Demo: Leveraging SDN in Critical Infrastructures

Simon Buttgereit, Michael Rossberg, Michael Pfeiffer, Guenter Schaefer
Technische Universität Ilmenau, Germany
firstname.lastname@tu-ilmenau.de

***Abstract*—Recent developments in computer networks increased flexibility, making them more dynamic and programmable, e.g., by SDN and NFV. However, this also increased complexity and volatility of network components. This is a challenge for highly regulated environments such as critical infrastructure networks where certified components are used to guarantee security requirements of infrastructures, e.g., through mandatory filtering or encryption of network traffic. This demo paper presents a setup where programmable and volatile components are separated from trusted, and thus certified, components. In particular, programmable Network Operating Systems (NOSes) and SDN controllers are deployed to steer the network flows in a VPN overlay. Yet, these flexible components do not have to be included into a certification process.**

***Index Terms*—Software-defined Networks, Virtual Private Networks, SD-WAN, Slicing, Critical Infrastructure Networks**

I. Introduction

Over the last years, developments in networking have brought programmability and thus greater dynamics and flexibility through technologies such as Software-defined Networking (SDN), Network Function Virtualization (NFV), and Infrastructure as Code (IaC). Network operators benefit in several ways, including the ability to route traffic in a much more fine-grained manner, dynamic adaption to new requirements, and reduction of operating costs. However, these new software stacks come with huge code bases and fast agile development cycles [1], [2]. These characteristics are not necessarily bad. Nevertheless, they contradict with life cycles (certification) in critical infrastructures, so that the latter can currently not benefit from these developments. Since failures or malfunctions in critical infrastructures in government, medical, energy, and financial sectors may have significant impact on today's society, the associated networks require a high level of protection. Common ways of securing networks are protecting the data using cryptographic measures, firewalling/packet labeling, or physical isolation. Policies are enforced by small and static components, which get certified for deployment in critical infrastructures.

One way to integrate new programmable components into critical infrastructures could be to separate them from security-critical components and control their impact on them. In short, they are treated as *untrustworthy*. This paper proposes an architecture that enables deployment of a freely programmable control plane in security-critical network infrastructures without violating security guarantees. As a result, certification is only required for relatively few and slim components, which are referred to as *trustworthy* in the following.

II. Requirements and Related Work

Assuming *trusted* components to be certified, implies huge effort and high costs in development. Code, functions, and number of potential states must be as small and simple as possible. Thus it forms a solid base, and we assume in the following that these have no security flaws. This corresponds to the current approach for critical infrastructures. In contrast, *untrustworthy* components may fail or not follow protocols, i.e., by programming errors or even malicious activity. They can be freely programmable to support:

- *Open programming interfaces:* It is possible to deploy network applications at runtime, either to directly control the data plane or to integrate new control interfaces.
- *Fine-grained traffic engineering:* Applications can steer flows dynamically and explicitly block, allow, or modify packets based on individual protocol fields (if allowed by security policies).
- *Monitoring:* Applications can monitor traffic properties to react accordingly.

Orthogonal to the separation of trustworthy and untrustworthy, many scenarios also require a separation of different tenants or security levels, collectively referred to as *security groups*. The security properties of the overall architecture shall only be based on the integrity of certified components. We have identified the following security goals:

- *Data CIA:* It must be possible to restrict untrustworthy components in their ability to read, modify, and forge network packets within their security group to protect the data's confidentiality, integrity, and authenticity (CIA). Furthermore, it should be impossible for entities in other security groups or untrustworthy networks to read, modify, or forge network packets.
- *Control plane CIA:* Entities in different security groups or untrustworthy networks must not be able to analyze each other's traffic flows or to fake control plane commands.
- *Availability:* Actions of untrustworthy control plane components must not influence the availability of components in other security groups.

There are only a few approaches that explicitly aim to deploy a freely programmable control plane in critical network infrastructures. One of the basic concepts of OpenFlow [3], [4] is

The authors would like to thank secunet Security Networks AG for funding this research.

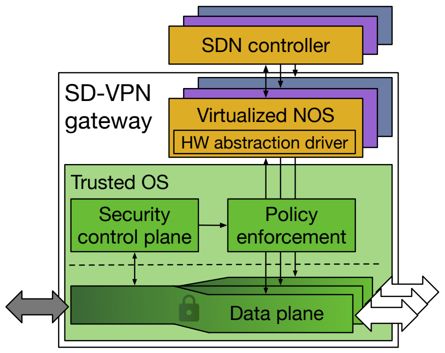

Fig. 1. SD-VPN gateway with three security groups with independent NOS and SDN controller.

an unrestricted exchange of network packets between control and data plane. Thus, even if additional slicing controllers are used to proxy all southbound communication [5], [6], OpenFlow cannot be used without violating the data CIA between data and control plane. HyPer4 [7], which slices a P4 data plane, is not freely programmable because it is only capable of running a small subset of P4 programs. Integrating an access control model into the controller [8]–[10] does help against malicious SDN applications, however, will not achieve data and control plane CIA, if vulnerabilities in the controller are leveraged [11]. A method, differing from securing SDN itself, is to make Virtual Private Networks (VPNs) more flexible [12]–[15]. These approaches push VPN into the direction of SD-WAN, however, do not address critical network infrastructures. Hence, the security requirements we stated are neither addressed nor fulfilled.

III. System Architecture

The core idea is to separate between freely programmable/untrustworthy and trusted components. To achieve this separation, we categorize all security policies that exist in the architecture into three different levels and assign their enforcement to a single component.

1) *System-inherent policies* are statically defined, cannot be overridden, and are subject to system certification.
2) *Globally configured policies* are subject to a scenario-specific configuration. They are enforced mandatorily by trusted components.
3) *Freely programmable policies* are short-lived, not mandatory as well as not subject to any certification.

When different policies conflict, system-inherent policies always overwrite globally configured policies, which are in turn higher prioritized than freely programmable ones.

A. Trusted Components

Apart from correctly forwarding network packets the data plane implements system-inherent policies to separate packets between different security groups to achieve data CIA. This is done by applying two different methods:

First, before handling network traffic in untrustworthy networks, it gets *encrypted mandatorily* resulting in a VPN which spans between all trusted networks. Second, the *data paths* of different security groups get *isolated*. The realization of separation in trustworthy networks depends on the specific security policy, e.g., by using encryption, physical separation, VLANs, etc. Control plane interaction, e.g., reading forwarding tables, as well as interfaces to untrustworthy control plane components are offered per security group.

Since the data plane must be managed securely, a minimized, trusted control plane has to exist. Its tasks include initial configuration and provisioning, management access, updating security policies, emergency shutdown, key management etc. This does not require free programmability, and can be realized by small, rather static components with the possibility of certification. All trustworthy components are colored green in Fig. 1.

B. Untrustworthy Components

All control plane functions that are not directly involved in mandatory security are isolated from the remainder of the system and instantiated per security group (different colors in Fig. 1). They are connected to the data plane in such a way that they can control the forwarding rules, but never violate system-inherent policies such as mandatory encryption or globally configured policies. To provide an easy, versatile and exchangeable programming interface, we utilized an open source Network Operating System (NOS), which is primarily intended to be deployed on a white box (hardware) switch. We attached it by virtualizing it and deploying a custom hardware abstraction driver, which translates control commands to the data plane. Thus, it provides the illusion to the NOS of controlling an off-the-shelf hardware switch. In our prototype, we used the Switch Abstraction Interface (SAI), a standardized C API [16], to provide hardware abstractions. Being freely programmable, it allows to run Linux routing daemons or control delegation to centralized SDN controllers. To achieve CIA between different security groups, no communication between multiple groups is allowed for untrustworthy components. As the NOS is bound to the data plane interface of its security group, it is impossible for the NOS to send commands to another security group to affect its availability.

C. Policy Enforcement

To enforce scenario-dependent invariants, a *Policy Enforcement* component is placed between trusted and untrustworthy components. It proxies all SAI commands exchanged between the data plane and the NOS. Since SAI is a well-defined interface, we can identify commands that could interfere with security requirements, i.e., functions that read, create, or modify network packets violate the goal of CIA between control and data plane. In SAI only a small subset of all functions and attributes ($\sim 1\,\%$) is involved in reading, creating, or modifying packets and must be controlled by policy enforcement. A scenario-dependent policy could allow only the exchange of BGP packets. If the NOS is not depended on interfering with users' network packets, the most restrictive policy – to deny all corresponding commands – may be applied. Note that even in

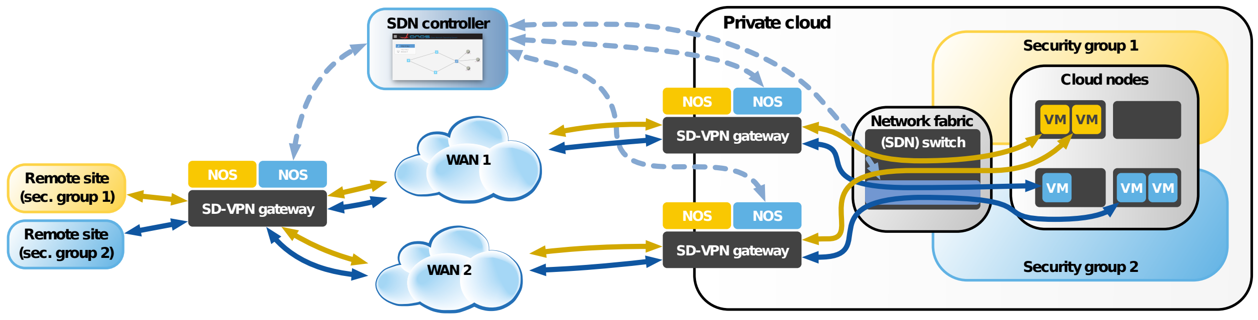

Fig. 2. Demonstration setup with three SD-VPN gateways, connecting a remote site and a cloud over two Wide Area Networks (WANs)

this case, flows may be measured and redirected using other, more abstract SAI functions, which have a lower impact on security.

IV. Demonstration

The demo setup implements the architecture described in Sec. III. It consists of three physical SD-VPN gateways with two security groups configured. They connect *remote sites* to a *private cloud* over the two WANs which may both be used to transport traffic (see Fig. 2). At the remote site, separation between the security groups is done by physical separation, whereas within the WANs and the cloud network traffic is encrypted. The separation of the security groups inside the cloud is based on the principles described in [17]. All methods of separation are enforced by the trustworthy data plane of the SD-VPN gateways. On every gateway two distinct untrustworthy NOSes are deployed. In particular, we run containerized OpenSwitch (OPX) [18] instances. Due to the described architecture they cannot access, modify or create network packets, let packets circumvent the mandatory encryption or communicate with another security group. The NOS of security group 2 is connected to a COTS SDN controller (ONOS [19]) via NETCONF [20], which itself also has no security critical access to data traffic. ONOS additionally controls the *clouds network fabric* via OpenFlow. Since the payload of network packets is encrypted, data CIA can still be guaranteed. The demo consists of three experiments which showcase the functional capabilities of the new architecture. First, we introduce discretionary access control inside the WAN. For security group 1 we write Access-Control Lists (ACLs) into the OPX instances. This leads to a restriction of packet types that can be sent over the SD-VPN gateways for security group 1. At the same time, the NOS is in no way able to see any traffic. Second, we show simple traffic engineering combined with a failover scenario between the two WANs by deploying so called intents inside the SDN controller for security group 2. This allows the prioritization of a WAN causing specific network flows to be routed over one specific WAN. If the connection over this WAN is lost, the SDN controller rewrites the flow rules so that traffic is forwarded over the other WAN. For the third experiment, monitoring is included by adding intents that pick a route based on the paths' traffic metrics. In our case we base the intent on the amount of packet loss. As we start to introduce a certain amount of packet loss in one WAN, the SDN controller reroutes traffic into the other WAN.

References

[1] Synopsys, Inc. OpenHub ONOS Metrics. [Online]. Available: https://www.openhub.net/p/onos
[2] ——. OpenHub OpenDaylight Metrics. [Online]. Available: https://www.openhub.net/p/opendaylight
[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.
[4] "OpenFlow Switch Specification (Version 1.3.0)," 2012.
[5] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," *OpenFlow Switch Consortium, Tech. Rep*, 2009.
[6] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow, "OpenVirteX: Make your virtual SDNs programmable," in *Hot topics in software defined networking*, 2014, pp. 25–30.
[7] D. Hancock and J. Van der Merwe, "HyPer4: Using P4 to Virtualize the Programmable Data Plane," in *CoNEXT*, 2016, pp. 35–49.
[8] A. Al-Alaj, R. Sandhu, and R. Krishnan, "A formal access control model for se-floodlight controller," in *Workshop on Security in Software Defined Networks & Network Function Virtualization*, 2019.
[9] Y. Tseng, M. Pattaranantakul, R. He, Z. Zhang, and F. Naït-Abdesselam, "Controller DAC: Securing SDN controller with dynamic access control," in *ICC*, 2017.
[10] Y. Tseng, Z. Zhang, and F. Naït-Abdesselam, "ControllerSEPA: a security-enhancing SDN controller plug-in for OpenFlow applications," in *PDCAT*, 2016, pp. 268–273.
[11] F. Xiao, J. Zhang, J. Huang, G. Gu, D. Wu, and P. Liu, "Unexpected Data Dependency Creation and Chaining: A New Attack to SDN," in *IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 264–278.
[12] F. Hauser, M. Häberle, M. Schmidt, and M. Menth, "P4-IPsec: Implementation of IPsec Gateways in P4 with SDN Control for Host-to-Site Scenarios," *CoRR*, vol. abs/1907.03593, 2019.
[13] Y. Li and J. Mao, "SDN-based access authentication and automatic configuration for IPSec," in *ICCSNT*, 2015, pp. 996–999.
[14] W. Li, F. Lin, and G. Sun, "SDIG: Toward software-defined IPsec gateway," in *ICNP*, 2016.
[15] G. Lopez-Millan, R. Marin-Lopez, and F. Pereniguez-Garcia, "Towards a standard SDN-based IPsec management framework," *Computer Standards and Interfaces*, vol. 66, 2019.
[16] "Switch Abstraction Interface (SAI) - A Reference Switch Abstraction Interface for OCP," Open Compute Project, Tech. Rep., 2015.
[17] M. Pfeiffer, M. Rossberg, S. Buttgereit, and G. Schaefer, "Strong Tenant Separation in Cloud Computing Platforms," in *ARES*, 2019.
[18] The Linux Foundation. OpenSwitch (OPX) Webpage. [Online]. Available: https://www.openswitch.net/
[19] Open Networking Foundation. ONOS Webpage. [Online]. Available: https://www.opennetworking.org/onos/
[20] R. Enns, M. Bjorklund, J. Schonwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," RFC 6241, 2011.