Controller-Related Security Risks and Vulnerabilities in Software-Defined Networking

By

Nicolas Plas
Supervisors: Mauro Conti, Chhagan Lal
EEMCS, Delft University of Technology, The Netherlands

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
06/17/2022

Abstract

Software Defined Networking (SDN) is a relatively new networking paradigm that proposes to separate the control and the data logic in networks. The control logic is centralized in a controller, which allows for a programmable network. SDN is promising but also introduces some critical security vulnerabilities to networks. This work proposes a survey of state-of-the-art research into attacks and state-of-the-art defences arising from controller placement, controller failure and the northbound interface. Furthermore, it proposes a comparison and analysis of the limitations of that research. Finally, it proposes future research directions to improve SDN security focused on network consistency and on the interoperability of different defences.

1 Introduction

Software-Defined Networking (SDN) is a relatively new networking paradigm that proposes revolutionary changes to networks as we know them today. SDN decouples the control logic from the data forwarding logic in networks [1]. A programmable controller handles the control logic, also known as the control plane, whereas routers, or switches, are responsible for the data forwarding logic, also called the data plane. The controller gets its instructions from applications connected to it, like load balancing applications or firewalls. In figure 1, a representation of a network using SDN is given.

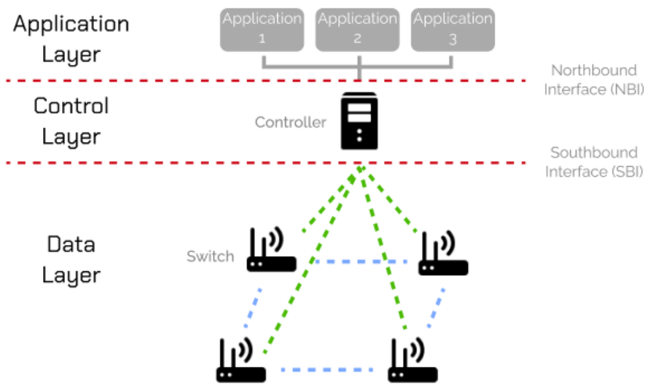

Figure 1: Representation of a network using SDN

This new approach to networking offers numerous advantages to the classical one. In most routers used today, the software or the control part is hard-coded and almost impossible to update after the deployment of a network. Such updates would require each router to be updated individually. In most networks, this is not feasible due to the enormous amount of devices and the differences in vendor-specific commands. These constraints are why getting rid of old networking standards like IPv4 is hard, even though these standards have become obsolete. Classical networks have almost no capabilities to deal with evolving, dynamic networks. The separation of the control and data plane, as SDN proposes, allows these issues to be solved. For example, an update from IPv4 to IPv6 would only require an update of the controllers. Additionally, the vertical dissociation of the control plane from the data plane allows for smart load-balancing and traffic forwarding.

However, one of the under-explored aspects of SDN is the vulnerabilities it possesses [2]. The separation of control and data planes means that if a malicious actor has access to the control plane, it has access to the entire network. This control centralization creates vulnerabilities and security threats that are not present in networks today. While SDN is a promising networking paradigm, its wide deployment beyond its specific use cases today requires addressing these vulnerabilities.

This research will focus on security vulnerabilities and threats arising from controller placement, controller failure, and the northbound interface. The controller placement problem was first proposed by Heller *et al* in [3] and consists of finding the optimal placement and amount of controllers for a network. A network using one controller will have very low latency but also a low fault tolerance, and vice versa. Finding an optimal controller distribution is still challenging, and this survey will address some of the latest proposed strategies to achieve this. Controller failure can be a significant issue for SDN as it can harm network performance: one failed controller could mean that an entire section of the network gets disconnected. Strategies mitigating the effects of controller failures are critical to SDN security. The northbound interface (NBI) in SDN is the interface between network applications and the control plane. Network applications used in SDN have an immense degree of control over the network: they can, for example, insert new flow rules, view network traffic and view the topology of the network [4]. Malicious or buggy applications connected to the NBI can use these privileges to perform a variety of attacks.

The main goal of this paper is to survey attacks exploiting the vulnerabilities outlined in the previous paragraph and state-of-the-art solutions to help counter those attacks. Furthermore, it will also contain an analysis of the effects on network performance caused by these vulnerabilities, as well as a future research direction for this subject.

The rest of this paper is structured as follows. Section 2 provides an overview of related work in this area. Then, Section 3 presents an overview of state-of-the-art controller-related attacks and defences. Section 4 compares similar solutions, their effectiveness in mitigating vulnerabilities and how they affect network performance. Section 5 contains a discussion on the ethical repercussions this paper might have. Section 6 examines the limits of the most recent research, future research directions to improve current solutions, and the vulnerabilities that are still under-explored. Finally, Section 7 contains the conclusion of this paper.

2 Related Work

Various research efforts into controller-related security have already been made. However, papers surveying controller-related vulnerabilities, comparing different attacks exploiting those vulnerabilities and surveying a wide range of state-of-the-art defences are more scarce.

Rauf et al [4] focus on the NBI and its importance in SDN security. They discuss the different types of NBI that exist and have made a detailed survey of NBI-related vulnerabilities. It contains an examination of how a malicious application could exploit those vulnerabilities to attack a network using SDN. Finally, the research proposes advanced solutions to all the examined vulnerabilities.

Vizarreta et al [5] have produced an in-depth overview of controller failure dynamic. It contains an analysis of the different ways a controller can fail and the impact of controller failure on network availability. To help quantify the network availability, it proposes a Stochastic Activity Network model, a powerful tool for modelling dependability in a network.

Das et al [6] have made a survey on papers discussing controller placement. It investigates the different proposed controller placement schemes and the metrics that have been used to qualify them. It then classifies the papers into different categories based on how the proposed controller placement scheme functions.

This paper will be, to the best of our knowledge, the first survey that considers attacks on the whole of the control plane, and not a specific subsection of it. Moreover, it also contains an evaluation of how attacks exploiting these vulnerabilities affect network performance and it will provide a summary and analysis of the state-of-the-art solutions that exist to help diminish the risks that result from those attacks.

3 Attacks and State-of-the-Art Defences

This section contains an analysis of multiple papers. They contain attacks and solutions to defend against them or solutions to mitigate the effects of controller failure. Each subsection contains one paper. Table 1 contains an overview of all the papers that will be analyzed.

3.1 Cross-App Poisoning in Software-Defined Networking

In [7] a new attack on the SDN control plane, Cross-App Poisoning (CAP), is introduced. This attack makes it possible to bypass Role-Based Access Control (RBAC) defences. It also proposes a solution to defend against this attack, named ProvSDN.

Role-Based Access Control

RBAC is an SDN defence strategy that attempts to mitigate the effect malicious applications can have on a network by assigning roles with different reading and writing privileges to network applications [13]. In theory, a malicious application that does not possess writing privileges will not be able to modify the flow rules of the control plane. Furthermore, if it does not have reading access, it should not be capable of reading the network state.

Threat model

The authors of this paper assume that the SDN controller is secure and that the used network applications can originate from third parties. Furthermore, they presume that an attacker controls a malicious application with the least privileged RBAC permissions. The attacker's objective is to install a malicious flow rule without having permission to do so.

Information Flow and Information Flow Challenges for SDN

Information Flow describes how data propagates through a system and how it affects other data in that system. Information flow control (IFC) is a mechanism that uses a security policy to decide what data can flow between various entities.

SDN lacks controller designs that record information flow and perform IFC between the numerous applications connected to the control plane. As a result, modern controller designs that use an RBAC defence are vulnerable to malicious applications exploiting the lack of IFC to bypass the privileges initially assigned to them.

Cross-app Poisoning Attack

Cross-App Poisoning (CAP) exploits the lack of information flow control in modern SDN controllers with RBAC. All attacks using malicious applications to poison the network view of other applications, thus misleading them into performing unauthorized actions on their behalf, can be described as CAP attacks.

In their experiments, the authors manage to execute a CAP attack by designing an application that poisons the network view of the reactive forwarding application by modifying the routing information the application uses. The forwarding application consumes the poisoned data to set up a new flow rule which causes a denial-of-service of the data plane for the victim. Furthermore, the ID associated with the malicious flow rule will be the ID of the forwarding app, wrongly causing the controller to believe that the forwarding app is responsible for the attack and allowing the malicious application to remain completely undetected.

ProvSDN

This paper also proposes a defence against CAP attacks which they name ProvSDN. ProvSDN uses data provenance to determine whether API calls to the controller should be executed or blocked. Data provenance informs us where data comes from and what changes it has made to the network. The three components of ProvSDN are the provenance collector, the online reference monitor and the provenance graph.

The provenance collector collects API call information like data usage, called methods, generated data, and identifies the network relations and entities involved. The online reference monitor checks the provenance graph against the chosen IFC policy and decides whether the application executing the call satisfies the security policy. If that is the case, the monitor includes the application to the provenance graph, and if not, the monitor blocks the API call. The provenance graph keeps track of all the entities and relations in the network and enables online policy checking and investigating how a network state came to be, effectively allowing the controller to detect malicious applications like the one in the example attack.

Research Work	Area of Interest	Focus	Heuristic
Cross-App Poisoning in Software-Defined Networking [7]	NBI	Attack	Using one application with low privilege to trick other applications into performing higher privileged operations
An Efficient Approach to Robust SDN Controller Placement for Security [8]	Controller Placement	Defence	Using greedy algorithms and a Monte Carlo simulation to quickly find near-optimal multi-link failure resilient controller placement schemes
Byzantine-Resilient Controller Mapping and Remapping in Software Defined Networks [9]	Controller Placement	Defence	Using a primary and backup controller approach to create a Byzantine fault tolerant network
SDN-RDCD: A Real-Time and Reliable Method for Detecting Compromised SDN Devices [10]	Controller Failure	Defence	Adding new controller role, auditor, to verify network interactions and detect compromised devices
AIM-SDN: Attacking Information Mismanagement in SDN-datastores [11]	NBI	Attack	Exploiting lack of synchronization between SDN datastores to cause memory overflows and permit unauthorized traffic
Attacking the Brain: Races in the SDN Control Plane [12]	Controller failure	Attack	Detect and exploit race conditions in SDN controllers

Table 1: Overview of discussed papers, their area of interest, their focus and the heuristic used

Evaluation

ProvSDN is evaluated by executing the example attack described in the CAP section. They use two different scenarios: one involving reads, and one involving writes. In both cases, ProvSDN manages to restrict the malicious application from inserting a malicious flow rule. They also evaluate the performance effects of ProvSDN on the network. On a baseline network without ProvSDN, the average latency for control plane state changes is 11.66 ms. With ProvSDN, the average latency is 29.53ms, which seems like a significant increase over the baseline network. It is meaningful to note that control plane changes are relatively infrequent as they only happen when internal controller services and new applications register to receive events. When flow rules already exist, ProvSDN is not used, and the network experiences no latency.

3.2 An Efficient Approach to Robust SDN Controller Placement for Security

In [7], the focus is on a new controller placement scheme resilient against multi and single-link failure. This placement scheme uses a greedy algorithm and a Monte Carlo simulation to reduce computational overhead.

Link Failure

There are two ways in which the control plane can fail: link failure and controller failure. Link failure is when a physical or virtual link between controllers is compromised, and a section of the network is disconnected from the rest. Controller failure is when a controller fails and cannot perform its duties anymore.

This paper exclusively focuses on the link failure problem. The link failure problem has two aspects: single-link failure and multi-link failure. This paper distinguishes itself from other research on this subject because the placement scheme it proposes is resilient against multi and single-link failure.

The authors find that in the data traces of the links of the China Education and Research Network consisting of 21 nodes and 23 physical links, the failure rate follows a “long tail” feature, which implies that a few specific links cause most of the link failures. This feature can also be observed in other networks and constitutes the basis for the placement heuristic employed in this paper as it implies that some links are more likely to than others fail [14].

Placement for Mitigating Single-Link Failure

This paper proposes two algorithms to solve the controller placement problem for single-link failures: the Optimal Placement Algorithm(OPA) and the Greedy Placement Algorithm(GPA).

OPA implements an exhaustive search of all the possible controller placement schemes to discover the optimal solution to the controller placement problem. Exhaustive searching causes this algorithm to run in non-polynomial time with worst-case time complexity of $O(m \times \binom{n}{k})$ where m is the number of physical links, n is the number of nodes and k is the number of controllers.

GPA, on the other hand, adopts a greedy strategy for controller placement. Utilizing a greedy strategy has the disadvantage of not always finding an optimal solution. However, it has the advantage of having a considerably lower worst-case time complexity of $O(mnk)$.

Placement for Mitigating Multi-Link Failure

For the multi-link failure aspect of the controller placement, the paper again proposes one optimal and one greedy algorithm. The OPA for multi-link failure is similar to the OPA for single-link failure. It performs an exhaustive search of the possible controller placement schemes while monitoring how all the possible failure scenarios can affect the worst-case latency and ultimately selects the one with the lowest worst-case latency. However, this algorithm runs in non-polynomial time with an exponentially higher worst-case time complexity than its single-link variant.

Each link in the network has a certain failure probability according to statistical data. Therefore it is possible to employ a Monte Carlo simulation to simulate the failure scenarios to reduce the computational overhead. The greedy algorithm implements the Monte Carlo simulation to estimate the multi-link failure scenarios according to the link failure rates N times. Subsequently, it implements the same algorithm from the single-link failure section to place the controllers. This algorithm has a worst-case time complexity of $O(knN)$.

Evaluation

To evaluate their algorithms, the authors of this paper use real network topologies, like Internet Topology Zoo [15], Internet2 [16], and Cernet2 [17]. The metrics they use to evaluate the network performances are worst-case propagation runtime, latency, and the probability of network survival.

To evaluate the results, the authors compare their results with two commonly used algorithms for controller placement and find that GPA performs significantly better. It produces almost optimal performance while having a substantially lower computational overhead than an optimal placement algorithm. Latency increase due to GPA is minimal. The probability of network survival is near 100%, particularly when the number of controllers in the network increases.

3.3 Byzantine-Resilient Controller Mapping and Remapping in Software Defined Networks

A critical security issue in SDN is controller failure due to attacks and software failure. The authors of this paper propose a dynamic Byzantine Fault Tolerant (BFT) controller placement scheme that helps mitigate the security issues arising from controller failure [9]. In contrast with the previous paper, this paper focuses on the controllers themselves and not the links between them.

Byzantine Fault Tolerance in SDN

The Byzantine generals problem is a famous and valuable computer science concept for describing how networks can agree on a consistent network state when it is impossible to know which parts of the network to trust. This is equally valid for SDN, and BFT represents an efficient strategy for guaranteeing data integrity in the control plane. However, the conventional BFT approach to secure the control plane against controller failures requires mapping each switch to $3f + 1$ controllers where f is the number of failures a network must be able to overcome [18]. Furthermore, all the controllers must coordinate to achieve a consensus before forwarding new flow rules. This approach makes the network experience a slowdown in throughput and flow-setup time [19]. This reduced effectiveness demonstrates the need for a new BFT algorithm that reduces overhead.

Primary-Backup Controller Approach for BFT

The authors of this paper propose an alternative approach for controller mapping that requires only $2f + 1$ controllers per switch. The new mapping is based on a primary and backup controller approach. Each switch is mapped to $f + 1$ primary controller and f backup controllers. When receiving a new forwarding rule, the switch verifies with all its primary controllers whether the received rule is consistent. If there is any inconsistency among the primary controllers, the switch contacts its backup controllers to receive and install the correct rule. This approach offers significant advantages:

- Reducing overhead: when there is no failure, it only requires $2f + 1$ messages; when there is a failure, it requires $2(2f + 1)$ messages. The conventional approach always requires $2n = 3n^2$ messages where $n = 3f + 1$ is the amount of controllers required per switch.
- Decreased controller load: a portion of the controllers operate as backup and are queried less often.
- Decreased number of controllers: the capacity of backup controllers can be shared among switches.

MINCON and MINRUS

Optimal algorithms exist to map the switches to primary and backup controllers. However, these algorithms can be extremely time-consuming, especially when the networks grow in size. The authors propose a fast algorithm for mapping switches and minimizing the network's number of needed controllers, named MINCON. MINCON uses the greedy programming paradigm to rapidly produce a mapping for switches to primary and backup controllers. MINCON will not use new controllers unless no controllers in the network satisfy the conditions required for connecting a new switch to keep the number of used controllers as low as possible. It also ensures that a controller can not be a backup and a primary controller for the same switch. MINCON has a worst-case time complexity of $O((f + 1) \times |N| \times |M|)$ where $|N|$ is the amount of nodes in the network and $|M|$ the amount of controllers. Since f usually is a low number, the worst-case time complexity is dominated by $O(|N| \times |M|)$.

If a faulty controller is detected, the network cannot use it anymore, and the switches connected to that controller must be remapped. The paper proposes another greedy algorithm to do the remapping, MINRUS. MINRUS remaps all switches connected to a defect controller to other non-defect controllers in the network. However, to reduce computational overhead, it is practical to avoid remapping switches connected to benign controllers as much as possible. To do this, MINRUS tries finding a controller with a communication delay beneath a certain threshold for each affected switch. Only when it cannot find such a controller it will remap an unaffected switch with an identical process to free up a spot and try to accommodate the affected switch. MINRUS uses a greedy strategy and, as a result, has a worst-case time complexity of $O(|N| \times |M|)$.

Evaluation
The key metrics the authors use to evaluate their algorithms are:

- The total number of controllers used.
- the Min-Max ratio, an index which is the ratio of the minimum load to the maximum load among the controllers used to determine how fair the load distribution is.
- The average controller load.
- number of unaffected switch remaps.

The algorithms are evaluated by comparing them to the optimal solutions for mapping and remapping controllers and the conventional BFT algorithm BG-FT. MINCON outperforms the conventional algorithm by a significant margin and enjoys better performances on all metrics. The experiments also show that MINCON and MINRUS perform slightly worse than optimal solutions while experiencing significantly lower computational overheads.

3.4 SDN-RDCD: A Real-Time and Reliable Method for Detecting Compromised SDN Devices

In [10], a new method for detecting compromised SDN devices, named SDN-RDCD, is proposed.

Detection of compromised SDN devices
Detecting compromised devices in SDN involves comparing a device's expected behaviour with its actual output. However, this detection problem is far more challenging for SDN than for conventional networks, where it is possible to compare the output a device produces with the output the protocol installed on the device should produce. In SDN, many applications and controller modules are involved in programming how the control plane and data plane should behave, which leads to controllers and switches behaving in a multitude of unpredictable ways.

Threat Model
The defined threat model assumes that neither controllers nor switches are trustful and that an attacker has compromised an unknown amount of controllers or switches. Then, he can instruct state changes and disable primary and security systems of compromised devices. However, to keep their solution effective, the authors assume that at least one of the switches and one of the controllers involved in an attack stay honest.

Proposed Solution: SDN-RDCD
To solve this detection problem, the authors of this paper propose a new solution: SDN-RDCD. To detect compromised devices in a network, SDN-RDCD collects all network update information and the handling of that update by all the controllers and switches in the network. To detect anomalies in the expected network state after network updates, SDN-RDCD proposes a new role for controllers, besides master and slave, the auditor-controller. The auditor-controller is charged with verifying whether the behaviour of switches and controllers is consistent with the expected behaviour after each network update. Each executed network update and its handling information is stamped with a unique ID to facilitate monitoring. The master controller is designed to pass on all legitimate network update requests and their execution results to the auditor-controller. Then, the auditor-controller will re-execute the update requests and store the two results in an audit record together with the unique ID of the request. If the results match, then the network is secure. If the results do not match, it means some device is compromised. The algorithm proposed in this paper classifies network information updates in four categories: network update requests, network update results, update instructions received by switches, and switch state updates. With this information and the execution results stored by the auditor-controller, SDN-RDCD can determine if a controller or a switch is behaving abnormally and whether a man-in-the-middle attack is likely to have occurred.

Evaluation
To evaluate the effectiveness of SDN-RDCD, the authors execute two attacks on a simulated network: one controller hijacking attack and a switch hijacking attack. The controller and switch attack both successfully manage to infect their target devices, but SDN-RDCD detects each misbehaviour conducted by the compromised devices. The added overhead is relatively insignificant and does not hamper network performance. The overhead can be reduced even more by using idle slave controllers for the auditor role.

3.5 AIM-SDN: Attacking Information Mismanagement in SDN-datastores

In [11] the vulnerabilities arising from information mismanagement in SDN datastores and the steps needed to address these vulnerabilities are discussed.

Information Management in SDN
SDN stores data in three different datastores classified into three categories: control/configuration data, inventory/operational data, and management data. The control/configuration datastore is where applications and services store flow rules, access control services, and quality of service criteria, amongst other information. This information is dynamically accessed and needs fast response times to ensure that the information remains up to date for other applications.

The inventory/operational datastore contains essential information about the current network view, like topology, runtime state and traffic statistics. This information is obtained through southbound plugins and cannot, and applications or services cannot modify it. This information must remain up to date as it gives all the applications their current view of the network, and incorrect information could, for example, lead to a load-balancer excluding a switch from the topology for no reason or a firewall giving access to an unauthorized host.

The management datastore contains management information like users, authorizations, and groups.

Information Mismanagement
The current design of SDN is problematic because it allows for information mismanagement This paper identifies three information management design flaws that can lead to an inconsistent network state: information disparity, blurred responsibilities, and unreliable service chaining.

- Information disparity: SDN applications use the NBI to send instructions to the controller, which stores these state changes in its control datastore. The controller services then apply the changes to the network, after which the controller's operational datastore updates itself by monitoring the network state. It is easy to imagine that the communication between these two components cannot be flawless, especially when considering link and device failures.
- Blurred responsibilities: when a rule is added to a network by a user, this rule must then be updated in an application, which then updates the control datastore. After that, the controller must implement the rule in the network, and finally, the operational datastore updates its state based on the network change. The issue in this process is that there is no clear ownership of the rule and that it is hard to decide who can delete it or who should re-execute it if it fails. This problem can lead to rules not being added or added twice, which can lead to inconsistent network states.
- Unreliable Service Chaining: if an application requests a series of network changes, it expects those changes to be executed in order. However, the current design of the datastores lacks synchronization and can not ensure a chained sequence of actions, which can again cause an inconsistent network state.

Network state consistency is one of SDN's main security challenges, even more so when distributed controllers are used.

Vulnerabilities and Proposed Attacks

To detect the vulnerabilities arising from these data stores and to develop attacks, the authors of this paper have developed a black-box fuzzing tool. This tool will send random network requests to a list of services in the network. Then, it uses the responses to analyze whether the service it communicates with is a datastore and what kind of datastore it is.

The vulnerabilities discovered by this tool have led the authors to discover a multitude of attacks that allow them to cause memory overflows leading to denial-of-service, as well as infiltrating the network to permit unauthorized and undesired traffic.

Mitigation strategies

To prevent an overflow of data in the NBI, the authors propose limiting the amount of configuration that external applications can add by using a rate-limiting proxy that also monitors suspicious amounts of traffic.

The paper also proposes a system clock to help avoid the vulnerabilities arising from the lack of synchronization between the control and operational datastores. This way, obsolete network configurations can be pruned using a timeout value, and applications can ascertain the age of a specific configuration.

Finally, the paper proposes to track ownership of configurations using metadata to mitigate the "blurred responsibilities" issue. For example, if an application adds a configuration, it owns that configuration, but if a service modifies it without external interaction, the ownership will be transferred to that service.

3.6 Attacking the Brain: Races in the SDN Control Plane

In [12] a new attack against SDN networks is proposed; a state manipulation attack exploiting harmful race conditions in controllers. It also proposes a tool, CONGUARD, to help detect these harmful race conditions.

Race Conditions in SDN

Race conditions occur when a system tries to perform two operations on the same data simultaneously. Simultaneous operations can lead to the system modifying data in ways both operations do not expect and ultimately to bugs. SDN's asynchronism leads to many race conditions in the control plane. In theory, these race conditions are not harmful since mutual exclusion synchronizations should protect them. However, due to oversights and logical flaws by programmers, many of these race conditions can be harmful and exploited by adversaries.

Exploiting harmful race conditions is not easy because it is hard to determine which race conditions are harmful and benign. Furthermore, an attacker must also be able to trigger these harmful conditions to exploit them without access to the controller.

Threat Model

The authors assume that an attacker is unable to compromise controllers, switches, and applications and is also unable to exploit SDN protocols and links between devices. In short, this means that the only entry point an attacker has is a compromised host.

CONGUARD

To detect harmful race conditions, the authors propose a tool called CONGUARD. It does this in two phases: first, it tries to detect harmful race conditions and then attempts to trigger them by remotely injecting network events with the proper timing.

Finding the harmful race conditions is done by first finding race conditions and then determining which race conditions are harmful. Detecting race conditions is done by generating network events and tracing their execution to determine if two race operations in a shared state occur. *Adversarial state racing* is used to determine which race conditions are harmful. *Adversarial state racing* is a technique developed by the authors which relies on the observation that harmful race conditions often arise from two operations on shared network states that are not commutative. However, controlling the scheduler allows forcing these operations to execute in the wrong order and introducing, for example, delays. Upon observation of an erroneous state, a harmful race condition is found.

It is difficult to trigger these conditions using only external network events because the amount of possible combinations of schedules is enormous. However, the authors devise a clever strategy of tracing back harmful race conditions with the control flow graph to an external operation. The attack strategy is based on the timing of requests to the controller and rapidly flipping connection states. If the attacker manages to time an update request with the trigger event correctly, a harmful race condition can be triggered. However, if this is impossible, this race condition can not be triggered by an adversary limited by the defined threat model.

Defense Schemes
The paper also proposes three defence schemes to counter the vulnerabilities revealed by CONGUARD. The first idea is to add safety checks to avoid concurrency violations. These safety state checks have been successfully used to patch 12 vulnerabilities discovered by CONGUARD. The second idea is to guarantee deterministic execution runtimes of state operations. Finally, it proposes sanitation of external events. This would, for example, ensure that an external host cannot flip connection states frequently in a short time.

Evaluation
CONGUARD allowed the authors to find 15 previously unknown vulnerabilities in three mainstream controllers that allow an adversary to cause system crashes, switch connection disruptions, service disruptions and service chain interference. CONGUARD completed its analysis in less than 2 minutes for all three controllers. Furthermore, CONGUARD was the only detector to detect harmful race conditions compared to other state-of-the-art race detectors.

4 Limitations and comparison of solutions

This section contains a comparison of solutions and an analysis of their effectiveness in mitigating SDN vulnerabilities.

4.1 Controller placement security limitations: comparing and analyzing [8] and [9]

The papers concerning controller placement discussed in this survey have a different focus; the first one is dedicated to mitigating link failure, while the second one is focused on mitigating controller failure. The difference in focus makes them complicated to compare, but it is interesting to analyze whether, when used together, they provide an even more secure control plane:

- The placement algorithm for mitigating multi-link failure from [8] introduces no significant latency overhead while still surviving a majority of failure scenarios, especially with a larger number of used controllers.
- The BFT placement scheme from [9] also boasts a relatively low worst-case time complexity for mapping and remapping operations, while the mappings it proposes are nearly optimal.
- The worst-case time complexities of both algorithms are nearly similar, implying that using both in a network will not cause significant overhead.

These two algorithms can be combined quite well. The multi-link failure produces a mapping of controllers, while the BFT algorithm produces a mapping of switches to controllers. When a remapping occurs for the BFT algorithm, this does not affect the mapping produced by the multi-link failure algorithm. This means that the placement scheme produced by combining the algorithms will produce a Byzantine fault-tolerant and multi-link failure resilient controller mapping for a low overhead cost.

4.2 Northbound Interface security limitations: comparing and analyzing [7] and [11]

The two papers discussing NBI security both propose an attack, but only one of them proposes a defence to their attack. This section contains an analysis of the shortcomings of both attacks. Furthermore, it will determine how effective the proposed defence in [7] is in mitigating both attacks. Both the cross-app poisoning attack proposed in [7] and the datastore attacks proposed in [11] use the NBI to poison information in the control plane or to perform black-box fuzzing to find datastores and seek to exploit them.

- The advantage of both these attacks is that they can be performed on open source and commercial controllers since they do not need access to the controller's source code to detect vulnerabilities.
- CAP can circumvent RBAC defences.
- The black box fuzzing tool used in [11] can not circumvent RBAC defences since it requires multiple different network requests that it would not have access to without the right role.

This is not to say that the vulnerabilities revealed in [11] are not relevant, and the designed fuzzing tool can be helpful for developers to test their networks. However, in a scenario where a network has an RBAC defence, it is not harmful.

The defence proposed in [7], ProvSDN has advantages and disadvantages:

- ProvSDN can detect cross-app poisoning attacks
- Due to its use of a provenance graph and of IFC to decide whether a request made by an application can be executed, it is likely that it will also refuse the function calls made by the black box fuzzing tool from [11].
- ProvSDN produces high overhead when it is used. However, it is important to note that this overhead only occurs with control plane state changes.

A developer designing an SDN network not interested in securing it from NBI attacks or ensuring that the applications used in the network are secure does not need ProvSDN. In contrast, ProvSDN could be a beneficial security addition for networks that regularly use third-party applications.

4.3 Controller failure security issues: comparing and analyzing [12] and [10]

Regarding controller failure, two papers have been discussed, one defence, SDN-RDCD and one tool facilitating the discovery of new attacks, CONGUARD. SDN-RDCD is an adequate defence, but not against CONGUARD, so it is challenging to compare these two algorithms. It is crucial to consider both the limitations and overhead CONGUARD and SDN-RDCD cause to determine whether the combination is an effective strategy:

- CONGUARD is an effective tool for discovering harmful race conditions in an SDN network. It takes two minutes to run, but it needs access to the source code of the used controller, making it a very ineffective tool for attacking commercial controllers. However, it is an extremely effective tool for developers wishing to detect harmful race conditions in their networks.
- SDN-RDCD uses an auditor-controller to verify network interactions and log network state changes at a relatively low overhead cost. Furthermore, in the experiments executed by the authors of [10], SDN-RDCD manages to detect every compromised network device after an attack.

To effectively combine these two tools, it would be wise to use them differently. For example, CONGUARD could be used every time a new network version is deployed to verify that no harmful race conditions have been inadvertently added. In contrast, SDN-RDCD could be continually running to detect attackers compromising devices.

5 Responsible Research

This paper has created a survey of cybersecurity vulnerabilities of a new networking paradigm, SDN. Summarising all these vulnerabilities in one paper can make the work of a malicious actor easier, should he wish to exploit controller-related exploits in SDN networks. However, the vulnerabilities that are analysed also come with proposed defences against them. Therefore, this paper can also be useful for SDN designers who want to improve their networks' security. Furthermore, the SDN architecture is still in an early phase of its life cycle. It is still subject to many vulnerabilities that must be addressed before it can be widely deployed. Papers like this one that point out those vulnerabilities and state-of-the-art solutions mitigating these are critical towards developing SDN further and making it ready for the real world.

In works focusing on surveying other research, it is essential to ensure that no plagiarism is conducted. Therefore, this work has tried to provide correct references for all the ideas it proposes and to avoid copying ideas from other research work.

6 Discussion and Future Research

Controller placement, controller failure mitigation, and northbound interface security are critical security issues for SDN. The papers discussed in this survey propose either solutions to defend against known vulnerabilities or new vulnerabilities that need further research. The proposed solutions are all reasonably practical and induce relatively little overhead. However, when all combined, that overhead can significantly impact network latency. One of the goals of the SDN design paradigm is to improve network speed, and adding a multitude of defences to a network that reduces the network speed ultimately defeats the purpose of SDN. Therefore, when designing a network using SDN, it is essential to assess what threats it will face, what capabilities an attacker will most likely have, and to cherry-pick the needed defences to avoid excessive speed reductions. When using SDN in a data centre that does not use third-party applications, for example, it is not beneficial to use ProvSDN.

As well as most solutions discussed in this survey work to tackle controller-related security issues, the control plane state consistency remains a significant concern, in particular when the number of controllers in a network increases [11]. The problem with the control plane state consistency is that it results directly from SDN's design, and ensuring consistency without drastically changing the design of SDN is a significant challenge. Some solutions proposed in this survey can prove quite effective in mitigating these issues, but new security issues will continue to appear.

Future research should improve the control plane state consistency and make it possible to use multiple defences in a network without excessively impacting network performance.

Network consistency

Improving consistency can be done directly in the design of SDN itself or by designing tools that can significantly mitigate the impacts of control plane inconsistency. While it is possible to redesign SDN to have a consistent control plane state, it will significantly impact its efficiency, as the parallel processing causing the inconsistencies are a significant reason that allows SDN to be so efficient.

One approach towards alleviating the impact of network state inconsistency could involve the detection of inconsistencies. Detecting inconsistencies between distributed controllers and their datastores could be done using a central database in the control plane. This database has read access to all the datastores and would use a consensus algorithm to see the most common network state among the controllers and select that state as the general state. Then controllers that do not conform to that state after a certain amount of time would have to be re-synchronized, and all network actions taken cancelled while warning applications that their actions may not have been completed. This central database would be closed for writes to everyone but owners of the network to keep it secure.

In a simple SDN network with three controllers, this would involve one database connected to both datastores of all controllers. When the database detects an inconsistency between controllers, it chooses the state occurring in most controllers as the general state. The divergent controller is then reset. Furthermore, network applications are warned that instructions have not been propagated through the affected part of the network and that they might need to re-execute them.

This approach has the advantage of not requiring a redesign of the control plane and being relatively simple to implement. However, it also has disadvantages:

- Some network data could be lost.
- Bugs could occur during re-synchronization
- New vulnerabilities could be introduced

Using Joint Defences

Research should also focus on making defences capable of joint use. A significant amount of defences have been proposed in the past decade, all of them addressing some area of

SDN security. Future research could focus on determining the best way of using these defences together and if it is possible to combine them in a network smartly. Without smartly combining solutions, networks will experience excessive amounts of performance overhead.

7 Conclusion

This work aimed to survey state-of-the-art attacks and defences for controller-related security risks and vulnerabilities in Software Defined Networking. Many papers have been thoroughly analyzed to provide an insight into the novel approaches they propose. These approaches include Byzantine Fault Tolerant and multi-link failure resilient controller placement algorithms, cross-app poisoning and information mismanagement attacks exploiting the northbound interface, a defence against northbound interface attacks using provenance, an algorithm to mitigate controller failure as well as attacks using harmful race conditions in the control plane to cause crashes and disruptions. The limitations of these approaches, as well as SDN security as a whole, have been discussed. Furthermore, it proposes future research directions, including an idea for improving network state consistency. This survey is limited by the number of papers it has used but provides an interesting overview of state-of-the-art research into controller-related SDN cybersecurity.

Widely deploying SDN outside of current specific use cases will remain a challenge until these security issues can be solved. However, SDN is a design paradigm that is still relatively new, and an incredible amount of progress can be made in the upcoming years.

References

[1] 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*, vol. 103, no. 1, pp. 14–76, 2014.

[2] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623–654, 2015.

[3] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 473–478, 2012.

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

[5] P. Vizarreta, P. Heegaard, B. Helvik, W. Kellerer, and C. M. Machuca, "Characterization of failure dynamics in sdn controllers," in *2017 9th International Workshop on Resilient Networks Design and Modeling (RNDM)*, pp. 1–7, IEEE, 2017.

[6] T. Das, V. Sridharan, and M. Gurusamy, "A survey on controller placement in sdn," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 472–503, 2019.

[7] B. E. Ujcich, S. Jero, A. Edmundson, Q. Wang, R. Skowyra, J. Landry, A. Bates, W. H. Sanders, C. Nita-Rotaru, and H. Okhravi, "Cross-app poisoning in software-defined networking," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pp. 648–663, 2018.

[8] S. Yang, L. Cui, Z. Chen, and W. Xiao, "An efficient approach to robust sdn controller placement for security," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1669–1682, 2020.

[9] P. M. Mohan, T. Truong-Huu, and M. Gurusamy, "Byzantine-resilient controller mapping and remapping in software defined networks," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 2714–2729, 2020.

[10] H. Zhou, C. Wu, C. Yang, P. Wang, Q. Yang, Z. Lu, and Q. Cheng, "Sdn-rdcd: A real-time and reliable method for detecting compromised sdn devices," *IEEE/ACM Transactions on Networking*, vol. 26, no. 5, pp. 2048–2061, 2018.

[11] V. H. Dixit, A. Doupé, Y. Shoshitaishvili, Z. Zhao, and G.-J. Ahn, "Aim-sdn: Attacking information mismanagement in sdn-datastores," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 664–676, 2018.

[12] L. Xu, J. Huang, S. Hong, J. Zhang, and G. Gu, "Attacking the brain: Races in the {SDN} control plane," in *26th USENIX Security Symposium (USENIX Security 17)*, pp. 451–468, 2017.

[13] R. S. Sandhu, "Role-based access control," in *Advances in computers*, vol. 46, pp. 237–286, Elsevier, 1998.

[14] G. Aceto, A. Botta, P. Marchetta, V. Persico, and A. Pescapé, "A comprehensive survey on internet outages," *Journal of Network and Computer Applications*, vol. 113, pp. 36–63, 2018.

[15] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[16] "Internet2 open science, scholarship and services exchange." https://www.internet2.edu/productsservices/advanced-networking/layer-2-services/. Accessed: 2022-06-16.

[17] "China education and research network 2 (cernet2)." http://www.cernet2.edu.cn/. Accessed: 2022-06-16.

[18] H. Li, P. Li, S. Guo, and A. Nayak, "Byzantine-resilient secure software-defined networks with multiple controllers in cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 4, pp. 436–447, 2014.

[19] K. ElDefrawy and T. Kaczmarek, "Byzantine fault tolerant software-defined networking (sdn) controllers," in *2016 IEEE 40th annual computer software and applications conference (COMPSAC)*, vol. 2, pp. 208–213, IEEE, 2016.