occur.

Encryption is commonly used to make traffic indistinguishable against a discriminating adversary [22]. Mowla et al. [23] propose an approach that encrypts packets and inserts encryption rules into switches to defend against spoofed IP attacks in SDN environments. In our work, we propose a user-transparent encryption/decryption scheme added under the OpenFlow protocol to obscure/unobscure packets to prevent the incorrect forwarding by discriminating switches.

VI. Conclusion

SDNs are believed to accelerate the innovation process. At the same time, however, they present new challenges to security research. This paper explores how SDN can help the detection and mitigation of compromised switches under a realistic threat model. We presented three promising solutions to localize and mitigate malicious SDN switches, compared these solutions, and articulated the research challenges during design and implementation. We hope this paper can serve as a roadmap to study data-plane attacks and defenses in SDN.

Acknowledgments

This study is conducted under the “Accelerating the development of mobile broadband services and industry - 4G+ network and application field deploy and testing project” of the Institute for Information Industry which is subsidized by the Ministry of Economic Affairs, Taiwan.

References

[1] P. Kazemian, M. Change, and H. Zheng, “Real Time Network Policy Checking Using Header Space Analysis,” in *USENIX NSDI*, 2013.
[2] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown, “Automatic Test Packet Generation,” *IEEE/ACM Transactions on Networking*, vol. 22, pp. 554–566, 2014.
[3] C. Fung and C. Fung, “FlowMon: Detecting Malicious Switches in Software-Defined Networks,” in *SafeConfig*, 2015.
[4] P.-W. Chi, C.-T. Kuo, J.-W. Guo, and C.-L. Lei, “How to detect a compromised sdn switch,” in *IEEE NetSoft*, 2015.
[5] X. Zhang, C. Lan, and A. Perrig, “Secure and Scalable Fault Localization under Dynamic Traffic Patterns,” in *IEEE S&P*, 2012.
[6] X. Zhang, Z. Zhou, H.-C. Hsiao, T. H.-J. Kim, A. Perrig, and P. Tague, “ShortMAC: Efficient Data-Plane Fault Localization,” in *NDSS*, 2012.
[7] T. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig, “Lightweight Source Authentication and Path Validation,” in *ACM SIGCOMM*, 2014.
[8] “SDN switches aren't hard to compromise, researcher says,” http://www.networkworld.com/article/2956777/security/sdn-switches-arent-hard-to-compromise-researcher-says.html.
[9] “NSA Laughs at PCs, Prefers Hacking Routers and Switches,” http://www.wired.com/2013/09/nsa-router-hacking/.
[10] S. Matsumoto, S. Hitz, and A. Perrig, “Fleet: Defending SDNs from Malicious Administrators,” in *ACM HotSDN*, 2014.
[11] R. P. Dilworth, “A decomposition theorem for partially ordered sets,” *Annals of Mathematics*, pp. 161–166, 1950.
[12] C. Berge, “Two theorems in graph theory,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 43, no. 9, p. 842, 1957.
[13] J. E. Hopcroft and R. M. Karp, “An n^5/2 algorithm for maximum matchings in bipartite graphs,” *SIAM Journal on computing*, vol. 2, no. 4, pp. 225–231, 1973.
[14] “Rocketfuel,” http://research.cs.washington.edu/networking/rocketfuel/.
[15] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, and S. T. King, “Debugging the data plane with anteater,” *ACM SIGCOMM Computer Communication Review*, vol. 41, p. 290, 2011.
[16] A. Khurshid, W. Zhou, M. Caesar, and P. Godfrey, “Veriflow: verifying network-wide invariants in real time,” *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 467–472, 2012.
[17] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, “A security enforcement kernel for OpenFlow networks,” in *ACM HotSDN*, 2012.
[18] N. Handigol, B. Heller, V. Jeyakumar, D. Maziéres, and N. McKeown, “Where is the debugger for my software-defined network?” in *ACM HotSDN*, 2012.
[19] J. R. Ballard, I. Rae, and A. Akella, “Extensible and scalable network monitoring using opensafe,” in *INM/WREN*, 2010.
[20] N. L. Van Adrichem, C. Doerr, F. Kuipers *et al.*, “Opennetmon: Network monitoring in openflow software-defined networks,” in *IEEE NOMS*, 2014.
[21] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, “Payless: A low cost network monitoring framework for software defined networks,” in *IEEE NOMS*, 2014.
[22] I. Avramopoulos and J. Rexford, “Stealth Probing: Efficient Data-Plane Security for IP Routing,” in *USENIX ATC*, 2006.
[23] N. Mowla, I. Doh, K. Chae *et al.*, “An efficient defense mechanism for spoofed IP attack in SDN based CDNi,” in *ICOIN*, 2015.