Cases for Including a Reference Monitor to SDN

Dimitrios Gkounis, Felix Klaedtke, Roberto Bifulco, Ghassan O. Karame
NEC Laboratories Europe, Germany – E-mail: <firstname.lastname>@neclab.eu

CCS Concepts

•Networks → Programmable networks; •Security and privacy → Access control;

1. INTRODUCTION

Network administrators consider misconfigurations to be the main source for network failures [6, 2]. In turn, a misconfiguration usually happens because the complexity of modern networks must still be mastered to a large extent with error-prone low-level manual interventions. Software-defined networking (SDN) provides the means to simplify and automate network operations. The SDN architecture comprises a logically centralized controller that provides a programmable logic to operate the network, abstractions, and services such as topology discovery or end-to-end connectivity. The services are accessed through so-called *north-bound interfaces* (NBIs), which can be used manually by network administrators or by network applications that run on top of the controller and automate network operation tasks.

Clean and high-level NBIs will reduce the number of network misconfigurations. They will however not completely prevent misconfigurations from happening. Furthermore, the impact of incorrectly using an NBI will be more severe, since the controller has access to the whole network infrastructure and a single high-level NBI command will result into multiple low-level actions that reconfigure the network's data plane. In addition, when a program automates network operations, misconfigurations can also be caused by software bugs within the implementation of the used NBI command. Note that while the NBIs may be simple, their implementation can be complex. Another source of misconfigurations can be the interplay of multiple network applications on top of the controller. For example, the applications can have competing objectives, or their direct or indirect interactions may trigger unexpected behavior. Of course, the mentioned problems are exacerbated when a misconfiguration is intentional, e.g., caused by a malicious subject. For instance, a compromised network application may harm the network by rerouting traffic.

SIGCOMM '16, August 22–26, 2016, Florianopolis, Brazil

DOI: http://dx.doi.org/10.1145/2934872.2959066

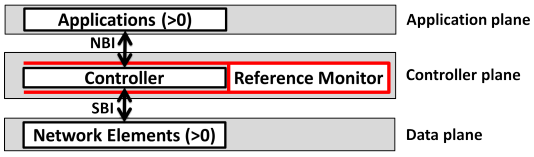

Figure 1: Extended SDN architecture

Several network verification techniques (e.g., [3]) have been proposed to detect misconfigurations of a network. Furthermore, checks at the interfaces between the controller and the network applications (e.g., [7]) have been proposed to protect the controller, for example, from malicious network applications. However, such techniques and checks do not prevent the controller from sending configuration commands that result in misconfigurations of the network's data plane. In [4], we proposed to include into the SDN architecture a *reference monitor*, i.e., a trustworthy component that must permit the controller's actions according to a given policy. Note that this reference monitor resembles a reference monitor of an operating system, i.e., a verifiable and tamper-proofed entity that enforces an access control policy to limit the ability of subjects to perform actions (e.g., write and read) on objects, i.e., system resources such as files and sockets. In general, the reference monitor guarantees that the activity of an intentionally or unintentionally misbehaving subject has limited effects.

In this work, we present a proof-of-concept implementation of the reference monitor for the state-of-the-art SDN controller ONOS [1], and show its effectiveness in protecting the network from misconfigurations caused by the interplay of applications and administrators, when using the ONOS' intent framework.

2. ARCHITECTURE

ONF's SDN architecture [5] specifies, in addition to the NBIs, *south-bound interfaces* (SBIs) for a network controller. SBIs are used to interact with the devices at the

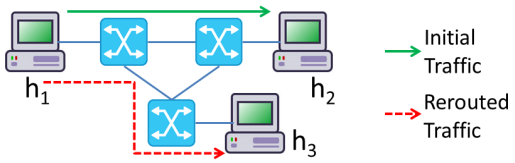

Figure 2: Network topology

data plane. We decided to take advantage of already defined SBIs, e.g., OpenFlow, to integrate the reference monitor into the SDN architecture (Fig. 1). In fact, the reference monitor enforces policy compliance for OpenFlow messages according to a given policy. In general, in addition to the SBI invocation, the reference monitor must usually receive information from the NBI-side of the controller, since SBI invocations typically do not carry information about the subject requesting the execution of a given action, while the NBI does.

For the implementation of our reference monitor for ONOS, our architectural design proved to be a good choice. In fact, our implementation required only minimal modifications to ONOS. In more detail, the ONOS controller is Java-based and utilizes the Java OSGi framework, which enables modules to be dynamically loaded at runtime. As such, the reference monitor has been implemented as a separate OSGi bundle. We only needed to modify ONOS' code related to the OpenFlow SBI. Here, we introduced two important changes. First, we made sure that the reference monitor is invoked for any OpenFlow message before it is sent. A message denied by the reference monitor is not sent and raises an exception. Second, we extended the SBI function calls to include information about the subject making the call. For this second point, we leveraged the concept of *application ID* that is already built in ONOS. That is, any ONOS application is assigned to an ID. Whenever a call to the NBI interface is performed, the corresponding application ID is kept during all the processing steps of the call, until the corresponding SBIs calls are performed.

3. USE CASES AND DEMONSTRATION

Our demo uses a Mininet's emulated network (Fig. 2) consisting of three switches and three hosts (h_1, h_2, and h_3). The switches are connected to each other, and each host is connected to one of the switches. We assume the following policy: connectivity should be provided between h_1 (the gateway) and h_2 (the server); h_3 (visiting hosts) can only obtain connectivity to h_1 (the gateway) and only after the network administrator grants it. Moreover, ONOS controls the network with two applications: (1) a forwarding application, which establishes connectivity between h_1 and h_2, and (2) a command-line interface (CLI) to submit connectivity intents.

Recall that ONOS' intent framework compiles *intents* into OpenFlow messages for configuring the switches. An intent is a high-level description of what the network should do. An intent may, e.g., specify connectivity between hosts or the redirection of a network flow to a network location. For our network, the network administrator may use the CLI to provide connectivity to h_3. However, since the CLI can be used to provide any type of connectivity intent, the CLI can also be used to redirect the traffic between h_1 and h_2 to h_3, which should not be permitted.

We perform two tests. In the first test, we run ONOS without the reference monitor, while for the second one, the reference monitor is activated. The provided policy only allows the CLI to configure connectivity between h_1 and h_3. In both tests, we use host h_1 to ping host h_2, showing that the forwarding application establishes connectivity between them (green arrow in Fig. 2). Using the CLI, we then submit an intent that redirects the network flows between h_1 and h_2 to h_3.

- When the reference monitor is not used, the traffic exchanged between hosts h_1 and h_2 gets redirected to host h_3 (red arrow in Fig. 2). Thus, we deviated from the intended usage of the CLI, which resulted in a policy violation.
- When the reference monitor is active, the submission of the connectivity intent does not change the packet flows. In fact, the reference monitor blocks the OpenFlow messages originating from the intent since they violate the given policy. The violation is reported in the form of warnings on the ONOS interface.

Another use case shows a subtle design flaw in the intent framework of ONOS (v. 1.3.0), which can lead to network misconfigurations. Suppose that the ONOS intent forwarding application and the CLI run on top of the controller. Moreover, suppose that both applications submit exactly the same intent. Both intents are successfully installed. However, ONOS recognizes that the rules generated for the second intent are identical to the ones installed for the first intent. Thus, ONOS avoids installing them at the switches.

If we use the CLI to withdraw one of the two installed intents, all the corresponding forwarding flow rules are removed from the switches. ONOS does not recognize that these flow rules also implement another intent that is still in use. Our reference monitor would block the OpenFlow messages for deleting the flow rules and report a policy violation.

Acknowledgments. This work received funding from the EU in the context of the H2020 projects VirtuWind and 5G-Ensure (grant agreements 671562 and 671648).

4. REFERENCES

[1] P. Berde et al. ONOS: towards an open, distributed SDN OS. *HotSDN* 2014.
[2] Juniper Networks. What's behind network downtime? www-935.ibm.com/services/au/gts/pdf/200249.pdf, 2008.
[3] P. Kazemian et al. Real time network policy checking using header space analysis. *NSDI* 2013.
[4] F. Klaedtke et al. Access control for SDN controllers. *HotSDN* 2014.
[5] ONF. SDN architecture overview. www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN-ARCH-Overview-1.1-11112014.02.pdf, 2014.
[6] J. Sherry et al. Making middleboxes someone else's problem: Network processing as a cloud service. *Comput. Commun. Rev.*, 42(4), 2012.
[7] S. Shin et al. Rosemary: A robust, secure, and high-performance network operating system. *CCS* 2014.