Covert Communication in Software Defined Wide Area Networks

Undergraduate Thesis

Submitted in partial fulfillment of the requirements of
BITS F421T Thesis

By

Dhruv Rauthan
ID No. 2019A7TS0095G

Under the supervision of:

Prof. Stefan Schmid
&
Prof. Vinayak Naik

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, GOA CAMPUS
December 2022

Declaration of Authorship

I, Dhruv Rauthan, declare that this Undergraduate Thesis titled, ' Covert Communication in Software Defined Wide Area Networks ' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: **13 December 2022**

Certificate

This is to certify that the thesis entitled, " *Covert Communication in Software Defined Wide Area Networks* " and submitted by Dhruv Rauthan ID No. 2019A7TS0095G in partial fulfillment of the requirements of BITS F421T Thesis embodies the work done by him under my supervision.

Supervisor
Prof. Stefan Schmid
Professor,
TU Berlin
Date: **13 December 2022**

Co-Supervisor
Prof. Vinayak Naik
Professor,
BITS-Pilani Goa Campus
Date: **13 December 2022**

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, GOA CAMPUS

Abstract

Bachelor of Engineering

Covert Communication in Software Defined Wide Area Networks

by Dhruv Rauthan

Software defined wide area networks (SD-WANs) have become increasingly popular in recent years due to their ability to provide efficient and secure communication across large geographical areas. However, the use of SD-WANs also presents potential security risks, including the possibility of developing covert channels between network hosts for malicious purposes. We analyse an open-source SD-WAN implementation, and investigate possible avenues for developing a covert timing channel in SD-WANs. We achieve this through surveying existing SD-WAN architectural literature and inspecting network traffic during simulations replicating specific network events. Our findings highlight the need for further research in this area to better understand the security implications of SD-WANs.

Acknowledgements

I would like to express my sincere gratitude to Prof. Stefan Schmid and Dr. Liron Schiff for giving me the opportunity to work with them and most of all for their guidance, feedback and encouragement. They were extremely supportive throughout the entirety of my work. I would also like to thank Prof. Vinayak Naik for his support and feedback. Last, but not least, I would like to thank my family for supporting and encouraging me endlessly.

Contents

List of Figures

List of Tables

Abbreviations

SDN	**S**oftware **D**efined **N**etworking
IP	**I**nternet **P**rotocol
WAN	**W**ide **A**rea **N**etwork
SD-WAN	**S**oftware **D**efined-**W**ide **A**rea **N**etwork
API	**A**pplication **P**rogramming **I**nterface
REST	**RE**presentational **S**tate **T**ransfer
LAN	**L**ocal **A**rea **N**etwork
MPLS	**M**ulti**P**rotocol **L**abel **S**witching
CPE	**C**ustomer **P**remises **E**quipment
VPP	**V**ector **P**acket **P**rocessing
FRR	**F**ree **R**ange **R**outing
OSPF	**O**pen **S**hortest **P**ath **F**irst
RIP	**R**outing **I**nformation **P**rotocol
BGP	**B**order **G**ateway **P**rotocol
JSON	**J**ava**S**cript **O**bject **Notation**
CLI	**C**ommand-**L**ine **I**nterface
URL	**U**niform **R**esource **L**ocator
UI	**U**ser **I**nterface
IPSec	**I**nternet **P**rotocol **Sec**urity
UDP	**U**ser **D**atagram **P**rotocol
GRE	**G**eneric **R**outing **E**ncapsulation
MTU	**M**aximum **T**ransmission **U**nit
ZTP	**Z**ero **T**ouch **P**rovisioning
VM	**V**irtual **M**achine
ARP	**A**ddress **R**esolution **P**rotocol

MAC	**M**edia **A**ccess **C**ontrol
NTP	**N**etwork **T**ime **P**rotocol
ESP	**E**ncapsulating **S**ecurity **P**ayload

Chapter 1

Introduction

With the advent of large scale network infrastructures, SDN was introduced to provide a more flexible and agile way to manage and control network traffic. Traditional networks are often inflexible and difficult to manage, requiring manual configuration of individual devices and making it difficult to quickly adapt to changing network conditions. As it is a relatively new technology (which is already a part of an estimated 30% of the world's IP WAN traffic [31]), there exists massive potential for security loopholes for attackers to exploit. Hence, the discovery and understanding of such security issues, such as covert channels is essential for the protection of sensitive data and the integrity of an organization's network. Covert channels can be used by attackers to secretly exfiltrate data, undermining security and potentially leading to overall network and information damage. Detecting and preventing their use is crucial for maintaining the confidentiality, integrity, and availability of sensitive information.

1.1 Problem Statement

Timing covert channels in SD-WANs present a potential security vulnerability that can be exploited by attackers to exfiltrate sensitive information from the network or to exchange information between hosts. The objective of this thesis is to identify the presence of timing covert channels in SD-WANs and to present effective methods for detecting and mitigating these channels.

1.2 Contributions

This thesis analyses the possibility of covert channels in an open-source SD-WAN implementation. First, we look at the various components which construct the working of an SD-WAN. In an

effort to identify methods to develop a covert channel, we analyse new branch authentication methods employed by SD-WAN solutions (both vendor deployed and in SD-WAN architectural paradigms). The second avenue we explore is tunnel reconfigurations in the network, particularly during network device disconnection. Unfortunately, we were unable to develop a covert channel using either of these methods, however we observe and conclude certain points related to the functioning and operation of an SD-WAN implementation, which include the nature of status updates exchanged between the network devices.

1.3 Thesis Structure

Chapter 2 goes through the networking paradigms such as SDN and SD-WAN which are used in our research. The architecture and implementation of an open-source SD-WAN solution is also discussed here. Chapter 3 explores the branch authentication approach and the reports the results of the experiments involved with the tunnel reconfiguration method. Chapter 4 discusses the findings from the previous chapter, addresses the limitations of our work and proposes a future direction for the covert channel research. Chapter 5 discusses previous research on the topics covered in this thesis. Finally, Chapter 6 wraps up all the thesis with some final remarks.

Chapter 2

Background

2.1 Software Defined Networking

SDN is a type of computer networking architecture that uses a centralized controller to manage network traffic and make decisions about where data should be sent [18]. This is in contrast to traditional networking architectures, which rely on individual network devices to make routing decisions based on local information.

In terms of how SDN works, the key idea is that the controller is responsible for making global decisions about where data should be sent. This allows the network to be more dynamic and flexible, because the controller can quickly respond to changes in network conditions and adjust the routing of data accordingly.

For example, if a particular network link becomes congested, the controller can use its global view of the network to find an alternative path for the data to take. This can help to improve network performance and reduce the risk of congestion.

2.1.1 SDN Architecture

The architecture of an SDN system typically consists of several key components. The first is the controller, which is a central server that is responsible for making global decisions about network traffic. This controller typically communicates with the other network devices using a northbound API, which allows it to send commands and receive information from these devices.

The other key components of an SDN architecture are the network devices themselves, which are typically referred to as ”switches” in this context. These devices are responsible for forwarding data packets according to the instructions they receive from the controller. In an SDN system,

these switches are often programmable, which means that they can be configured to perform a wide range of functions in response to commands from the controller.

SDN separates the data and control planes in network devices, and generally contains a total of 3 logically separate planes:

- **Data plane:** network infrastructure consisting of the hardware/software devices (routers, switches etc.). They only differ from a traditional IP network in that they do not have any control functions.
- **Control plane:** the SDN controller. The controller takes care of the management and control functions of the network.
- **Application plane:** the SDN applications. It uses the above two planes to maintain the logic for the network of applications.

Figure 2.1 gives an overview of these 3 planes.

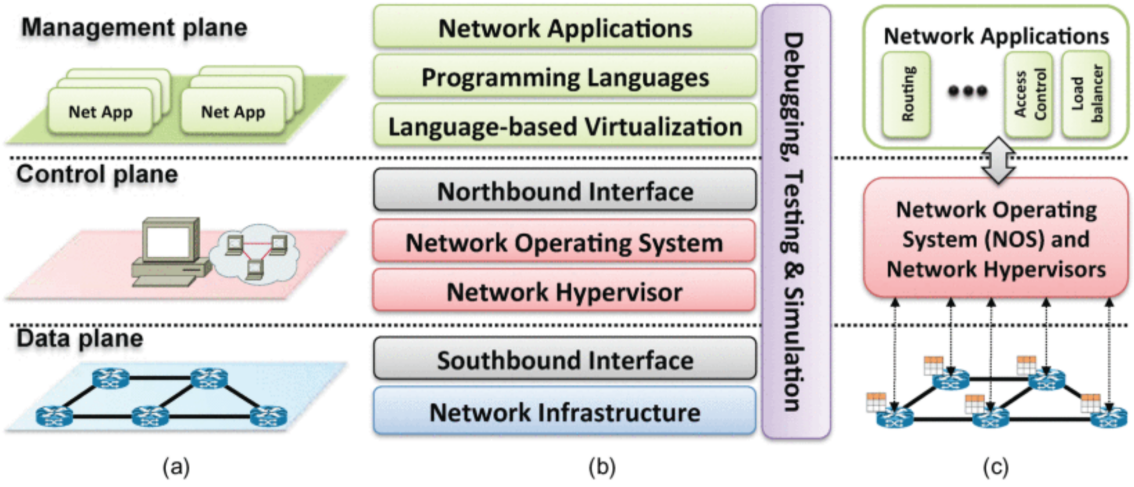

FIGURE 2.1: SDN in (a) planes, (b) layers, and (c) system design architecture. [25]

One of the key benefits of an SDN architecture is that it allows network administrators to manage and control the network more easily and flexibly than they can with a traditional network. This is because the controller provides a single, centralized point of control for the entire network. This makes it easier to implement new network policies, change the way the network is configured, and troubleshoot problems when they arise.

2.1.2 APIs

Northbound and Southbound

Communication between the planes takes place through the Northbound and Southbound APIs. The Northbound API connects the control plane to the management, i.e, application plane. This can be achieved using REST APIs, which allow the SDN application to control and manage the network by sending control instructions to the controller. The Southbound API translates these instructions and sends them to the network devices. This enables the controller to send commands to the network devices as well as receive information from them regarding their status etc. This is generally achieved using OpenFlow protocol APIs. [18]

Eastbound and Westbound

In contrast to the above, the Eastbound and Westbound APIs enable communication horizontally, i.e, between controllers and are an essential component of distributed controllers. These APIs are leveraged to create more scalable and distributed network control platforms. A particular SDN methodology [16] distinguishes between them both, by handing SDN to SDN protocols to the Westbound interfaces and management of communication with legacy networks to the Eastbound interfaces.

2.2 Software Defined-Wide Area Network

A WAN is a type of network that spans a large geographic area, such as a country or continent. WANs typically connect multiple smaller networks, such as LANs, and allow devices on these networks to communicate with each other. A SD-WAN is a type of network architecture that allows for the management and operation of a WAN through the use of SDN principles.

2.2.1 Why SD-WAN?

Although MPLS can be used to guarantee quality of service, it presents some challenges, such as:

- High bandwidth cost, MPLS is more expensive than Internet services which only supports best-effort mechanism
- Configuration overhead, zero-touch deployments are not possible, each device is configured separately
- Time required to transition/upgrade: operation time is directly related with the number of branch offices and edge devices.

The high cost of MPLS is pushing companies to use Internet/broadband services [27].

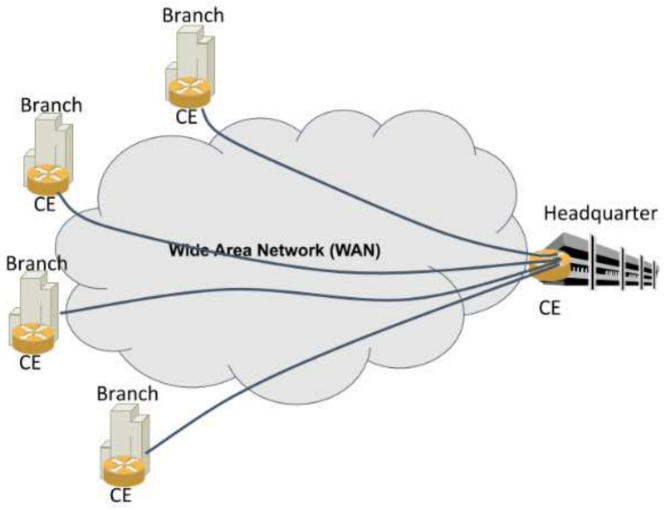

FIGURE 2.2: WAN Reference Scenario [24]

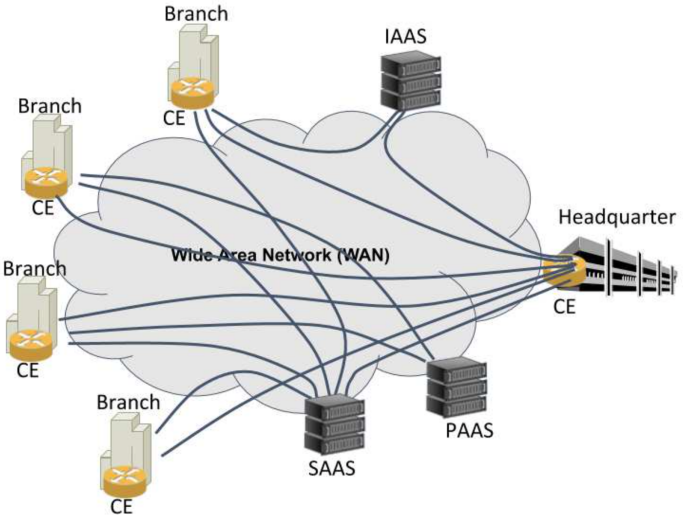

FIGURE 2.3: SD-WAN Reference Scenario [24]

Figure 2.2 shows an example of a traditional WAN scenario, where an organization owning several branches needs to interconnect them with the main office. With the advent of cloud, enterprises have started outsourcing their applications and using Software-as-a-Service (SaaS) and Infrastructure-as-a-service (IaaS) from multiple cloud providers. Figure 2.3 highlights the connectivity needs of the SD-WAN customers. It is clear from the figure above that the hub-and-spoke communication model of traditional WANs (shown in Figure 2.2) was not designed with these concepts in mind and cannot meet the needs of today's digital businesses [24]. The user experience is poor, due to the large number of links increasing network traffic congestion, or cost for maintenance is not sustainable. By adapting WAN solutions in the form of software-defined, enterprises have many advantages, such as better network performance, automation on networking deployment, cost reduction and expedited service delivery [28]. SD-WAN's ability to

optimize network traffic in real-time, flexibility, scalability and enhanced security has led to its adoption by vendors such as Cisco [3], Palo Alto Networks [22] and Juniper Networks [17].

2.2.2 SD-WAN Architecture

SD-WAN architecture is similar to SDN; having 3 planes: data, control and orchestration [25].

- **Data plane:** It is designed to simplify communication between geographically separated sites, as well as with cloud applications and services. SD-WAN creates its own software logical infrastructure, i.e, the overlay network, over the physical infrastructure, i.e, the underlay network. Since a WAN is made up of several separate entities, there might be differences in physical infrastructure used. The overlay can create a uniform and consistent network, by using links or tunnels between sites.
- **Control plane:** Similar to SDN, this manages the control logic for the network. There may be one or several controllers used at once. This layer is responsible for controlling the configuration of the connected devices, while the CPE is connected to the controller by the southbound API. The programming of the control instructions is done by sending instructions from the orchestration plane through the northbound API.
- **Orchestration plane:** This plane is broadly used; for enforcing policies, configuration, monitoring, performance analysis, troubleshooting etc. The entire SD-WAN service is managed at this level of abstraction. Through SD-WAN application interfaces, it uses the northbound API to send instructions to the controller(s) and receive information as well.

2.3 flexiWAN

SD-WAN has been widely adopted by major routing and networking vendors. These companies often offer a monolithic software stack that controls every aspect of the solution, making it difficult for customers to have control over their deployment and future costs. This traditional business model can lock customers into using a specific vendor's solution.

flexiWAN is an alternative solution to this issue, by providing an open-source SD-WAN infrastructure that includes the vRouter, management, orchestration, automation and core networking functionality [2].

2.3.1 flexiWAN Architecture

flexiWAN consists of 2 major entities, namely flexiEdge, a software edge network device, and flexiManage, the central orchestration/management system. [9].

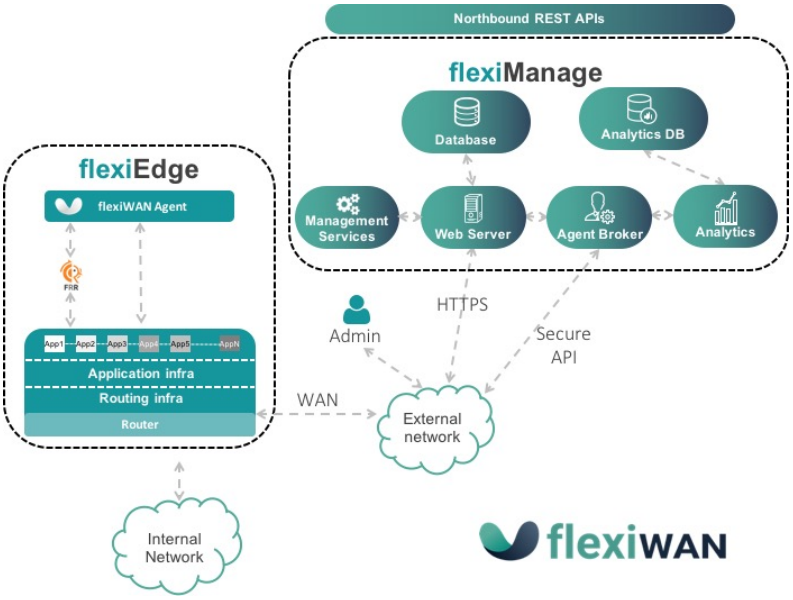

FIGURE 2.4: flexiWAN High Level Block Diagram [9]

Figure 2.4 outlines flexiWAN architecture. Communication between the flexiEdge device and flexiManage takes place through a secure encrypted API. flexiManage, part of the SD-WAN orchestration plane, provides management and configuration of the flexiEdge network devices as well as provides network statistics. Network administrators can use flexiManage as a central point of control for all flexiEdge devices [9].

2.3.1.1 flexiEdge

Every flexiEdge device is composed of 3 entities:

- **Router infrastructure:** This is a modified version of FD.io's VPP [30]. FD.io is a networking technology used to build network functions, an open-source networking data plane [5]. VPP, the underlying technology behind FD.io, helps achieve this by allowing low latency, multiple packet processing. Improving scalability, this is a crucial component of the SD-WAN implementation.
- **Routing control plane:** The control plane is managed by FRR. FRR is an open-source IP routing suite for network devices and provides implementation of protocols such as OSPF, RIP and BGP [12]. SD-WAN's control plane is handled by FRR.

- **flexiWAN Agent:** This component connects the flexiEdge device with the flexiManage orchestrator using secure APIs. This is discussed in detail at 2.3.1.2.

flexiWAN provides a localhost UI for device onboarding, configuration and troubleshooting. flexiEdge UI is enabled by default on all interfaces and it can be accessed using device IP and port 8080 [6]. Every flexiEdge contains a system checker which verifies that the device can run flexiWAN software. If any errors are present, the system checker can automatically change particular software configurations to try and fix them.

flexiEdge can be installed through a pre-built virtual machine, bare metal appliance or the cloud. To add a flexiEdge device to the SD-WAN, flexiWAN uses tokens. Adding the organization's unique token to the flexiEdge links the device to the flexiManage account.

2.3.1.2 flexiWAN Agent

flexiWAN Agent or flexiAgent is responsible for managing the communication between the flexiEdge device and flexiManage. It uses a bi-directional secured web socket connection for network configuration and statistics. It supports the following capabilities [8]:

- Receive simplified JSON commands from flexiManage.
- Separate and translate APIs into internal commands.
- Network configuration storage.
- Manage the execution sequence of various operations.
- Restoring the last system state and configuration after device reboot.
- Monitoring components and restart in case of failure.
- Provide a JSON structure of the entire configuration.
- Provide CLI commands for troubleshooting.

2.3.1.3 flexiManage

The flexiManage service is used for managing the flexiWAN network, configuring and connecting to the flexiEdge devices. Users can create accounts to manage the entire network inventory of the organizations using SD-WAN. A web URL provides a UI for flexiManage which can be accessed. It collects statistics from devices using the analytics system and provides device monitoring and regular status updates. Network administrators can manage the SD-WAN

through flexiManage, which is responsible for communicating with flexiEdge devices. It also behaves as a communication channel between the network devices and the server.

The open-source backend component for flexiManage provides a REST API for SD-WAN management [7].

2.3.2 Routing

Routing through the flexiEdge devices is managed by FRR, an open-source IP routing suite for Unix-based network devices. flexiWAN allows addition of static routes, used for routing traffic through various network interfaces. Static routes can be redistributed to other sites connected via tunnels through either OSPF or BGP.

For OSPF, the users can configure various OSPF parameters, such as Hello interval and Dead interval, according to the network requirements. Figure 2.5 shows an OSPF configuration with separate LANs.

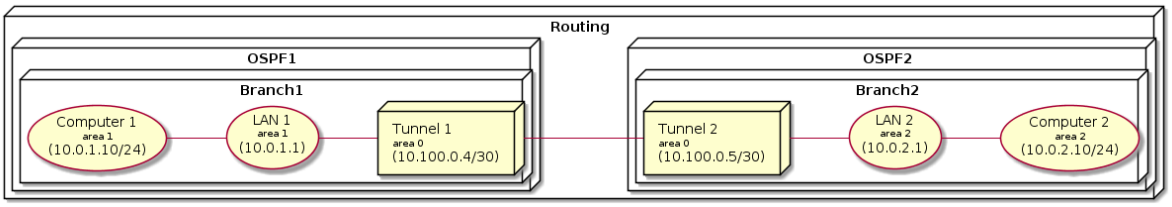

FIGURE 2.5: flexiWAN OSPF configuration [10]

flexiWAN also provides BGP routing, which can be combined with OSPF as well.

2.3.3 Tunnels

SD-WAN uses tunnels to provide overlays on the transport layer underlay and are used to connect multiple CPEs (LANs) into a larger network which is the SD-WAN itself [21]. They work by encapsulating packets which allows for secure movement of data from one site to another.

flexiWAN uses encrypted IPSec over VXLAN tunnels to create its own tunnels [11]. The VXLAN tunnel uses UDP, which wraps around an IPSec tunnel, which in turn wraps around a GRE tunnel which contains the original packet. The tunnel headers, from outermost to innermost, in order are VXLAN + UDP, IPSec, GRE, original packet header. flexiWAN offers various topologies such as hub and spoke, full mesh or a custom topology for tunnel creation. Using OSPF and BGP, the LAN routes of the sites are advertised across tunnels, enabling cross LAN communication. Several key encryption methods are available, for example, Pre-Shared Key, Internet Key Exchange v2 and no encryption. Users can opt for path labels which help

organize and manage the network more efficiently. Figure 2.6 provides a graphical overview of a tunnel network between sites such as Tel Aviv, Berlin, Paris etc, which can be viewed through flexiManage.

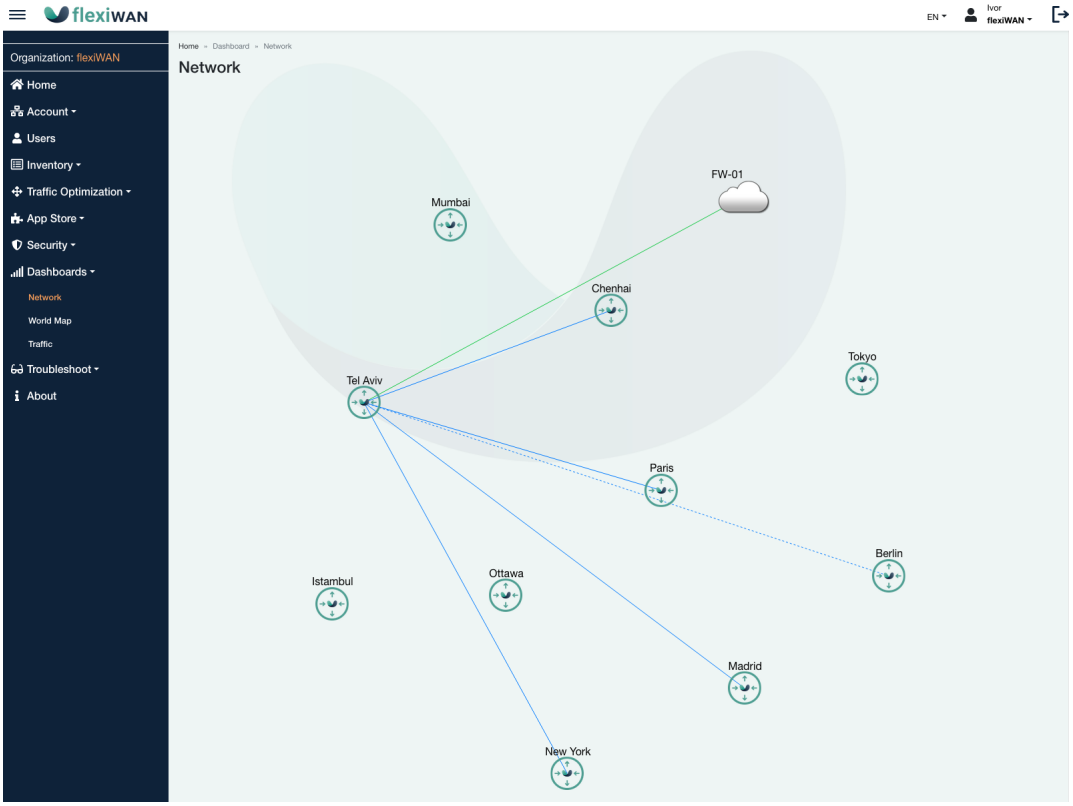

FIGURE 2.6: An example of a flexiWAN tunnel network [11]

However, there exist several issues with using tunnels in SD-WANs [21]. Adding headers during packet encapsulation results in an overhead, which consequently causes inefficient utilization of the available bandwidth. Approximately 40% additional bytes are used for an IP packet, which is generally not needed if sending the packet without encapsulation. Furthermore, multiple encapsulations, as in the case of flexiWAN, leads to even more overhead, reaching up to 123% in the worst case. If the links between the sites do not have sufficient bandwidth to carry the encapsulated packet, the network can become congested and lead to a decline in performance of applications dependent on the SD-WAN. It is crucial to have enough tunnel bandwidth for smooth operation. The second issue plaguing tunnels is fragmentation. Encapsulation often increases the packet size above the regular MTU size. This leads to increased fragmentation, which has several negative effects, including increasing computational work for the routers. Scalability, according to the maximum number of tunnels supported by a router, and security, including evasion of filters by tunneling, are some more issues faced during the implementation of SD-WAN tunnels. Discussion about a tunnel-free SD-WAN is still ongoing [1] although most popular vendors are yet to implement it.

Chapter 3

Exploring Covert Channels in SD-WAN

We explored the implementation of proprietary and open-source SD-WAN solutions to examine the network and architectural paradigms used. The aim was to exploit the timing of control plane or general router communication to identify potential covert channels where hosts in different LANs can exchange information between each other. Section 3.1 discusses the approaches taken SD-WAN solutions to authenticate a new branch into the network. Section 3.2 goes over the tunnel reconfiguration policies, with flexiWAN's policies being discussed in detail in Section 3.3. Finally, Section 3.4 investigates security exploits in the VXLAN tunnels commonly used in SD-WANs.

3.1 Branch Authentication

Branches, or LANs, in the SD-WAN are an important feature for network segregation and organization. When a branch wants to be added to the network, it needs to be authenticated by the manager first. This approach was inspired by the idea that during branch authentication, the manager (or to put it more simply, the SD-WAN controller) might incur delays in handling requests from other devices in the network, especially if a single processor is handling all internal and external requests. For example, if a network device requesting information from the manager sends its request packet at the same time as a new branch is trying to authenticate itself. In this case, the device will observe an unnatural delay in receiving the response back, which is a potential timing channel for exchanging binary information. Importantly, the new branch need not be authenticated in the end, only the process of authentication needs to take place.

Cisco's SD-WAN uses ZTP for authenticating new devices [4]. ZTP is a method of setting up devices, wherein a new device in the network is automatically configured, without the network administrator manually approving each one. Cisco deploys a separate ZTP server in its SD-WAN configuration that validates the device after confirming its serial number and root certificate. Since the authentication server and the controller are not handled by the same processor, authenticating a new device will not cause any delays in the network.

The MEF Standard for SD-WAN assumes that an agreement is already present between the subscriber, the WAN who wants to become SD-WAN, and the service provider, who provides software defined capabilities to the WAN [20]. The existing agreement between the two signifies that the subscriber will provide the correct credentials for the entire network to the service provider, leaving no room for an external branch trying to authenticate itself. Furthermore, no algorithms or changes were mentioned for the provisioning of new devices of CPEs.

As discussed in Section 2.3.1.1, flexiWAN uses tokens for device authentication. The network administrator has to create a token in the flexiManage portal and share it to the flexiEdge device's administrator to manually enter it in the configuration file. Here, addition of a new device in the network becomes impossible without the token, which will only be sent to trusted and verified entities. Other implementations also assumed mutual trust between the network administrator and unauthenticated device, leaving no room for a potential exploit.

3.2 Tunnel Reconfiguration

SD-WAN uses tunnels to provide overlays on the transport layer underlay and are used to connect multiple CPEs (LANs) into a larger network (Section 2.3.3). flexiWAN uses tunnels to realise links between sites in the SD-WAN. Another open-source implementation also makes use of GRE tunnels to connect the virtual CPEs in their network. Now, what if a tunnel or a router goes down with malicious intent? Will the hosts or routers in other connected LANs be made aware of the disconnection? Timing differences between control plane messages sent by the manager to the network devices about tunnel/router disconnection can be exploited to form a timing channel. This will however, not be a very 'covert' channel, since it involves a very noisy disconnection of entire links or devices which might be reported to the network administrator for suspicious activity. We explore this idea in flexiWAN in Section 3.3.

3.3 flexiWAN

We study the messages exchanged between the network devices during the disconnection of a particular router, which leads to one or more tunnels going down. Section 3.3.1 goes over the

network setup and Section 3.3.2 discusses the experiments carried out during router disconnection.

3.3.1 Setup

The setup consists of 3 flexiEdge devices (flexiEdge device and router are used interchangeably here), namely, Liron (L), Google 1 (G1) and Google 2 (G2). Figure 3.1 gives an overview of the topology. Router L is connected by a tunnel to both router G1 (Tunnel 1) and router G2 (Tunnel 2). Additionally, router L is a flexiWAN VirtualBox image instance running on a personal network, and both routers G1 and G2 are Ubuntu VMs, with the flexiWAN software installed, hosted on Google Cloud. All the routers were deployed according to the instructions given by flexiWAN [14]. A Linux end-host device is connected to both router G1 and G2, to test network connectivity. They have the IP addresses of Table 3.1 shows the network configuration, particularly the WAN and LAN subnets for each of the routers. Static routes were added in all the routers to allow communication between the separate LANs. This topology was chosen to observe if any messages about routing changes were sent by the manager to a router not directly connected to the disconnected router. Packet capture is done using *vppctl*, a VPP command.

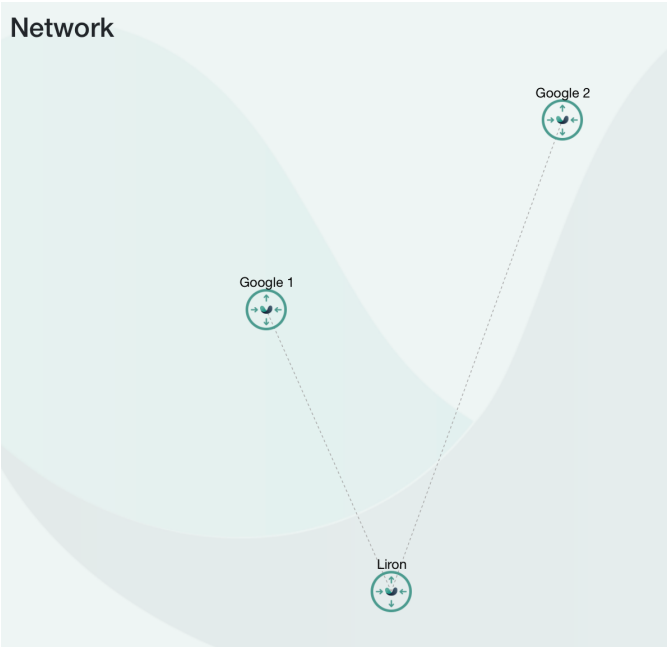

FIGURE 3.1: The network topology

3.3.2 Experiment and Observations

The experiments involve disconnecting router G2 and observing any configuration changes in routers G1 or L due to control plane messages sent by the SD-WAN manager. The router disconnection is achieved using both, shutting down the entire VM and stopping only flexiWAN's

Router	WAN subnet	LAN subnet	Tunnel Interface(s)
L	10.0.0.6/24	192.168.1.1/24	10.100.0.4, 10.100.0.6
G1	10.190.0.2/32	10.4.1.2/32	10.100.0.7
G2	10.160.0.2/32	10.3.1.2/32	10.100.0.5

TABLE 3.1: Network configuration

virtual router running on the VM. Shutting down the VM directly leads to Tunnel 2's status showing *Not Connected* and this takes a few minutes to update in the flexiManage portal.

Analysing traffic capture on router G1, there is no direct communication between it and router G2, which is expected since they are not connected by a tunnel. All messages go through router L (both Tunnel 1 and Tunnel 2) to reach either side.

No.	Time	Source IP	Destination IP	Protocol	Host	Info
1	1970-01-01 00:09:29.748746	10.190.0.2	104.21.67.29	TLSv1.2		Application Data
36	1970-01-01 00:09:29.768859	104.21.67.29	10.190.0.2	TCP		443 → 34722 [ACK] Seq=1 Ack=217 Win=8 Len=0
1081	1970-01-01 00:09:36.960056	104.21.67.29	10.190.0.2	TLSv1.2		Application Data
1090	1970-01-01 00:09:36.963405	10.190.0.2	104.21.67.29	TCP		34722 → 443 [ACK] Seq=217 Ack=116 Win=501 Len=0
1097	1970-01-01 00:09:37.066720	10.190.0.2	104.21.67.29	TLSv1.2		Application Data
1104	1970-01-01 00:09:37.210308	104.21.67.29	10.190.0.2	TCP		443 → 34722 [ACK] Seq=116 Ack=654 Win=8 Len=0
2295	1970-01-01 00:09:46.960178	104.21.67.29	10.190.0.2	TLSv1.2		Application Data
2304	1970-01-01 00:09:46.960978	10.190.0.2	104.21.67.29	TCP		34722 → 443 [ACK] Seq=654 Ack=231 Win=501 Len=0
2334	1970-01-01 00:09:47.220394	10.190.0.2	104.21.67.29	TLSv1.2		Application Data
2341	1970-01-01 00:09:47.240455	104.21.67.29	10.190.0.2	TCP		443 → 34722 [ACK] Seq=231 Ack=1091 Win=8 Len=0
2464	1970-01-01 00:09:48.330195	104.21.67.29	10.190.0.2	TLSv1.2		Application Data
2473	1970-01-01 00:09:48.330838	10.190.0.2	104.21.67.29	TCP		34722 → 443 [ACK] Seq=1091 Ack=255 Win=501 Len=0
2480	1970-01-01 00:09:48.332204	10.190.0.2	104.21.67.29	TLSv1.2		Application Data
2487	1970-01-01 00:09:48.460230	104.21.67.29	10.190.0.2	TCP		443 → 34722 [ACK] Seq=255 Ack=1119 Win=8 Len=0
3708	1970-01-01 00:09:56.960048	104.21.67.29	10.190.0.2	TLSv1.2		Application Data
3717	1970-01-01 00:09:56.963393	10.190.0.2	104.21.67.29	TCP		34722 → 443 [ACK] Seq=1119 Ack=370 Win=501 Len=0
3724	1970-01-01 00:09:57.074004	10.190.0.2	104.21.67.29	TLSv1.2		Application Data
3731	1970-01-01 00:09:57.210122	104.21.67.29	10.190.0.2	TCP		443 → 34722 [ACK] Seq=370 Ack=2010 Win=8 Len=0
5567	1970-01-01 00:10:06.960206	104.21.67.29	10.190.0.2	TLSv1.2		Application Data
5610	1970-01-01 00:10:06.962796	10.190.0.2	104.21.67.29	TCP		34722 → 443 [ACK] Seq=2010 Ack=485 Win=501 Len=0
5633	1970-01-01 00:10:07.064120	10.190.0.2	104.21.67.29	TLSv1.2		Application Data
5640	1970-01-01 00:10:07.084226	104.21.67.29	10.190.0.2	TCP		443 → 34722 [ACK] Seq=485 Ack=2446 Win=8 Len=0
5786	1970-01-01 00:10:08.460056	104.21.67.29	10.190.0.2	TLSv1.2		Application Data
5795	1970-01-01 00:10:08.460683	10.190.0.2	104.21.67.29	TCP		34722 → 443 [ACK] Seq=2446 Ack=509 Win=501 Len=0
5802	1970-01-01 00:10:08.461082	10.190.0.2	104.21.67.29	TLSv1.2		Application Data
5809	1970-01-01 00:10:08.481066	104.21.67.29	10.190.0.2	TCP		443 → 34722 [ACK] Seq=509 Ack=2474 Win=8 Len=0
6872	1970-01-01 00:10:16.960058	104.21.67.29	10.190.0.2	TLSv1.2		Application Data
6905	1970-01-01 00:10:17.003865	10.190.0.2	104.21.67.29	TCP		34722 → 443 [ACK] Seq=2474 Ack=624 Win=501 Len=0
6912	1970-01-01 00:10:17.068939	10.190.0.2	104.21.67.29	TLSv1.2		Application Data
6919	1970-01-01 00:10:17.210266	104.21.67.29	10.190.0.2	TCP		443 → 34722 [ACK] Seq=624 Ack=2910 Win=8 Len=0
8586	1970-01-01 00:10:26.960183	104.21.67.29	10.190.0.2	TLSv1.2		Application Data
8595	1970-01-01 00:10:26.963325	10.190.0.2	104.21.67.29	TCP		34722 → 443 [ACK] Seq=2910 Ack=739 Win=501 Len=0
8645	1970-01-01 00:10:27.084123	10.190.0.2	104.21.67.29	TLSv1.2		Application Data

FIGURE 3.2: Periodic TLS/TCP packets exchanged with flexiManage

The packets observed during regular communication were:

- From Figure 3.2, we can observe that router G1 exchanges encrypted TCP segments with flexiManage almost every 10 seconds. These are periodic status updates messages to check the router functionality and to send information regarding important network events.
- As OSPF was the chosen routing protocol for communication between the LANs, OSPF Hello Packets were exchanged periodically to maintain connectivity with the router's neighbours (in this case, router G1's neighbour will be router L).

- ARP packets were exchanged to discover and match the tunnel endpoint IP with the MAC addresses.
- NTP packets were used to synchronise time between the network devices.
- ESP is a part of the IPSec set of protocols used for encryption of data. These are the actual tunnel packets exchanged between routers G1 and L. The level of encapsulation goes in order from Ethernet II, IP, UDP, VXLAN, Ethernet I, IP, ESP. This matches the order discussed in Section 2.3.3. All communication between the routers occurs through packets encapsulated using ESP.

After disconnecting router G2, router L keeps sending ESP packets to the other Tunnel 2 endpoint (*10.100.0.5*). It receives no reply from router G2, but uses keepalives to check for router G2's status in case it comes online again. Once router G2 reconnects, it identifies it has a tunnel configured and sends an initial ESP packet to router L, which resumes normal communication. Importantly, there is no special communication from flexiManage to either router L or G1 regarding G2's status. Even when static routes are configured for G2's LAN, the other routers do not know if that LAN is accessible anymore.

However, from Figure 3.3, the logs where we can verify the periodic status updates to flexiManage, it can be observed that the flexiAgent present on router L identifies the status of Tunnel 2 as *Not Connected* and according changes the router configuration, by removing the static routes and rebuilding the Linux interfaces cache. Further, Tunnel 2's status changes to *Down* in further status updates to flexiManage. Interestingly, router G1 receives no message from its own flexiAgent to remove the static routes to router G2's LAN subnet.

```
Nov 14 13:25:41 flexiwan-router fwagent: get_reconfig_hash: 1f71a20959313be3fc2f8686d58b1e7f:
deviceType:dpdk,addr:10.0.0.6,gateway:10.0.0.138,metric:100,mtu:1500,public_ip:79.183.34.252,public_port:4789,link:up,deviceType:dpdk,addr:192.168.
1.1,gateway:,metric:,mtu:1500,link:up,
Nov 14 13:25:41 flexiwan-router fwagent: get_linux_interfaces: Finished to build Linux interfaces cache
Nov 14 13:25:40 flexiwan-router fwagent: get_linux_interfaces: Start to build Linux interfaces cache
Nov 14 13:25:40 flexiwan-router fwagent: FwAgent: 8557622:OUYSkIkO handle_received_request:request#012{#012 "entity": "agent",#012 "message": "get-
device-stats"#012}
Nov 14 13:25:35 flexiwan-router fwagent: remove static route through the broken tunnel: {'addr': '10.3.1.0/24', 'via': '10.100.0.5',
'redistributeViaOSPF': True, 'redistributeViaBGP': False, 'onLink': False}
Nov 14 13:25:35 flexiwan-router fwagent: Disable IPv6 default command successfully executed: sysctl -w net.ipv6.conf.default.disable_ipv6=1 > /dev/
null
Nov 14 13:25:35 flexiwan-router fwagent: Disable IPv6 all command successfully executed: sysctl -w net.ipv6.conf.all.disable_ipv6=1 > /dev/null
Nov 14 13:25:34 flexiwan-router fwagent: add_remove_route: netplan apply
Nov 14 13:25:34 flexiwan-router fwagent: sudo ip route del 10.3.1.0/24 metric 0 proto static
Nov 14 13:25:33 flexiwan-router fwagent: FwStunWrap: Re-try to discover remote edge for tunnel: 1 on dev pci:0000:00:03.00
Nov 14 13:25:31 flexiwan-router fwagent: FwAgent: 8557483:Dy6SnGCs handle_received_request:reply#012{#012 "entity": "agentReply",#012 "message":
```

FIGURE 3.3: Router L's logs after router G2's disconnection

3.4 VXLAN

As flexiWAN uses VXLAN tunnels to create links between sites (Section 2.3.3), if a channel can be developed in general VXLAN tunnels, that concept can be extended to SD-WAN who use these tunnels. Due to time constraints, only a normal SDN network with VXLAN tunnels to connect CPEs could be set up on Mininet. No experiments were carried out in this configuration.

Chapter 4

Discussion

4.1 Exploring Potential Covert Channels

We looked at open-source and proprietary SD-WAN architectures and implementations in an effort to identify potential timing covert channels that present a security vulnerability. The problem was approached from multiple angles, the two major approaches being branch authentication and tunnel reconfiguration.

During new branch authentication for an SD-WAN, none of the SD-WAN implementations observed had a singular processor handling internal network requests and external branch authentication requests simultaneously. The authentication of new branches in most cases was through mutual trust and verification by the SD-WAN administrator, through the addition of tokens or a pre-existing agreement between the entities. There was no possibility of developing a covert channel in this network scenario.

In Section 3.3.2, it was observed that no extra communication was made to other devices in the network to indicate and update them regarding particular LANs that might be unavailable due to disconnection. Only through a regular ping, can other devices deduce the status of the disconnected router. Furthermore, only a router which is directly connected to the disconnected router, has information regarding its status, and only because of the tunnel status communicated to it by the flexiAgent.

Other SDN covert channel implementations make use of MAC spoofing [23]. This approach is simply not possible here since routes in the SDN flow tables are installed based on the IP subnet of the LAN, instead of MAC addresses. Additionally, the IP subnets cannot be spoofed as flexiWAN does not allow its virtual router to start if the LAN subnet matches with any other subnet in the network. It is important to note that flexiWAN is just an orchestrator, i.e, it just

enforces network policies, configurations etc. It does not behave as an SDN controller, and does not manage the control logic for the network.

4.2 Limitations and Future Work

Although we were not able to develop a timing covert channel in an SD-WAN, we analysed different SD-WAN solutions to try and develop a covert channel. The assumption of the SD-WAN manager sending status update messages to all network devices was flawed as its function was very different as compared to a normal SDN controller, from which we had initially drawn parallels to. The experiments with flexiWAN confirmed this. Even if messages were sent regarding router disconnection, it would not have been a very 'covert' channel as a major network device is down, which will attract the attention of the network administrator. Furthermore, technical bugs with flexiWAN's software at the time led to obstacles in setting up the Google Cloud VMs due to subnetting issues. The future direction of this work can take the form of analysis of vulnerabilities in VXLAN tunnels. As these tunnels are commonly used in SD-WAN software, there will be fewer assumptions in the problem statement and a potentially successful result can be applied to not only SD-WAN tunnels but generic routing tunnels as well.

Chapter 5

Related Work

5.1 SD-WAN Security

Varuna and Vadivel discuss SD-WAN security trends, including DoS attacks, flow of conflict rules, improving middleboxes, security policies and data encryption [29]. However, they do not mention the possibility of existing side channels in SD-WAN allowing communication between hosts without the knowledge of the controller. Another SD-WAN security checklist only mentions secure connectivity, deployment and segmentation [32]. Previous research does not cover covert communication in SD-WAN. Additionally, the implementations mentioned in Section 5.3 only go through basic encryption methods, and do not provide countermeasures against other forms of security exploits.

5.2 Covert Channels in SDNs

A solution proposing a timing covert channel in SDNs uses malicious switches which can fabricate messages triggering the SDN mobility capability, and resulting in the reconfiguration events at certain switches [26]. Another covert channel makes use of malicious switches exploiting the OpenFlow handshake for establishing covert channels [19]. Le et al. designed an SDN covert channel enabling both host-host and switch-switch communication. However, their implementation assumes that the attacker can install certain applications on the SDN controller itself. Finally, Macchiato improves on these approaches and identifies a covert channel for SDNs between any 2 isolated network devices, removing the need for a malicious switch [23]. This is achieved via MAC spoofing, wherein a MAC learning application running on the SDN controller performs flow reconfigurations post discovery of a new MAC address in the network. These reconfigurations introduce measurable delays during forwarding of data and can be used to

transmit confidential information. We analyse similar approaches such as branch spoofing to introduce delays in SD-WAN network requests.

5.3 Open-Source SD-WAN Implementation

As SD-WAN is a relatively newer technology, there were only a few open-source implementations available. A broad architecture proposed by MEF outlines the policies and regulations to be followed by both, the customer and the SD-WAN service provider [20]. Troia et al. proposed an open-source SD-WAN implementation using GRE tunnels to realise links between sites. flexiWAN, an industry level implementation, provided us with a good understanding of how an SD-WAN manager functions and manages network devices. Both of these implementations used VXLAN and/or GRE tunnels for connecting CPEs. ICONA is an interesting peer-to-peer approach for SD-WANs, which distributes the load of the control plane between the network entities [13]. It implements the SD-WAN by simply connecting different SDN controllers with different LANs together, there is no central manager or orchestrator like flexiWAN does. This is evident as they do not use tunnels to connect the various CPEs. A multi-controller SD-WAN algorithm does incorporate request delays in the network, which can be investigated further to establish timing channels [15], although again they do not mention which type of tunnels they use to connect sites.

Chapter 6

Conclusion

SD-WAN will continue to be an important and widely-used technology in the coming years. As more businesses adopt cloud-based services, the demand for SD-WAN will continue to grow. Hence, it is crucial to understand security exploits in such networks. In an effort to identify and establish covert channels in SD-WANs, we looked at different SD-WAN architectures and implementations, both proprietary and open-source. We understood the policies and regulations used for network events. Our first approach, identifying timing differences in network request during new branch authentication, did not show any potential for a covert channel due to the use of a separate server and pre-existing agreements between the branch and the manager. The second approach involved tunnel reconfigurations, where if a tunnel goes down, the manager would send network status updates to all other devices in the network. However, we observed that the manager behaved unlike an SDN controller and wasn't responsible for any such flow reconfigurations. The future of this work involves investigating VXLAN tunnels, since they are commonly used in SD-WAN implementations, and identifying any potential covert channels there.

Bibliography

[1] *3 Reasons Why the Next Evolution of SD-WAN Will Be Tunnel-Free*. URL: https://www.spiceworks.com/tech/networking/guest-article/3-reasons-why-the-next-evolution-of-sd-wan-will-be-tunnel-free/ (visited on 08/12/2022).

[2] *About flexiWAN*. URL: https://docs.flexiwan.com/overview/about.html (visited on 08/12/2022).

[3] *Cisco SD-WAN*. URL: https://www.cisco.com/c/en_in/solutions/enterprise-networks/sd-wan/index.html (visited on 08/12/2022).

[4] *Cisco SD-WAN: WAN Edge Onboarding*. URL: https://www.cisco.com/c/dam/en/us/td/docs/solutions/CVD/SDWAN/sdwan-wan-edge-onboarding-deploy-guide-2020nov.pdf (visited on 09/12/2022).

[5] *FD.io*. URL: https://fd.io (visited on 08/12/2022).

[6] *flexiEdge UI*. URL: https://docs.flexiwan.com/flexiEdgeUI/overview.html (visited on 08/12/2022).

[7] *flexiManage*. URL: https://gitlab.com/flexiwangroup/fleximanage (visited on 08/12/2022).

[8] *flexiWAN Agent*. URL: https://gitlab.com/flexiwangroup/flexiagent (visited on 08/12/2022).

[9] *flexiWAN Architecture*. URL: https://docs.flexiwan.com/overview/architecture.html (visited on 08/12/2022).

[10] *flexiWAN Routing*. URL: https://docs.flexiwan.com/management/routing.html (visited on 08/12/2022).

[11] *flexiWAN Tunnels*. URL: https://docs.flexiwan.com/management/tunnels.html (visited on 08/12/2022).

[12] *FRR*. URL: https://frrouting.org (visited on 08/12/2022).

[13] Matteo Gerola et al. "Icona: A peer-to-peer approach for software defined wide area networks using ONOS". In: *2016 Fifth European Workshop on Software-Defined Networks (EWSDN)*. IEEE. 2016, pp. 37–42.

[14] *Google Cloud Platform*. URL: `https://docs.flexiwan.com/guides/gcp.html` (visited on 09/12/2022).

[15] Xiaolan Hou et al. "Multi-controller deployment algorithm in hierarchical architecture for SDWAN". In: *IEEE Access* 7 (2019), pp. 65839–65851.

[16] Hoßfeld Tran-Gia Jarschel Zinner and Kellerer. "Interfaces, attributes, and use cases: A compass for SDN". In: *IEEE Communications Magazine*. 2014, pp. 210–217.

[17] *Juniper SD-WAN*. URL: `https://www.juniper.net/us/en/solutions/sd-wan.html` (visited on 08/12/2022).

[18] Verissimo Rothenberg-Azodolmolky Kreutz Ramos and Uhlig. "Software-Defined Networking: A Comprehensive Survey". In: *Proceedings of the IEEE*. 2014, pp. 14–76.

[19] Robert Krosche et al. "I DPID it my way! A covert timing channel in software-defined networks". In: *2018 IFIP Networking Conference (IFIP Networking) and Workshops*. 2018, pp. 217–225.

[20] *MEF Standard SD-WAN Service Attributes and Services*. URL: `https://www.mef.net/wp-content/uploads/2019/07/MEF-70.pdf?id=122&fileid=file1` (visited on 09/12/2022).

[21] Mota. "Tunnel-Based versus Tunnel-Free SD-WANs". In: (2020).

[22] *Prisma SD-WAN*. URL: `https://www.paloaltonetworks.com/sase/sd-wan` (visited on 08/12/2022).

[23] Amir Sabzi et al. "Macchiato: Importing Cache Side Channels to SDNs". In: *Proceedings of the Symposium on Architectures for Networking and Communications Systems*. 2022, pp. 8–14.

[24] Carmine Scarpitta et al. "EveryWAN - An Open Source SD-WAN solution". In: *2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*. 2021, pp. 1–7.

[25] P. Segeč et al. "SD-WAN - architecture, functions and benefits". In: *2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. 2020, pp. 593–599.

[26] Kashyap Thimmaraju, Liron Schiff, and Stefan Schmid. "Outsmarting network security with SDN teleportation". In: *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2017, pp. 563–578.

[27] Maralit Troia Zorello and Maier. "SD-WAN: an Open-Source Implementation for Enterprise Networking Services". In: *2020 22nd International Conference on Transparent Optical Networks (ICTON)*. 2020, pp. 1–4.

[28] Woo Uppal and Pitt. "Software-Defined WAN for Dummies". In: (2015).

[29] W. Rose Varuna and R. Vadivel. *Recent Trends in Potential Security Solutions for SD-WAN: A Systematic Review*. 2021.

[30] *VPP Technology*. URL: https://fd.io/gettingstarted/technology/ (visited on 08/12/2022).

[31] *What is SDN and where software-defined networking is going*. URL: https://www.networkworld.com/article/3209131/what-sdn-is-and-where-its-going.html?page=2 (visited on 07/12/2022).

[32] Michael Wood. "Top requirements on the SD-WAN security checklist". In: (2017).