P4Runtime Security and Man-in-the-Middle Attacks

Areti Katsikis
Supervisor: Chenxing Ji
Responsible Professor: Fernando Kuipers
EEMCS, Delft University of Technology, The Netherlands

A Report Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

22-6-2022

Abstract

In software defined networking a controller can control where the data-plane routes packets to. Programmable data-planes make networks even more flexible, as the algorithms on the data-plane can be updated. The P4 programming language can be used to program data-planes, and the P4Runtime data-plane API can be used for controller to data-plane communication. The possibility of man-in-the-middle attacks when using P4Runtime was investigated. Man-in-the-middle attacks are possible either between the controller and data-plane, or between two hosts on the network. A virtual network in Mininet was used to try and demonstrate the difference between secure and insecure channels in these two scenarios. A malicious controller can take control of a switch in order to use it for man-in-the-middle attacks when the P4Runtime channel is insecure, but not in a secure channel. The man-in-the-middle attack between the controller and switch was not achieved due to the switches in Mininet only running on localhost and not being able to run the controller in-band. It was concluded that it is indeed recommended to only use secure P4Runtime channels, and possible extensions to this research could be to attempt the same experiment using a different setup or to research the effects that a successful man-in-the-middle attack can have.

1 Introduction

Programmable data-planes are not limited to vendor-provided algorithms because they can be reprogrammed. P4 (Programming Protocol-Independent Packet Processors) [1] is a programming language that is used to program data-planes and create custom algorithms for them. Data-plane APIs let the controller directly control the data-plane during runtime. For P4 this means controlling Match Action Tables (MATs), externs, and packet I/O between the control plane and the P4 target. MATs map packet headers to actions and externs are methods not included in the core P4 functionality [2].

P4Runtime is a data-plane API that lets a controller and data-plane communicate using RPCs (Remote Procedure Calls) [2]. The P4 target that hosts the data-plane is required to implement a gRPC server, and the controller a gRPC client. P4Runtime uses protocol buffers (protobuf) [3], a format for serializing structured data, to format messages sent through the gRPC channel. P4Runtime can be used to either control a data-plane locally, if the controller is on the same machine as the P4 target, or remotely [4].

When the controller is running remotely there is a larger attack surface and a higher danger that the P4Runtime communication channel could be corrupted. If the channel of communication between the control-plane and data-plane is compromised then so are the switches that the controller controls. One attack that P4Runtime is vulnerable to is the man-in-the-middle attack, as mentioned by Agape et al. [5], who look at the attack surface of networks using P4 switches and P4Runtime on the controller. The section related to P4Runtime were taken as a starting point for looks at man-in-the-middle attacks between the P4Runtime engine on the P4 target and the client running on the controller in greater detail.

The motivation for this work is that while there is research on related topics, there is a lack on P4Runtime and man-in-the-middle attacks combined. Related research covers how P4 can be used to improve network security [6] and protecting the links between P4 switches from man-in-the-middle attacks [7]. Research on the communication between control and data-planes in SDN includes works such as Brooks and Yang's paper on man-in-the-middle attacks on the OpenDayLight controller [8] and Network Topology Poisoning Attacks in OpenFlow controllers that can pave the way for man-in-the-middle attacks [9]. As far as the author is aware, there is nothing specifically on the security of the P4Runtime communication channel against man-in-the-middle attacks. Therefore the communication safety of P4Runtime with regard to man-in-the-middle attacks merits investigation.

The main research question of this work is: Can the communication channel between the client and the P4Runtime engine be corrupted? To answer this question, how the P4Runtime engine and client communicate and what attacks can affect P4Runtime were considered, with a focus on man-in-the-middle attacks.

The main conclusions of this research are that as is mentioned in the P4Runtime Specification [4] and the work of Agape et al. [5], the communication channel is vulnerable if an insecure gRPC connection is used. In the case of an insecure connection an attacker can eavesdrop on the communication between the controller and data-plane or take control of that data-plane.

The paper answers the research question in the following way. Section 2 describes the methodology of the research and Section 3 discusses the security of P4Runtime in more detail. Section 4 describes the experimental setup and the results. A section on responsible research is discussed in Section 5. Section 6 discusses the results while Section 7 handles the conclusions and recommendations for future work on the subject.

2 Methodology

The methodology described in this section gives an overview of how the research for this paper was conducted and the plans for performing a man-in-the-middle attack between a P4Runtime controller and switch. For a more in depth description of the experiment performed see Section 4 Experimental Setup and Results.

In order to answer the research question, how the

P4Runtime engine and client communicate was investigated by reading the P4Runtime [4], gRPC [10], and protobuf [11] documentation. As mentioned in Section 1 Introduction, P4Runtime uses gRPC channels sending messages in protobuf format.

The existing research involving P4Runtime or man-in-the-middle attacks mentioned in the Introduction (Section 1) was read to identify where further research should continue. The work of Agape et al. [5] was taken as a starting point as it describes man-in-the-middle attacks and channel flooding as the main vulnerabilities of P4Runtime.

To investigate the security of the P4Runtime channel with regard to man-in-the-middle attacks, the attack was attempted in practice on both an insecure P4Runtime channel and one secured using TLS (Transport Layer Security), which is considered best practices [12]. Mininet, a network emulator that can create virtual networks [13], was used to simulate a network and the bmv2 model, a reference P4 software switch [14], was used to simulate the P4 switches in that network. Using Wireshark [15], the contents of the packets being sent could be inspected. Bettercap [16], a tool for man-in-the-middle attacks, was selected to perform the man-in-the-middle attacks because it includes tools for ARP poisoning and other man-in-the-middle approaches. ARP poisoning, in which spoofed Address Resolution Protocol (ARP) messages can be used to facilitate man-in-the-middle attacks, can affect SDN networks and therefore various mitigation strategies have been discussed [17]. Seeing as research has been done on adding ARP to P4 switches [18], it is plausible that switches using P4Runtime to communicate with controllers would be vulnerable to ARP poisoning.

Two different possible scenarios for man-in-the-middle attacks using P4Runtime were considered. The first scenario is when a malicious entity gets in the middle of the P4Runtime communication between the original controller and the switch and impersonates that controller to send commands. The second scenario is when a malicious controller takes control of the data-plane and makes that switch the man-in-the-middle, targeting the traffic that passes through it. In both cases the attacker gains control of a switch and can do almost anything on it, including updating the rules on the switch to inspect the messages or change the routing tables.

3 Man-in-the-Middle Attacks and P4Runtime

In P4Runtime multiple controllers can be connected to one P4 target, however only one can have write access to the entities on the target. This controller is the primary controller. All the other controllers can only have read access to the entities on the P4 target [4]. Thus, being the primary controller means having complete control over the switch and being able to modify its functionality. The primary controller can load a new P4 program onto a P4 target by installing a new `ForwardingPipelineConfig` file using the `SetForwardingPipelingConfig` RPC.

As stated in the P4Runtime specification [4], the data-plane identifies controllers based on a 3-tuple composed of a `device_id`, a `role`, and an `election_id`. The `device_id` identifies the P4 target and the `role` determines which entities on the data-plane the controller has access to, the default being access to everything. The controller with the highest `election_id` is the primary controller.

The primary controller is determined through the process of arbitration. When a new controller wants to connect it sends a `MasterArbitrationUpdate` request, which contains the `device_id`, `role`, and `election_id`. If the new controller has a higher `election_id` than the previous primary controller then it will become the primary controller.

P4Runtime trusts the messages that it receives through its gRPC client [4]. This means that if no authentication mechanism is enabled in gRPC, the switch will accept any connection request as long as no duplicate `device_id`, `role`, `election_id` combination is used. In this way a malicious controller can provide an `election_id` higher than the previously highest recorded one and become the new primary controller.

In a network of P4 switches and controllers running P4Runtime, there are two main ways for man-in-the-middle attacks to be set up. As described in Section 2 Methodology, the man-in-the-middle can be between the controller and switch, or between hosts on the network.

In the first scenario, there is a malicious entity monitoring the communication channel between the controller and the switch and it can perform man-in-the-middle attacks.

In an insecure connection where the messages are not encrypted, it is easy for the entity in the middle to read or modify the messages, or impersonate the controller to give commands to the data-plane. To impersonate the controller, the attacker needs to obtain the `device_id`, `role`, and `election_id` that the controller is using to identify itself to the data-plane.

In a secure channel on the other hand, it is more difficult for a man-in-the-middle attack to take place. gRPC supports multiple authentication mechanisms, such as mutual TLS [4]. If the communication is encrypted then an attacker cannot read the `device_id`, `role`, and `election_id` from the messages. While the attacker may be able to guess these, if the P4 target is authenticating connection requests from controllers then the attacker will not be able to connect.

In the second scenario a malicious controller can establish itself as the primary controller to make the switch the man-in-the-middle between different hosts on the network.

If the P4Runtime communication channel between the controller and the switch is insecure, a malicious controller can easily become the primary one. The only things the malicious controller needs to know to establish a connection are the address and port to connect to, and the `device_id` that identifies the data-plane that it wants to control, which can be obtained by sniffing the unencrypted traffic.

When the communication channel is secure, which is the most likely, a malicious controller cannot simply connect to a data plane because only known controllers are allowed to connect. Using an authentication mechanism between the controller and data-plane prevents malicious controllers from becoming the primary controller.

4 Experimental Setup and Results

As described in Section 2 Methodology, the P4Runtime communication channel was studied in a virtual network in Mininet using bmv2 switches on an Ubuntu 18.04.1 virtual machine. The starting point for the experimentation was the P4Runtime tutorial exercise of the p4lang repository on GitHub [19]. This exercise uses a topology of three switches, depicted as circles labeled s1, s2, and s3, connected in a triangle. Each switch is connected to one host, depicted as rectangles labeled h1, h2, and h3, as seen below in Figure 1. Because the topology was already set up to use P4Runtime for the controller-switch communication, it was used for the experimental setup as well, since only one switch, two hosts, and one controller would be needed for the scenarios described in the methodology. The p4lang exercise uses an insecure channel by default. To use a secure channel a gRPC server using a secure port needs to be created while initializing the switches, in the `p4runtime_switch.py` file in the utils.

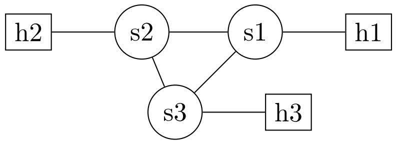

Figure 1: Topology of network

The hosts are assigned their own IP addresses while the switches run on localhost. In the tutorial exercise the controller is run from localhost as well. Attempts were made to run the controller with a different IP address from the switches by either running it as an inband controller on one of the hosts or as a remote controller. However, due to difficulties with achieving this in Mininet and time constraints this was not successful and all experiments were done running the controllers from localhost, using the topology pictured below in Figure 2.

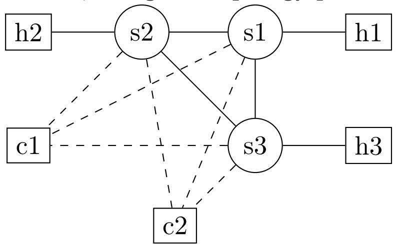

Figure 2: Topology of network with identity competing controllers

For the scenario where a malicious controller becomes the primary controller when the data-plane accepts insecure connections, only the `election_id` has to be updated, provided the `device_id` is known, as well as the address and port number of the P4Runtime Engine on the data-plane. The default `role` can be used to request access to everything on the data plane.

To create the malicious controller, the default controller in the example was altered to use a modified version of the utils provided. All methods that make use of the `election_id` were adjusted to use a higher value for the `election_id` instead. The value 900000 was chosen arbitrarily, as the example controller used the value 1. The methods that had to be modified to use the higher `election_id` where `MasterArbitrationUpdate` used for selecting the primary controller, `SetForwardingPipelineConfig` used to load a new P4 program on the switch, `WriteTableEntry` used to update the Match Action Table, and `WritePREEntry` used for updating the packet replication engine.

While the original controller was running on localhost in one terminal, the modified controller was run from another. This resulted in the modified controller becoming the primary controller. The original controller could not regain control of the switch afterwards because it had a lower `election_id`. This demonstrates that when the data-plane accepts insecure communication channels, a malicious controller can easily become the primary controller by using a very high `election_id`, and would then be able to control the data plane however it wants, including for performing man-in-the-middle attacks for traffic passing through that switch.

To perform man-in-the-middle attacks between the controller and a switch, the same setup using the p4lang tutorial example was used. Using Wireshark, the traffic on the virtual network was monitored to detect which ports were being used for the communication between the controller and the switch. While the original idea was to perform a man-in-the-middle attack using bettercap 2.32.0 with the switches running on localhost and the controller running on a host in order to have a different IP address, this was not achieved.

For the man-in-the-middle attack between the controller and the switch, the topology shown on the following page in Figure 3 was supposed to be used. The malicious entity performing the man-in-the-middle attack between switch s3 and host h3 containing the controller is depicted as h4.

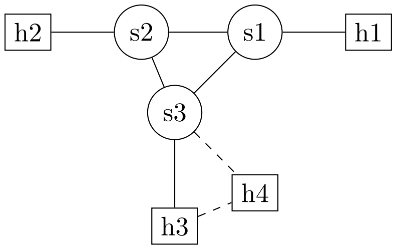

Figure 3: Topology of network for man-in-the-middle attack between controller and switch

This setup was not achieved because the python script for the controller could not be run on a host in the Mininet network and then connect to the switches running on localhost. Instead bettercap was run on localhost to sniff the messages sent from the controller on localhost to the switches. The performance of a man-in-the-middle attack was not successful because the topology used did not use ARP, making an ARP poisoning attack impossible. In the example controller the addresses of the switches were hardcoded.

5 Responsible Research

Researching the security of the P4Runtime communication channel is important for the security of networks in which the controllers use P4Runtime to communicate with the switches. The research was performed in an ethical and reproducible way.

The main ethical concerns related to looking into the security of communication channels is how any vulnerabilities that are discovered are handled. It is not ethical to simply disclose vulnerabilities to the public or sell them to someone other than the maintainer because this puts any systems using that software at risk. If any vulnerability was found it would have been reported to the P4.org API Working Group that works on the P4Runtime specification so that it could be addressed. However, no new vulnerability was found and therefore this ethical dilemma never arose.

The methods used in this research are reproducible. Section 2 describes the methodology used and Section 4 gives a more detailed explanation of the experimental setup and the exact steps taken in such a way that the results can be reproduced.

6 Discussion

The main results were that in the case of an insecure P4Runtime communication channel a malicious controller can become the primary controller and seize control of a switch, as was described in Section 4 Experimental Setup and Results. This was the expected result after reading the P4Runtime specification, and reemphasizes the statement that securing the connection using TLS is best practice [12] and prevents man-in-the-middle attacks [5].

To answer the research question, the communication channel between the P4Runtime engine and a P4Runtime client can be corrupted, especially when the communication channel is insecure. This agrees with the work of Martinez-Yelmo et al. [18], who advocates keeping the control-plane close to the data-plane to decrease the attack surface.

To reflect upon the progress made, it was unfortunate that only the scenario in which a malicious controller takes control of a switch to use that switch as the man-in-the-middle between two hosts communicating over that switch could be performed. In order to perform a man-in-the-middle attack between a controller and a switch the communication between them would have to be rerouted through a malicious entity, and doing this between two different ports on localhost was not possible within the time frame. This is because much time and effort was put into trying to run the controller on a host in Mininet.

The conclusion that some form of authentication between the controller and switch should take place agrees with what is stated in the P4Runtime Specification [4] and related literature [5], [12].

7 Conclusions and Future Work

To answer the research question of if the communication channel between the P4Runtime Engine and the client can potentially be corrupted, the sub-questions of how the P4Runtime Engine and client communicate, what attacks are possible, and how they can be executed and/or prevented were considered.

Man-in-the-middle attacks and channel flooding are the main attacks for P4Runtime, according to Agape et al. [5]. The work of Agape et al. was extended to demonstrate how to seize control of a switch using an insecure gRPC connection to become the primary controller, allowing for that switch to be used in man-in-the-middle attacks on the traffic passing through that switch.

A malicious controller can take control of a switch by sending a very high `election_id` in its requests to the data-plane. When the data-plane uses insecure connections the malicious controller only needs to know the address of the data-plane, the port used by the gRPC client, and the `device_id` of the target. With secure connections this is more difficult because if mutual TLS is used then the controller and switch authenticate each other and the switch will not accept a connection from a different controller.

The second scenario for man-in-the-middle attacks using the P4Runtime communication channel is when an entity between the controller and switch performs the attack. The planning for the experiment was to run the controller in-band on one of the hosts in the Mininet network, however this was not successful, and may not even be possible. An improvement to the setup may be to use a different program other than Mininet to simulate the network or to use a different controller, such as the ONOS controller [20], which supports P4Runtime, instead of the lightweight python script that was used.

Further research to build upon the work done could be to investigate the effects of man-in-the-middle attacks by a compromised switch on the network and detection strategies for this.

Another possibility for future research could be to look at the possible use cases of P4Runtime controllers in real networks, because the network architecture has an effect on the viability of man-in-the-middle attacks. In the case of a network where a controller uses ARP to relate the IP address of a switch to its MAC address, ARP poisoning could be used to set up a man-in-the-middle attack. In a network where ARP is not used and the controller connects to a switch through a fixed link, this approach is not possible.

References

[1] The P4 Language Consortium, "P4 16 language specification," 2017. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html

[2] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, "A survey on data plane programming with P4: fundamentals, advances, and applied research," *CoRR*, vol. abs/2101.10632, 2021. [Online]. Available: https://arxiv.org/abs/2101.10632

[3] "Protocol buffers," 2022. [Online]. Available: https://github.com/protocolbuffers/protobuf

[4] "P4runtime specification." [Online]. Available: https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html

[5] A. Agape, M. C. Danceanu, R. R. Hansen, and S. Schmid, "Charting the security landscape of programmable dataplanes," *CoRR*, vol. abs/1807.00128, 2018. [Online]. Available: http://arxiv.org/abs/1807.00128

[6] Y. Gao, Z. Wang, and S.-B. Tsai, "A review of p4 programmable data planes for network security," *Mob. Inf. Syst.*, vol. 2021, jan 2021. [Online]. Available: https://doi.org/10.1155/2021/1257046

[7] F. Hauser, M. Schmidt, M. Häberle, and M. Menth, "P4-macsec: Dynamic topology monitoring and data layer protection with macsec in p4-based sdn," *IEEE Access*, vol. 8, pp. 58 845–58 858, 2020.

[8] M. Brooks and B. Yang, "A man-in-the-middle attack against opendaylight sdn controller," in *Proceedings of the 4th Annual ACM Conference on Research in Information Technology*, ser. RIIT '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 45â49. [Online]. Available: https://doi.org/10.1145/2808062.2808073

[9] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning network visibility in software-defined networks: New attacks and countermeasures," in *NDSS*, vol. 15, 2015.

[10] "grpc documentation," 2022. [Online]. Available: https://grpc.io/docs/

[11] "Protocol buffers," 2022. [Online]. Available: https://developers.google.com/protocol-buffers

[12] C. Black and S. Scott-Hayward, "Adversarial exploitation of p4 data planes," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2021, pp. 508–514.

[13] "Mininet," 2021. [Online]. Available: https://github.com/mininet/mininet

[14] "Behavioural model (bmv2)," 2022. [Online]. Available: https://github.com/p4lang/behavioral-model

[15] "Wireshark," 2022. [Online]. Available: https://github.com/wireshark/wireshark

[16] "Bettercap," 2021. [Online]. Available: https://github.com/bettercap/bettercap

[17] Z. Shah and S. Cosgrove, "Mitigating arp cache poisoning attack in software-defined networking (sdn): A survey," *Electronics*, vol. 8, no. 10, p. 1095, Sep 2019. [Online]. Available: http://dx.doi.org/10.3390/electronics8101095

[18] I. Martinez-Yelmo, J. Alvarez-Horcajo, M. Briso-Montiano, D. Lopez-Pajares, and E. Rojas, "Arp-p4: Deep analysis of a hybrid sdn arp-path/p4runtime switch," *Telecommun. Syst.*, vol. 72, no. 4, p. 555â565, dec 2019. [Online]. Available: https://doi.org/10.1007/s11235-019-00588-2

[19] "P4 tutorial," 2022. [Online]. Available: https://github.com/p4lang/tutorials

[20] "Onos: Open network operating system," 2022. [Online]. Available: https://github.com/opennetworkinglab/onos