Improving the security of exposed safety critical systems using SDN

DANIEL BJARNESTIG & FILIPPA DE LAVAL

MASTER´S THESIS

DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Improving the security of exposed safety critical systems using SDN

Daniel Bjarnestig & Filippa De Laval

Department of Electrical and Information Technology
Lund University

Supervisor: Paul Stankovski Wagner and Sven Nilsson

Examiner: Thomas Johansson

September 12, 2019

© 2019
Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

The purpose of this thesis is to study if software defined networks (SDN) can function as a second layer of defence in safety critical sensor networks. SDNs are controlled and topologically defined by a logically centralised control unit. The centralised control logic makes it possible to control the behaviour of the network, and react to network events. In this thesis we examine if and how SDNs can be used to isolate a compromised host from the network. As the intended use case requires galvanic isolation of the devices we test and find that the solution is compatible with the use of media converters and optic fibre. We evaluate the security of SDNs compared to traditional networks and implement a proof of concept using the OpenFlow protocol and Open vSwitch. We find that SDNs could be used to isolate compromised hosts and provide security benefits, but the uncovered method is too immature to be used in a safety critical network.

Popular Science Summary

Software defined networks (SDN) have a lot in common with traditional networks. The main difference is how the packet forwarding devices, for example switches, work. In a traditional network the switch is responsible for both forwarding the data and figuring out where to forward it. In a software defined network a distinction is made between the two jobs. The switch is only responsible for forwarding the data and a new device, a controller, is introduced. The controller contains the logic deciding if and where to forward the data, and the switch has to ask the controller what it should do.

Separating the control logic from the switch makes the network easier to program. This means that applications can be used to monitor the network and alert the controller if something seems odd. The controller can in turn react by redistributing traffic or isolating parts of the network. In a large network this means that traffic in different parts of the network can be coordinated. Another benefit of the separation is that the network traffic governing how the network works can be kept more protected.

The purpose of this thesis is to look at if SDN can be used to improve security in a safety critical network by acting as a second layer of defence if the network would be breached. By setting up a small network we show that it is possible to isolate units which do not follow the rules of the network. We use two different methods to isolate the units, forbidding the traffic from the start and letting an intrusion detection system alert the controller. However, we also find that it is likely possible to bypass the isolation mechanism and that it is not sufficient for a safety critical network.

Another requirement of the network is that the devices should be *galvanically isolated*. Galvanic isolation means that no current flows between the devices. This is a requirement from The Swedish Armed Forces since the current generates an electromagnetic field which could be used to listen to the communication from a large distance. By inserting media converters into the network and using optic fibre to transfer the signal we show that the media converters do not prevent the network from functioning as intended. Their impact on the performance of the network could depend on the hardware used. In the implemented configuration the potential impact is smaller than the variation which occurs between attempts.

We evaluate the security of the implemented network using a threat modelling tool called STRIDE which facilitates finding vulnerabilities in systems by looking at different categories of threats. We find that the biggest issue with the implementation is that traffic could be forged tricking the controller into isolating healthy parts of the network. If an attacker succeeded in this the SDN solution would in practice make it easier for an attacker to shut down the entire network. If it would be a good idea to implement an SDN solution depends on if these issues could be solved, the complexity of the network, and the quality of the intrusion detection system.

Acknowledgements

First and foremost, a thank you to Jonas Lindgren and Sven Nilsson for creating an opportunity for us to write our master thesis at SAAB. It has been a valuable learning experience and a lot of fun. Thank you also to Rickard Nilsson for suggesting that we should write our thesis with SAAB and putting us in contact with Jonas and Sven.

Our warmest gratitude also goes to the Innovation Lab team at SAAB in Gothenburg, Sven Nilsson, Karl Larsson, Emil Hukic, Emil Rosenberg, Jacob Lundberg, Victor Christoffersson and Nils Kåvemark for being great colleagues these last few months. You always made us feel welcome, guided us and were willing to help as soon as we had a question. A special thank you to Sven who has given us a lot of constructive feedback, inspired us, kept us on track, and helped us find the resources to finish the thesis. An extra thank you also to Kalle, who shared his knowledge of networking and Linux operating systems with us. We learned a lot from you all, both things related to the thesis and completely unrelated, and you inspired our love for science and technology.

We would also like to thank Paul Wagner Stankovski at LTH for being our supervisor and trusting us to be able to complete the thesis. We are very thankful for all the advise and for the feedback you took time out of your vacation to give us.

Terminology

- Administrative station - A computer on a network from where the network can be managed and configured.
- API - Application Programming Interface.
- Forwarding device - A network device with the purpose to forward traffic from point A to point B.
- IDS/IPS - Intrusion Detection System/Intrusion Prevention System
- Interface - A connection enabling exchange of some type of information between between two different software or hardware components. In the context network interface it refers to the physical or virtual port on which data is received or transmitted. In the context northbound/southbound interface it refers to the common set of rules for how information is to be exchanged.
- Middlebox - A network device that inspects, modifies or filter traffic, e.g. a firewall.
- Network bridge - A network entity which connects two or more separate networks into one mutual
- RHEL - Red Hat Enterprise Linux, an open source operative system
- Safety-critical system - A system where a failure could result in loss or sever injury to people, property or equipment.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

This chapter will introduce the motivation for the thesis. The first subsection contains the purpose of the thesis. We then move on to present research questions and limitations. Methodology is covered in the fourth section before related work in the fifth. The disposition of the thesis is covered in the final section.

1.1 Purpose

To have an efficient military defence it is necessary to know where the enemy is and to have information about the surrounding area. One method of obtaining this knowledge is by using sensor networks. A sensor network can, for instance, be a number of interconnected radar sensors which exchange information with each other. If the enemy gains access to such a network the damage could be enormous.

To protect the sensor networks they typically communicate on purpose built networks with a relatively low capacity. The cost of building high capacity networks makes it unpractical so if it was possible to use existing infrastructure in a secure way the bandwidth could be greatly improved and the functionality of the networks would increase. Furthermore, if the networks could be connected to the Internet, external information could be gathered for additional improvements. However, when connecting military networks directly to the internet, there is a very high probability that actors with almost never ending resources will try to gain access to those systems.

Software defined networking is a relative new way of setting up and maintaining networks. In a traditional network, a topology usually consists of computers, switches, and routers. The network will remain the same until the system administrator manually changes the topology. It can be a new host or a new setup for how the switches are connected. With software defined networks, or SDN for short, it is possible to logically, and dynamically, control and change the network topology. A network controller unit has a full view of the network topology and can manage all network communication. This technology opens up for a new way of interacting with and configuring networks.

According to [1], SDN brings a lot of positive new possibilities for network management, but also a new set of threats. To introduce the SDN concept in safety critical systems, a thorough examination of how well it is possible to protect the network against these new threats must be done. The purpose of this thesis is to evaluate if software define networking could be used to mitigate the damage of an intrusion that has already occurred.

1.2 Research Questions

In order to evaluate if software defined networking can be used to mitigate the damage from an intrusion in a safety critical sensor network, three research questions will be answered.

- How can SDN be used to shape the network dynamically in a way that improves overall security without introducing new significant threats?
- Given that a node is compromised, can the intrusion be contained by redefining the network around that node?
- Is it possible to build an SDN where the nodes are galvanically separated?

Galvanic isolation is necessary to fulfil the requirements of the Swedish Defence Material Administration. The concept will be further explained in the theoretical background.

1.3 Limitations

As the first research question is broadly stated several limitations have been made to the project.

First of all, no security measures to avoid an outside attacker from entering the network will be taken. In other words, solutions such as cryptographic methods to protect communication with an outside network and authentication of outside hosts will not be covered. These solutions are considered to be independent of whether the implemented network is an SDN or not.

Second, attack detection is not part of the scope. Only reacting to known events will be considered. Although an intrusion detection system (IDS) is included in the built network, it is solely for the purpose of showing how such a system can be incorporated into the architecture. The configuration and dependability of the IDS in particular is not of interest as a different IDS most probably would be implemented in production.

Third, when implementing the test network, it will focus on the method for isolating nodes. Other aspects, e.g. redundant controllers and switches, which most probably would be necessary in a production grade network, will be left out. This is because of time limitations and that the concepts are trivial and would not add any real value to the thesis result. Instead, a discussion of outcomes if redundant switches and routers are added is held in the discussion chapter.

Finally, only the configuration of the physical machines is of interest. In production most processing units would likely run a number of virtual machines or containers. To limit the scope of the project, and as there is no need to design the virtual environment, these are not included. However, the solution must enable connecting the virtual environments to the network.

1.4 Methodology

Before any practical implementation of a network, a thorough literature review will be conducted. The scope will be narrowed down and a small proof of concept network will be implemented. As the first research question, *How can SDN be used to shape the network dynamically in a way that improves overall security without introducing new significant threats?*, has a very broad scope a large part of the analysis will be done in the discussion. The other two research questions will be practically tested, although the isolation of nodes requires a little more theoretical discussion. Conclusions will be drawn based on the obtained results and the theoretical analysis.

1.5 Related Work

Although different versions of SDN or related technologies has existed for decades, the amount of research was limited before the invention of OpenFlow. After Openflow was created and released, the interest for SDN rose and the amount of research increased.

In [2], it shows a feasible attack on how to perform a Denial of Service attack on the control plane of an SDN and also a resource consumption attack on the data plane. This demonstrates the increased attack surface of an SDN compared to a traditional network, and the importance of protecting the network from these new attacks.

In [3], more attacks that are possible to execute on some of the more common controllers are shown. It is also discussed how attacks that can be performed and stopped in traditional networks, may not be stopped the same way in an SDN. This is because the protection mechanisms against some of the attacks rely on a switch's intelligence. This intelligence does not exist in a software defined network, where a switch simply is a forwarding device and has no intelligence.

Attempts have been made to create dependable architectures for software defined networks. Examples of such architecture include OrchSec and AutoSec. In [4] [5], OrchSec is presented and is shown as an orchestration based solution involving a monitor where the monitor informs the orchestrator of irregularities, which in turn updates the controller. In [6], the functioning of AutoSec is presented and shows that it should provide encryption and be able to isolate and contain intrusions. However, information about the two projects is sparse and there seems to be no publicly available implementation of either.

When looking at other possibilities than local networks to use in organisations with high security standard, [7] examines and analyses if cloud infrastructure could be trusted enough to be utilised in a military environment

1.6 Disposition

In the first chapter, a short introduction and motivation for the thesis has been provided. Chapter two will lay a theoretical foundation and cover topics such as how SDN works, security aspects and how said aspects can be evaluated. It will also introduce the applications used for the proof of concept network. The third chapter lays out the architecture for the implementation and the fourth chapter describes the steps taken to set up the network. In the fifth chapter the method of testing and the results are disclosed. The sixth chapter is the discussion and includes the theoretical analysis of how secure an SDN can be made. In the seventh, final chapter conclusions are drawn and the research questions answered.

Chapter 2

Theoretical Background

This chapter will introduce some techniques and concepts that are either used in or linked to the work of this thesis. A deeper introduction is given for the central parts. Other techniques are briefly explained to get a grasp of the concept and a theoretical background for the area.

The first section introduces software defined networks, and presents what they are and how they works. As most aspects are highly intertwined, a brief introduction will be given first. More technical explanations will be given later on for the specific applications used. The second section covers security aspects, both attack vectors towards SDN and STRIDE, a threat modelling tool. Galvanic isolation will also be covered in the second section as it is fundamental for the security of the network in the given setting. The third section covers the components of the network we built for testing purposes.

2.1 Definition of Software Defined Networks

There is no universally accepted definition of SDN as there are many different technologies which share some of the aspects. This section will describe the definition and characteristics used in this thesis.

2.1.1 Characteristics of Software Defined Networks

According to [1], there are four characteristics defining SDN. The four characteristics will be presented and explained in this section.

Separation of the Control and Data Planes

In a traditional network control messages are exchanged on the same network as data traffic. According to [1], in an SDN, the control and data traffic is normally placed in different networks. The separation can be achieved either by virtualisation, for instance VLAN, or by physical separation.

Figure 2.1 shows a comparison between a traditional network and a software defined network architecture. Looking at the SDN architecture, control plane communication can be isolated between the infrastructure layer and the control layer. Data plane communication take place on the infrastructure layer.

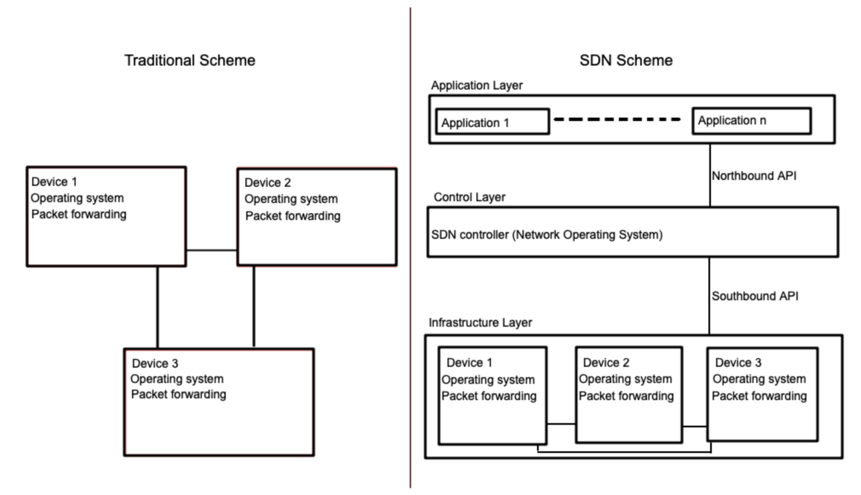

Figure 2.1: Illustration of the SDN architecture in relation to the traditional scheme.

Flow Based Forwarding Decisions

Another defining function of an SDN is the rules defining how the switches should react to incoming packets. In one SDN implementation, each switch in the network will have their own flow table with entries given to them by the controller, as it is explained in [8]. A flow table entry contains packet matching, actions and counters. The packet matching defines for what packets the actions should be applied to. The actions define how the packet should be handled, such as how it should be forwarded, if it should be dropped or forwarded to another flow table for further processing. The counters exists for the controller to gather statistics, and make decisions based on the counter values. The main difference compared to a traditional network is that forwarding is based on source, destination and additional parameters such as load balancing, rather than on a jump to jump basis.

Abstracted Control Logic

[8] also points out how SDN separates the control logic, deciding if and where to forward packets, from the act of forwarding packets. Some implementations, such as OpenFlow, allows the control logic to be placed in a different device. Other implementations, such as ForCES, keeps the control logic in the same device as packet forwarding while still abstracting the logic from the forwarding.

Abstracting the control logic allows for more modularity, and the same controller can be used for several forwarding devices. Furthermore, abstracting the control logic facilitates programming the network through network applications as they are provided a common interface. Reconfiguration also becomes easier as changes to the network does not have to be implemented separately in every affected forwarding device.

Programmable Through Software Applications

The last characteristic in an SDN that [8] mentions is how the network is programmable. The controller in a network has basic functionality for controlling a network. In order to improve upon it, it is possible to integrate the controller with software applications. This opens up for more network functions, e.g. more complex live monitoring, network intrusion detection systems and load balancing applications. By providing an API for the controller, it is possible to have the controller react to network events captured by the applications.

2.1.2 Components of Software Defined Networks

[1] splits an SDN into different separate components, and defines their respective responsibilities. Only the major components are mentioned here, as the other minor components are tied to other use cases where SDN can be utilised, but is not tied to this thesis work.

Southbound Interfaces

According to [9], a southbound interface provides a means of communication between the forwarding device and the controller. There are several available protocols for the southbound interface, such as ForCES and Openflow. However, Openflow has been able to gather a lot of attention in recent years and could be considered the standard protocol for southbound communication. The protocol used for the southbound interface has to be compatible with the forwarding device, so that the forwarding device can decode the instructions sent from the controller.

Hardware Infrastructure

The physical infrastructure of an SDN resembles the infrastructure of a traditional network. It can include components such as switches, routers and middleboxes. However, as explained in [10], the hardware has to be compatible with necessary SDN protocols. For instance, a switch in a network which uses OpenFlow as its southbound protocol has to be able to process OpenFlow packages and act accordingly. Many switches that support the Openflow protocol does also support normal routing, and is therefore called a hybrid switch.

Another crucial difference between an SDN switch and a traditional switch is that the SDN switch is a dedicated packet forwarding device. While a traditional switch sometimes has a lot of extra functionality, such as storm control or QoS applications, an SDN switch normally does not. These functionalities can instead be implemented through network applications and does not have to be located in the same physical machine as the switch.

Network Virtualisation

SDN can be used to provide cloud Infrastructure-as-a-Service, as explained in [8]. Instead of having a physical machine correspond to a single host it is possible to run several virtual machines on one physical machine. As these virtual machines may vary in number it is preferable for the network to be dynamically adaptable so that virtual machines may be added or removed without the need to manually reconfigure the network. Network virtualisation is the virtualisation of not only the hosts but also the network infrastructure, as

Controllers

The controller is in [8] referred to as the network operating system. It enforces the rules for the network topology by updating the forwarding devices over the south bound interface. Furthermore, it is often possible to have applications, such as an intrusion detection system, send updates to the controller which in turn updates the network. While SDN normally provide centralised control logic it is not necessary for the controllers to be physically centralised. It is possible to have different controllers for different parts of the network.

Northbound Interfaces

[8] explains the northbound interface as the interface between the controller and the network applications. As opposed to the case with the southbound interface there is not a standard implementation of the northbound interfaces. Rather each controller implementation defines its own northbound interface. Furthermore, different types of applications may have different requirements on the northbound interface. This makes standardisation more difficult and could result in a set of different APIs emerging for different types of applications.

Network Applications

[1] shows how applications can be implemented as a part of the SDN. Most of these focus (as off 2015) on traffic engineering, security and dependability, debugging, verification or measurement and monitoring. Some are also specific for data centres or wireless and mobile networks.

2.2 Security in SDN networks

Although SDN are exposed to new threats compared to a more traditional configuration, it has made some aspects of network security easier to enforce, as it is reasoned about in [1]. For instance, SDN can implement firewall functionality and access control, making policy enforcement easier. Traditional networks often lack the capability to automatically respond to network events. SDN, on the other hand, can, through the integration of network applications, reconfigure the network dynamically. Although SDN can be used to decrease the threat from some types of attacks, new threats are introduced, which is discussed in [11]. While some see the great possibilities of enforcing network security through SDN, others believe it is a much harder challenge to make a secure SDN than it is to make a secure traditional network.

2.2.1 Attack Vectors against SDN

In [12], seven attack vectors for software defined networks are presented. These attack vectors are not specific to an implementation of the southbound interface, which means that they are relevant for both OpenFlow and other southbound protocols. Some of them apply also to the traditional scheme, but could become more severe due to the centralisation of the control logic.

Forged Traffic Flows

A forged traffic flow is a flow entry installed on a forwarding device that does not belong there. Forged traffic flows can be caused either by a faulty device or by a malicious user. Depending on the type of forged flow, it can either consume resources of the network link, tap into the network communication or drop important traffic. No matter if it is a forwarding or drop flow entry, it is sure to disrupt the functionality of the network and is often used in Denial of Service attacks.

Attacks on Vulnerabilities in Switches

Attacks on vulnerabilities in switches have four main purposes: to drop, slow down, clone or deviate packets, as discussed in [12]. In [13], three types of attacks are discussed: flow rule flooding, switch firmware abuse, and control message manipulation. In the case of flow rule flooding the issue can be either contradictory rules or that the flow table is filled up by a large number of flows. Switch firmware abuse is concerned with exploiting limitations in certain switch models. Finally, control plane manipulation re-configures the data plane, either by removing already existing flows or adding new ones to a switch's flow table.

Attacks on Control Plane Communications

The control plane connects the SDN enabled switches to the controllers. In a basic set-up, a single controller can be connected to a single switch using a normal Ethernet cable. According to [14], the connection has been shown to be as reliable as the internal process in a traditional switch. However, according to [12], the link between the switch and the controller is not necessarily secure and connection must be sufficiently encrypted. Bailey and Stuart suggest that the connection between the controller and the switch can be secured using either certificates or MACsec.

Attacks on Vulnerabilities in Controllers

While a single controller is sufficient to control a small network, [11] discusses the negative aspects of having it as a single point of failure. According to [12], attacks on vulnerabilities in controllers are likely to be the most severe threats to software defined networks. A malicious application in a controller could do a lot of damage as it has access to the entire network, or at least a considerable part of it. An attacker with control over the controller can both gain access to valuable information by redirecting a flow and/or limit the functionality of the system.

Another issue proposed by [4], is that the control and monitoring functionalities are not necessarily decoupled in software defined networks. This reduces performance of the controllers as they have to simultaneously monitor traffic and send control messages to the network switches, which could make it easier for an attacker to overload the controller. Furthermore, to protect against some attacks, e.g. DDOS, it is desirable to have more information than what is available from the control traffic. OpenFlow controllers can provide flow metrics, but not per packet metrics, which is why a separate monitoring service must be used to discover these attacks.

Improper level of Trust

To ensure trust between the controller and the applications it is necessary to have reliable mechanisms in place. Traditionally different mechanisms, like passwords and digital certificates, has been used to ensure trust between applications and between network devices. Too little trust between the controller and an application would limit the functionality of the system, while too much trust would make it easy for an attacker to install his own applications.

Attacks on Vulnerabilities in Administrative Stations

Vulnerabilities in administrative stations are not specific for software defined networks, they exist also in the traditional scheme. However, if the administrative station is running the controller software, far more damage can be done to the network compared to a normal network.

Lack of Trusted Resources for Forensics and Remediation

Resources for forensics and remediation, such as logs and traces, can be very useful for analysing network problems. However, they are only useful if their correctness can be assured.

2.2.2 Improving the Security of SDN Components

This section presents possible methods on how to secure an SDN against the above mentioned attack threats. Some methods are more theoretical and practical implementations does not yet exist, but might be implemented in the future. All subsections below are discussed in [12], who argues that these methods are all valid protection mechanisms from attacks on an SDN.

Replication

Replication of controller software is a good counter measure to forged traffic flows and attacks on vulnerabilities in controllers. It also helps with improper trust between controllers and applications, and improves resources for forensic analysis. By replicating the controllers and applications, so that there are several instances of them, an error is less likely to be detrimental to the network. If one instance is faulty or compromised it can be discovered by comparing it to the other instances that are running. Furthermore, multiple instances removes single point of failure errors. The biggest shortcoming of this method is the performance loss by having to run multiple instances of the same software. Although there is an implementation for support for multiple controllers in Openflow, it only supports backup functionality. Comparing flow entries from different controllers is not yet possible, which makes the replication of controller software a future solution.

Diversity

While replication of the controllers limit some threats to the network, a vulnerability in one controller is likely to be present in all identical copies of the device. The same goes for network applications and other devices. By using both redundant and diverse software and hardware, vulnerabilities, such as implementation errors in the controller software, become less severe. Diversity can be applied to decrease the threat from attacks on control plane communications, vulnerabilities in controllers, vulnerabilities in switches, and vulnerabilities in administrative stations.

Self-healing

In case the network is compromised it could be possible to restore the system by replacing the affected components. Depending on the characteristics of the component this could be made dynamically. When a component is replaced it should not be replaced by an exact copy of the original one, but rather a different or modified one. The reason behind diversity in the recovery is to prevent the exact same attack being repeated.

Dynamic Device Association

In case a controller is compromised or faulty, the switches associated with it should be able to stop listening to that controller. In case there are several available controllers, the switches could associate with another controller instead. Furthermore, the associations have to be secure which could be accomplished by threshold cryptography and authentication. This is called dynamic device association and could help prevent attacks on control plane communications, and attacks on vulnerabilities in controllers.

Trust between Devices and Controllers

In [12], two different approaches to ensure proper trust between switches and controllers are proposed. One option is to assume all devices to be trustworthy until that trust is abused. The devices behaviour is monitored and a threshold implemented for how much abnormal behaviour is tolerated. If a device does not behave, it is quarantined by the other devices .

The other option is white listing. While not suitable for a larger network, and not very flexible, it could be sufficient for a smaller network.

Trust between Applications and Controllers Software

Autonomic trust management can be used to ensure a proper level of trust between the controller and the network applications. Using a model that supports autonomic trust makes it possible to adapt the trust level based on run-time performance. The level of trust in the model suggested in [15] is based on several different quality attributes, such as confidentiality, integrity and availability.

Security Domains

Different domains in the network may have different requirements on the security level. For instance, the requirements on the domain containing the controllers may be higher than the host to host communications. Furthermore, in a larger network containing several tenants the tenants may have different security requirements and, most likely, expect other tenants to have no interference in their part of the network. Sandboxing and virtualisation are techniques which may be used for security domains in SDN controllers. The goal is to allow only the minimal necessary communication between domains.

Secure Components

Secure components is a necessity for a secure system. In [12], the author argues that a trusted computing base (TCB) should be used to store sensitive data, such as cryptographic keys. A TCB is defined in [16] as a small amount of software and hardware that security depends on and is separated from the other parts of the system.

Software Updates and Patching

Software updates and patching should be used to remedy known issues in the switches, controllers, administrative stations and other devices in the network. It is therefore important to have efficient and secure means for installing updates, without making these paths available to attackers.

2.2.3 Using SDN to improve Network Security

The main benefits of using SDN is the ability to dynamically reconfigure the network, and the improved ability to gather and aggregate network information from different parts of the network. While many of the methods mentioned in this section are available also in traditional networks, the ability to efficiently gather information could make them more effective in SDN.

Access Control

SDN can be used to enforce access control, thus preventing an attacker from reaching critical areas of a network. The access control can be implemented on ports and/or VLANs, depending on the use case.

Several network programming languages, such as FML, Procera and Nettle, can be used to implement access control. OpenFlow controllers handle access control using access control lists (ACL). The ACLs define which IP and MAC addresses that may be used to access a given part of the network. There are also network applications, such as FlowNAC and MAPPER, which can be used.

Attack Detection

Attack detection is an important aspect of securing software defined networks. Although monitoring and intrusion detection systems are not exclusive to SDN, reacting to threats is aided by the holistic network view. Events in different parts of the network can be connected and reported to the controller, which can take appropriate action.

With the possibility of gathering statistics from flows in forwarding devices, new methods of detecting intrusions and attacks can be implemented. As shown in [17], it is possible to compare the rate between incoming and outgoing traffic for a set of flows, and by doing so detect DDoS attacks.

Firewall and IPS

Using OpenFlow, it is possible to implement firewall functionality in the switches. Stateful edge firewalls, that is firewalls which keep track of open connections and only allows known traffic, can be implemented directly in the forwarding devices. Furthermore, as the controller can be chosen to be as powerful as the network architect desires, it is possible to chose a more capable machine and implement firewall applications in the controller. Another option is placing the firewall application in a dedicated machine and connect it to the controller through the northbound interface, which is one of the methods explained in [14].

Forensics Analysis

Forensic analysis is aided by the possibility to securely store traces of network activity, for example by logging, consistent reporting and event correlation. The analysis strives to find the root cause of any network disturbances. Logging and forensic analysis is necessary also in traditional networks. However, SDN may make it easier to gather and aggregate information from diverse parts of the network, since the controller can maintain a full view of the network topology and the switches are keeping statistics on the amount of data passing through each flow entry.

Intrusion Tolerance

Intrusion tolerance is measures taken to keep the system operational despite intrusions in the network. As SDN can be used to dynamically reconfigure the network compromised hosts can be isolated.

Packet Dropping

Packet dropping is, as the name suggests, to let forwarding devices drop packets based on predefined rules.The rules can be either security rules or based on load conditions. An example use of packet dropping is to mitigate DoS attacks.

Separation of the Control Plane Communications

In a traditional network both data traffic, for instance traffic sent between applications on two different hosts, and control traffic, traffic containing network information sent between forwarding devices, are often sent on the same network. SDN can separate the control traffic making it more difficult for an intruder to tamper with it.

Rate limiting can be used to decrease the communication speed between the packet forwarding devices and the controller. It is implemented to prevent DoS attacks on the control plane.

Shorter Timeouts

An SDN switch keeps entries in its forwarding tables for a limited time. Reducing that time it follows that the switch has to ask the controller how to handle packets more often. However, longer intervals between timeouts could make the network more vulnerable to certain types of attacks. For instance, an attacker could attempt to gain access to flows by making the controller install rules which divert traffic. The attacker only has access to the flows as long as the rules diverting traffic is in place. Therefore a shorter timeout would mean that the attacker has to create more forged packets to maintain the path.

2.2.4 Galvanic Isolation

When a current is transported over a twisted pair cable an electromagnetic field is created around the cable. The field can be noticeable from a large distance and by recording the signal it can be possible to decode the signal transported over the cable. Furthermore, the electromagnetic field can affect metallic objects, such as water pipes, which in turn can carry the signal further. Galvanic isolation, in this context, means that there should be no current travelling between the communicating devices. If there is no current no electromagnetic field will be generated.

To avoid exposing the signals optic fibre is often used. As the signals are transferred in the form of light there is no current and no electromagnetic field. In [18], it is explained that he Swedish Armed Forces require communication without approved protection of the signals to be transferred over optical fibre.

2.2.5 STRIDE

[19] explains that STRIDE is a security threat model originally developed by Kohnfelder and Garg for Microsoft. It is intended to be used in the design process so that potential threats to a system can be spotted early on. STRIDE is an abbreviation for the six suggested threat categories:

- Spoofing of user identity
- Tampering with data
- Repudiability
- Information disclosure
- Denial of Service
- Elevation of privilege

Spoofing means pretending to be someone you are not. This category can be expanded to include also other types of spoofing, such as pretending to be the controller or modifying IP addresses.

Tampering is a violation of the integrity property. It is the modification of data and is not necessarily dependent on read access.

Repudiability is about not being able to hold a user accountable for its actions. Non-repudiation means that the action can not be denied and makes it easier to take action against the undesirable act.

Information disclosure is a violation of the confidentiality property. Data can be read from files, but meta data, such as type of traffic and load, can also be of interest.

Denial of service is a violation of the availability property. Most often it refers to attacks where a unit on the network is overloaded with traffic which makes it unavailable to other users.

Elevation of privilege is when a user gains more access than intended. It violates access control and authentication.

2.3 Components of the prototype Network

This section will cover the software components used to setup the network.

2.3.1 Controller - Ryu

Ryu is an open source SDN controller, written in Python. It is under active development and is one of the more popular SDN controllers in use today. It has support for the latest Openflow version 1.5, and also the previous releases of Openflow.

Ryu is built with modularity in mind. Modularity in this context means that Ryu takes code modules as input when started and executes the code from these modules. This gives flexibility when running Ryu and it makes it easy to modify the behaviour by loading another set of code modules.

In addition to the code modules that Ryu uses, it also handles a lot of functionality under the hood. Setting up new communication channels with forwarding devices, updating the current network topology are all automatically done by Ryu and does not need to be implemented by hand by the network administrator.

2.3.2 Southbound Protocol - OpenFlow

Openflow is the backbone for communication between the controller and forwarding devices in a software defined network. This section will explain the idea behind it and its internal design.

About

Openflow was developed at Stanford University in 2008 as an answer to the question; *As researchers, how can we run experiments in our campus networks?*. In 2008, there was no practical way to test new network protocols in a real production network, without disturbing the real traffic. Not being able to properly test new protocols was a sure way to decrease innovation, and [20] describes that scenario as the basis for why Openflow was developed.

Protocol Functionality Overview

Openflow works by modifying a switch's flow tables, which contains flow entries. A flow entry is in this context a rule for how a switch should process packets, and can be read more about in the upcoming section. With Openflow, researchers would be able to create and control their own flows specified for their research, deciding the route and how the network packets should be processed. The normal data traffic would be handled as normal, isolated from the research experiment. This method enabled researchers to test new functions and protocols without disturbing the normal traffic, other than increasing the workload for the routers and switches.

The flow table of a Openflow switch contains all flows for a network. A flow in this context means how some specific traffic is to be processed in the network, and can be split into three parts; *matching, actions* and *statistics.* Figure 2.2 shows how the flow tables are used for an incoming packet to a switch.

Matching The match of a flow decides what traffic the flow is valid for. It is built from fields and values of network protocol headers. Different header fields can be combined to match specific communication, or be broadly defined to match more traffic. As an example, a flow match can be all traffic to an IP address, or all traffic on the same VLAN with a specific MAC destination address. The ability to build very specific matches makes it practical to create a flow that only will match on the exact traffic that is defined.

Action The action part of a flow decides how the flow is to be processed when the match for a flow is triggered. In Openflow 1.3 and forward, there are four actions that must be implemented by every switch that supports Openflow, and more optional actions that are normally implemented too. The four actions that are required are *output*, *send to controller*, *drop* and *group.*

The *output* action specifies what port the incoming packet should be forwarded out to, much like how a normal layer 2 switch functions. It is possible to send out

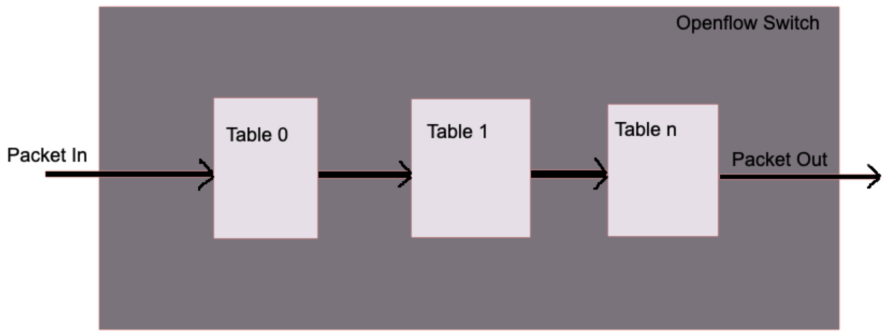

Figure 2.2: An overview of the Openflow pipeline, which shows how incoming packets are handled by the switch.

a specific port or flood the packet, meaning that it is sent out every port except the port it came from.

The *send to controller* action encapsulates the incoming packet with a ID and sends it to the controller entity, which decodes it and handles how to process the packet. This action is mostly used for the first incoming packet with no matching flow, so the controller can decide how to handle this new traffic, and add a new flow to the switch's flow table so it does not have to ask the controller what to do with further packets of the same match.

The *drop* action simply drops the incoming packet and does not forward it to anyone. This action is often used as a security measure or for limiting broadcast messages in a big network.

The *group* action. This sends the packet to be processed by a specific group entry. A group entry is defined by four parts:

- Group identifier
- Group type
- Counters
- Action buckets

The main use of groups in Openflow is to gather logic in groups and not have it in separate flow entries. This simplifies both development for network administrators and makes it easier to maintain the flow tables

One of the more common actions that are not mandatory to implement is the *normal* action. This tells the switch to handle the packet as a legacy forwarding device would. This action is how a hybrid switch functions. If it is a layer two switch, normal switch behaviour is expected. If the forwarding device is a router, then the packet is processed as it would be by a legacy router.

Statistics The last part of a flow are the statistics. This field contains counters for the number of packets and the amount of bytes that has gone through the flow. It does also contain timers for when the flow was added to the flow table and when a packet last was matched to it. The statistics help building network functionality, e.g. load balancing and removing inactive flows.

2.3.3 Switch - Open vSwitch

About

Open vSwitch is a virtual multi layer switch. It behaves like a normal physical switch, except that it is implemented in software. Open vSwitch is capable of switching between both virtual network interfaces and physical ones, making it suitable to route both between virtual machines and physical machines, as explained in [21]. Open vSwitch does support the Openflow protocol, which makes it suitable for software defined networks.

How it works

Open vSwitch makes use of the integrated Linux network bridge when run on a Linux system. A network bridge is a connection between two or more local networks. From the participants perspective the connected networks appear to be the same. Utilities, such as traceroute, does not show the bridge as it operates on layer 2 and traffic is forwarded transparently. Open vSwitch also supports layer 3 routing. This is a truth with modification, since Open vSwitch relies on layer 3 routing built into the Linux kernel. To be able to perform layer 3 routing, it must be enabled in the underlying kernel, as explained in [22].

Openflow support

As mentioned in the above subsection, Open vSwitch supports Openflow. To make configurations and show information about Openflow, the tool *ovs-ofctl* can be used. This tool makes it possible to add new flow entries, show flow tables, change which Openflow versions that are allowed and more. This tool provides an easy way to quickly test a new setting of the network.

2.3.4 Intrusion Detection System - Snort

About

Snort is an intrusion detection system (IDS) which can be used to monitor and find attacks in a network, but it can also be used for other purposes, as described in [23]. When running as an IDS, Snort detects traffic that is not allowed and alerts the system administrator or, in this case, the controller. Snort works by analysing incoming packets and compares it to predefined rules. It also provides the possibility to detect known attacks by comparing incoming packets' data with previously known attack patterns. It is possible to run Snort as a monitor service, which only logs traffic and does not analyse it, or as an intrusion prevention system

(IPS). The difference between the IDS and an IPS modes is whether Snort discards the sent traffic or alerts the controller. Running it in IPS mode is only useful if traffic has to pass through Snort to get to its destination. The monitor mode is useful for testing purposes and to verify network configurations.

Placement and different modes

When analysing packets, Snort consumes resources. Depending on the rules configured, it can either be an irrelevant amount or slow down a system a lot. This requires a system administrator to balance performance against security. By choosing the rule configurations smart, it is often possible to get good performance and security, i.e. by just having rules customised for the specific service. The placement for where to install Snort is also important. Possible placements and the advantages and drawbacks for each placement are discussed in [24]. To install it directly on a computer only allows Snort to analyse the traffic to and from that computer. On the other hand, if it is installed on a dedicated server meant to just run Snort, then all traffic to analyse must be mirrored to this dedicated server, increasing the overall traffic in the network environment.

Usage with other applications

Snort is often used together with other applications. Together with a firewall, it is possible to use Snort as a second layer of defence if the firewall would go down or be poorly configured. To integrate Snort into an SDN, different modes are supported. It can be directly connected to Ryu, as explained in [25]. It can also be used as a stand alone application, depending on the use case.

It is explained in [25] explains how Ryu supports Snort through integration using a Unix socket, and the application Pigrelay can be used if Ryu and Snort run on different physical machines.

2.3.5 Media Converter and Optic Fiber

As described in [26], a media converter is a device that can convert a signal from one medium to another. This makes it possible to use different media formats when connecting a network, since the signals can be translated to a medium that each network section can understand. In practice, it is often used to convert between fiber optic signals and an copper medium, which is what a normal local network utilizes for wiring.

Optic fiber is a medium which can be utilized for communication. It works by having a transmitter send pulses of light through a fiber cable, which are read by a decoder in the receiving end. The advantages of fiber optic communication compared to copper cables, are speed and distance, which is shown in [27]. The rate of data that fiber optic cable can put through is magnitudes bigger compared to a normal twisted pair cable, and the distance on which is can send information without repeaters is much further.

Chapter 3

Implementing an SDN

This section will show how a network was set up and configured to test our solution for isolating nodes. The network will be a simple proof of concept and more complex aspects will be saved for the discussion, as noted in the limitation section.

3.1 Architecture

The architecture of the physical network is as simple as it can be while still acting as a proof that it is possible to isolate malicious hosts. The network consists of three Intel NUCS and one Raspberry Pi. Two of the Intel NUCS acts as general processing units, which would in a real life scenario have a dedicated task and communicate with each other. One Intel NUC is configured to act as the switch of the network, connecting all hosts with each other. The Raspberry Pi is dedicated to run Ryu, the controller software. Figure 3.1 shows a figure of the network's topology.

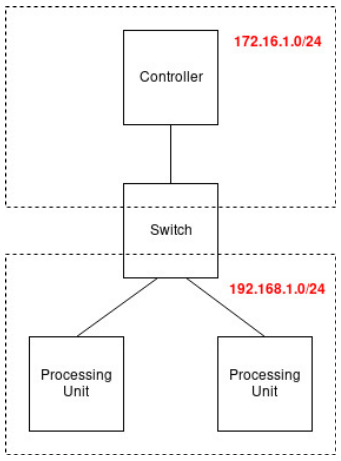

Figure 3.1: Network architecture

3.2 Software and Hardware Choices

3.2.1 Controller

The controller is one of the central parts of an SDN architecture. Since it is involved with every decision made in a network, it is important to choose the right software. While there are many implementations of controller software to choose from, many are no longer maintained or suffer from lack of documentation. Many of the controllers we examined did also not support version 1.3 of OpenFlow, which is one of the most widespread releases, and the one used in this thesis. The controllers that still are maintained and supports version 1.3 did roughly provide the same functionality. While the performance of the controller is of big importance in a real world scenario, where maximum throughput of packets and a low latency for flow modifications are desired, this proof of concept network has no need for that. While there exists benchmarks for controller software, as seen in [28], these results were not taken into account for this thesis, since performance not a factor in this experimental network. Instead, the decision for what controller to use was based on documentation available and on what language to write the controller functionality in. The controller software with the best documentation was Ryu, and it is written in Python. A built in integration with the IPS Snort also comes with Ryu.

3.2.2 Switch

The switch for the physical network have to support Openflow and have at minimum three ethernet ports to connect all components with each other. While there are some production grade switches with support for Openflow and have a sufficient number of ethernet ports, they cost several thousands of dollars. The company Northbound Networks [29] has two Openflow switches meant for hobby projects, and would work for this project. However, as the production is done in small scale in Australia, and due to the time constraints of the project, acquiring these switches were unfeasible. Instead, an Intel NUC7i7DNH was configured with Open vSwitch installed and Rj45 to USB converters were added to get enough ethernet ports onto the NUC. The performance of the Intel NUC together with Open vSwitch can not match a real hardware switch, but as mentioned before, the performance is not important for a proof of concept network, so it does not matter.

3.2.3 Processing Units

The performance of the processing units in the network are not prioritized, since no demanding processes will run on them. Two Intel NUC NBR running CentOS are used as these are readily available. CentOS were chosen due to its similarity to RHEL which is common in production environments.

3.3 Installation and configuration

This section will cover the steps taken to set up the network.

3.3.1 General processing units

The computers that were to act as general processing units did not have any particular requirement for operating system, but centOS 7 is used by many production grade system, which is why that was chosen. A fresh installation of centOS 7 were installed on the computers. After installation, a static IP address were configured on each of these. The IP address chosen were 192.168.1.98/24 and 192.168.1.99/24. The software program iperf3 was shipped with the operating system, but hping had to be downloaded manually.

3.3.2 Switch

The computer to act as a switch was one of the Intel NUCS. At first, the Raspberry Pi was chosen to act as a switch but it quickly showed that it was much faster to work on the Intel NUCS, since it had more powerful hardware, so the switch hardware changed. For compatibility, Ubuntu was chosen for this NUC as the operating system. While open vSwitch is compatible with more than Ubuntu, we found the most information and guides on open vSwitch for Ubuntu. This NUC were configured with 2 USB to rj45 adapters to allow all computers in the network to connect to each other.

The Open vSwitch software was downloaded and installed. A network bridge was created with the utility tool *ovs-ctl*. The command to run was

```
ovs-ctl add-br br0
```

This creates a network bridge that Open vSwitch can utilize. The interfaces from the USB to RJ-45 adapters were added to this bridge with the command

```
ovs-ctl add-port br0 x
```

were x is the name of the interface. To utilize Snort later on, an internal port named "snort0" was also added to the bridge with the following command.

```
ovs-ctl add-port br0 snort0 type=internal
```

The IP address of the network bridge was configured to 192.168.1.100 and all interfaces on the network bridge were configured to not have an IP address, since they are connected to the bridge. The network bridge was brought up with the command

```
ip link set br0 up
```

To configure open vSwitch to find the controller software, the command

```
ovs-vsctl set-controller br0 tcp:172.16.1.X:6653
```

This configures open vSwitch to try to connect to that IP address and on what port the controller software is listening on. The command

```
ovs-vsctl set controller br0 connection-mode=out-of-band
```

was executed after, which tells open vSwitch that the controller resides on a different network than the subnet the Open vSwitch resides in.

Configuring TLS

To not have the controller and switch communicate in clear text, a TLS connection was setup between the two nodes. The utility tool *ovs-pki* was utilized to create keys and certificates, on the same computer that has open vSwitch installed. To create a Public Key infrastructure, the command

```
ovs-pki init
```

was executed. This creates certificate authority files for both the controller and switch. *cacert.pem* is the root certificate for the controller certificate authority and *cakey.pem* is the private key associated with the root certificate. Analogous files are created for the switch. These files are created in the directory controllerca and switchca, respectively. Next step was to create a private key and a certificate for the controller, done with the command

```
ovs-pki req+sign ctl controller
```

which creates a public key and a certificate for the controller entity. The controller's private key and the certificate authority file were copied to the raspberry Pi, running Ryu, with scp.

A private key and a certificate was also created for the switch, with the command

```
ovs-pki req+sign sc switch
```

To get Ryu and open vSwitch to utilize these certificates when estblising a connection, the command

```
ovs-vsctl set-ssl /etc/openvswitch/sc-privkey.pem
/etc/openvswitch/sc-cert.pem /etc/openvswitch/cacert.pem
```

was executed. This command specifies the certificate for the switch and its private key, and also the certificate authority that verifies the switch.

To make the connection between open vSwitch and Ryu use TLS, the command

```
ovs-vsctl set-controller br0 ssl:172.16.1.99:6653
```

was executed. To start the controller with TLS, the keys and certificate must be specified as with this command

```
ryu-manager --ctl-privkey ctl-privkey.pem \
            --ctl-cert ctl-cert.pem \
            --ca-certs
            /usr/var/lib/openvswitch/pki/
            switchca/cacert.pem \
            --verbose ModuleName.py
```

After this command is executed, Ryu will connect to the switch and communicate over TLS instead of normal TCP/IP.

Verifying setup

The command

```
ovs-vsctl show
```

was used to verify that open vSwitch had been configured right. A printout of the command can be seen here:

```
c043847a-959e-4768-ad23-fffc87152080
    Bridge "br0"
        Controller "ssl:172.16.1.10:6653"
        Port "enx58ef68b551ac"
            Interface "enx58ef68b551ac"
        Port "enx58ef68b55017"
            Interface "enx58ef68b55017"
        Port "br0"
            Interface "br0"
                type: internal
        Port "snort0"
            Interface "snort0"
                type: internal
    ovs_version: "2.11.1"
```

3.3.3 The controller entity

The controller was chosen to run on the Raspberry Pi. After a fresh install of Raspberrian OS, the Raspberry Pi was configured with the IP-address 172.16.1.99. The software Ryu was downloaded and installed. The code module with the isolation method was also copied to the Pi over scp, since it was developed on another computer.

3.3.4 Installing and configuring Snort

Snort was installed on the same computer as Open vSwitch, the Intel NUC. To allow snort to listen to all traffic on the data plane was mirrored to the "snort0" port. Snort was configured to listen to the port "snort0". A rule that does not allow ICMP echo replies was added to the rule set that Snort utilizes to determine what traffic is allowed. To send the Snort alerts to Ryu, the program Pigrelay was used, which is recommended by the Ryu development team to send alerts to Ryu when Snort is placed on a different computer.

3.3.5 Connecting and starting

After all computers were setup and configured as described in the above section, the network setup was connected. The open vSwitch software is started with the command

```
ovs-ctl --system-id=uuidgen start
```

After open vSwitch is started, the controller software is started on the Raspberry pi with the command

```
/home/pi/ryu/bin/ryu-manager --verbose ModuleName.py
```

where ModuleName.py is the module we have developed to isolate nodes. The verbose option will print out network events and is used for debugging the network. After a short time, Ryu will confirm that a connection is established and prints out information about the connection to the standard output.

Chapter 4

Testing - Methodology and Results

4.1 Methodology

The following sections will describe how each of the research questions have been evaluated and the motivation behind it. The first question will be answered through a theoretical analysis. The second and third question will be answered by tests conducted on the physical set up of a network and be reasoned about in the discussion chapter.

4.1.1 Network Security

The first research question, *How can SDN be used to shape the network dynamically in a way that improves overall security without introducing new significant threats?*, is answered in the discussion based on the theoretical background. A direct comparison to a traditional network is not possible to perform, since an SDN has a different attack vector than a traditional network. Some overlap of the attack vectors does exist and in the discussion we will reason wether an SDN are more protected against these overlaps. The attacks that only are possible to perform on an SDN will be discussed wether they are a significant threat or not. This will provide a complete theoretical analysis of the system and should answer the question.

4.1.2 Isolation of Nodes

The second research question, *Given that a node is compromised, can the intrusion be contained by redefining the network around that node?*, will mainly be tested using the physical set up. The isolation is tested in two different ways.

The first method is attempting to send a packet, which is forbidden by the controller, through the switch. This typ of illegal packet should be instantly blocked and never reach the destination. The second method is blocking packets through the intrusion detection system, Snort. A rule forbidding ICMP traffic will be added to Snort. Ping will then be used between the two processing units. As the packets are not considered illegal when first sent, a few packages should be allowed to pass before the switch flow table is updated to block all traffic from the hosts sending ICMP traffic.

Hardware vulnerabilities will be dependent on the hardware the network is implemented on. As the hardware used in possible future implementations will differ from the hardware used in this thesis hardware vulnerabilities will not be evaluated.

4.1.3 Galvanic Isolation

The third research question, *Is it possible to build an SDN where the nodes are galvanically separated?*, will be evaluated in two steps. First the twisted pair cable between one of the processing units and the switch will be replaced with two media converters and an optical fiber. The fiber optic cable has been previously tested and is known to galvanically isolate nodes its connected to, since it is based on optics and not current. The reason behind the test is to show that the media converters become transparent units in the network and do not interfere with data plane communications.

The second step is to replace the connection between the node running Open vSwitch and the traditional switch with media converters and an optical fiber. This is done to show that the media converters do not interfer with control plane communications. The same tests as for isolation of nodes will be conducted for galvanic isolation. This is done to see if the media converters slow down the system giving an attacker more time to send malicious traffic.

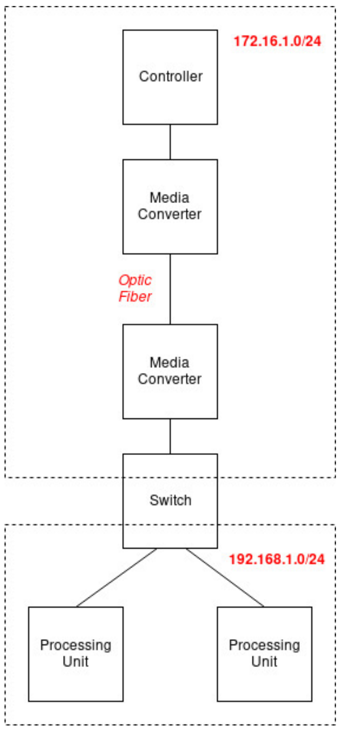

Figure 4.1: Galvanic isolation of the controller using optical fiber

4.1.4 Method

To perform the testing in practice the tool hping3 was used. The following steps were taken:

- Legal packets were sent from processing unit A to processing unit B. The packets used were TCP packets sent to and from port 5001, which the controller allows.
- Illegal packets were sent from processing unit A to processing unit B. The packets used were TCP packets sent to port 6001 from port 5001, which the controller does not allow.
- Legal packets were sent from processing unit A to processing unit B again, this time to verify that A was blocked. Again TCP packets to and from port 5001 were used.
- The system was restarted to clear all flows
- ICMP echo request with no payload data were sent from processing unit A to processing unit B. These packages causes snort to alert the controller.
- The system was restarted to clear all flows
- ICMP echo request with 1400 byte payload data were sent from processing unit A to processing unit B. These packages causes snort to alert the controller.

4.2 Results

In this section the results from the testing are presented. They are separated into the categories isolation with pre-defined rules and isolation with IDS, rather than isolation and galvanic separation. This is done to facilitate comparision between the cases.

4.2.1 Isolation with Pre-Defined Rules

To guarantee that the processing units not were blocked legal TCP packets were sent from the processing unit with IP address 192.168.1.98 and source port 5001 to the processing unit with IP address 192.168.1.99 and destination port 5001. This was done using hping3. The flag *–fast* was set to achieve a packet transmission rate of ten packets per second.

```
#hping3 --fast -p 5001 -s 5001 -k 192.168.1.99
```

For all three cases, the packet loss was close to zero percent.

Once it had been established that the packets were transmitted correctly the destination port was changed to 6001, which should cause the controller to block the transmitting processing unit.

```
#hping3 --fast -p 6001 -s 5001 -k 192.168.1.99
```

In all three cases all packages were blocked and the package loss 100 percent.

Finally the destination port was switched back to 5001. However, this time the processing unit 192.168.1.98 was already blocked by the switch.

```
#hping3 --fast -p 5001 -s 5001 -k 192.168.1.99
```

As expected, this resulted in a 100 percent package loss in all three cases.

4.2.2 Isolation with IDS

The tests involving ICMP packages were performed twice to get an indication whether any conclusions about performance could be drawn from the results. As can be seen from 4.1 and 4.2, no conclusions can be drawn as to how the media converters affect response time. However, it is clear that some traffic will be passed before snort has alerted the controller and the controller has updated the switch's flow table.

	Attempt 1	Attempt 2
No Galvanic Isolation	10	12
Galvanic Isolation between Switch and Controller	8	16
Galvanic Isolation between Switch and Processing Unit	16	15

Table 4.1: Number of echo replies received before the processing unit is isolated when 10 echo requests with no payload are sent per second.

	Attempt 1	Attempt 2
No Galvanic Isolation	16	9
Galvanic Isolation between Switch and Controller	11	13
Galvanic Isolation between Switch and Processing Unit	9	14

Table 4.2: Number of echo replies received before the processing unit is isolated when 10 echo requests with a payload of 1400 bytes are sent per second.

Chapter 5

Discussion

The network will be broken down into its vital components which will be evaluated separately. While this results in some overlap it also makes it less likely that any threat is forgotten. The vulnerabilities in each asset will be evaluated using the STRIDE framework.

After the evaluation is completed the applicability of the attack vectors suggested by [12] will be discussed.

5.1 Evaluation of Implementation using STRIDE

Four major assets for the function of the network have been identified.

5.1.1 Controller

The controller is placed, together with the switch, in its own subnet. The subnet is separated from the data plane and is not connected to the Internet. The communication between the switch and controller is over TLS with mutual authentication, which means that both the controller and switch are authenticated before connecting to each other. This would make it next to impossible to spoof the controller, since spoofing the controller without a valid certificate would be detected when the connection is established.

Directly tampering with the data of the control plane should be unachievable. There is a possibility of indirect tampering by sending packets through the switch to manipulate alerts sent to the controller. As access rights are based on MAC address rather than physical port, it could be possible to spoof another processing unit's MAC address to make the controller exclude that unit from the network.

As the implementation only contains one controller and switch there should be no issue with repudiability. A message to the controller is bound to come from the previously authenticated switch. This logic holds even with many switches and controllers, since they all will be authenticated before they can start to communicate. There is also no direct information disclosure from the controller. All information sent to the switch is encrypted and can only be decrypted by the switch that it is destined for.

Although our test method has not revealed it, it is possible that the network is susceptible to a denial of service attack. If a lot of traffic that does not match any flow in the flow table is sent to the switch, the switch will send a corresponding number of *PACKET_IN* to the controller. This means that an attacker who sends packets that change over time, e.g. a new MAC destination address for each packet, would fill up a switch's flow table rather quick. When a switch's flow table is full and a new flow is to be added, the switch does not allow it and sends an error message: *OFPFMFC_ALL_ TABLES_FULL*, to the controller. Not being able to add new flows means that traffic that does not already match any existing flow entry can not be forwarded by the switch. In most scenarios, this is not a problem since the normal traffic most likely already has their corresponding flow entries installed in the flow table before the attack is executed. If the attack is performed before the expected traffic has their corresponding flow entries installed in the switch, as in just when the system starts, then the attack might succeed to stop the expected traffic depending on the size of the flow table and how fast the expected traffic starts when the system boots up. While traffic analysis can be used to try to determine if a flow installation is valid or not, i.e by looking at the rate of incoming flow entries, there is no way to guarantee that the analysis will catch every attack attempt. False positives of fake flow entries are also a consideration that would slow down the network performance, depending on how the analysis is implemented.

When looking at escalation of privileges, it is not a consideration in the controller software. The different privilege levels are rather on the computer running Ryu, and then if the attacker can get root access or not. With root access, the attacker can of course do more harm on the system. While it is not sure if Snort or Open vSwitch can help an attacker get root access to the computer running the controller software, we have not found any known attacks for this purpose.

5.1.2 Switch

As there is only one switch in the physical set up, the switch is necessary for the operation of the network. It has one network interface to the control plane and two to the data plane.

Since the switch is authenticated before connecting to Ryu, spoofing of the switch is not a realistic scenario. With regards to the processing units in the network, they only have one physical connection to the switch computer, which makes spoofing of the switch most unlikely without physically changing the network topology.

Although no know vulnerability has been found in the Linux bridge, there is a risk of an unknown vulnerability in Open vSwitch or one of its dependencies. If an attacker gained administrator privilege in the machine running Open vSwitch packets could be modified before they were sent to the receiver. It would also be a simple task to listen to all traffic on the network and send it to an application of the attackers choice.. Although it would be difficult to know who had gained access, the issue would be easy to locate since there only is one switch.

The testing of the set up has not revealed any susceptibility to DoS attacks in Open vSwitch. However, if the switch would have to make many requests to the controller it could temporarily slow down the processing of incoming packets, as explained in the controller subsection. Furthermore, if an attacker got access to the the computer running OvS, they could configure existing flow entries and add new ones, without communicating with Ryu. This would of course make it possible to perform a DoS attack.

5.1.3 Snort

While Snort is not necessary for the network to be operational, the intrusion detection system is necessary to detect attacks that can not be identified by a single packet, e.g. a SYN flood.

Applying the STRIDE framework we find that spoofing of Snort makes little sense. However, with root privileges it could be possible to install another IDS and connect that to the controller, before Snort has connected to Ryu. The other IDS could be configured to send alerts on normal traffic and therefore disturb the networking functionality, depending on how Ryu is configured to handle alerts. For this attack to succeed, the other IDS would need knowledge of what port Ryu is listening on for a connection to Snort, and to connect to Ryu before the original Snort instance does. To fulfill both these requirements are improbable and the spoofing scenario of Snort is not feasible.

Tampering with the data, in this case the rules Snort apply for when to alert the controller, is a more probable scenario than spoofing the Snort instance. This would first require access to the computer running Snort and also authority to modify the configuration file that Snort utilizes. While possible in theory by of the system, it should be unfeasible in practice.

As there are no mechanism to ensure trust between Snort and the controller, repudiability is limited. At the same time, like the case with the switch, there is only one intrusion detection system located in one physical machine so an error should be easily traced.

Access to the machine running Snort could reveal configuration files and logs. What information that is leaked also depends on the content of the configuration file. Some rules might require to specify what ports that are allowed, which an attacker can make use of in further attack scenarios. This problem is easily solved by only allowing an administrator get read access to the configuration file.

When looking at a DoS aspect, it is possible to overload Snort. Depending on the amount of traffic and how extensive the configuration file is, Snort can easily be overloaded. Theoretically, an attacker could start to send a huge amount of traffic and hope that Snort's resources are overloaded. When overloaded, Snort will try to catch up while new traffic is buffered to be analyzed. While Snort is busy analyzing the huge amount of new traffic, a new attack could be launched. This new attack would not be discovered until snort has finished analyzed the old traffic.

When looking at escalation of privileges, the same reasoning as before can be made. If an attacker got access to an administrator account, more damage and leaks can be done. From what we can find, Snort does not in any way help an attacker to get higher access levels in the operative system.

5.1.4 Processing Units

Out of the four listed assets the processing units can be considered less critical than the other three. If one of the processing units would be compromised, and the intrusion detected, the functionality of the cluster would decrease but it would not prevent the system from functioning.

As has been described as an issue in the previous sections, spoofing of a processing units IP address is possible. Spoofing could, in our implementation, be used to block other nodes by sending traffic not allowed after taking their MAC address. It could also be used to get access to the system again from a node that is already blocked, by taking another MAC address which is allowed. This would of course require the attacker to have knowledge of what MAC addresses that are allowed, but it is a theoretically possible scenario.

With regards to tampering of data, it is difficult to conclude what data could be affected. Most types of communication is limited between the processing units. How a user could exploit the network and tamper with data depends on the applications on the processing units.

The issues regarding information disclosure are similar to the ones regarding tampering with the data. As mentioned in previous sections there could be other possibilities of revealing information, such as the IP addresses of the processing units.

When looking at repudiability, there is a problem since no proof of authorship is used in the system. Still, it is not really a problem for the system as a whole. If an attack is detected, it does not matter if node A denies sending messages sent or not. The focus should be to stop the attack and make sure it can not happen again. To put effort into providing non-repudiation is the wrong way to stop attacks from happening.

The processing units should be vulnerable to a DoS attack only if the IDS is not properly configured, or a new DoS attack is discovered that Snort is not configured to detect. There is also the case of spoofing a processing unit's MAC address and sending illegal traffic, which would in our implementation block the real node.

An escalation of privileges could be an issue depending on the software running on the processing units, which is not known beforehand. If different privilege levels exists in the running software, then it is assumed that there are sufficient methods to stop a lower privileged user to get access to information which require a higher authorization level.

5.2 Looking at the Attack Vectors

Out of the seven attack vectors most have already been covered by the STRIDE analysis. In the previous section we saw that forged traffic flows can cause problems in all parts of the network.

As for attacks on vulnerabilities in controllers and switches the STRIDE analysis showed potential vulnerabilities in single devices. Some possible solutions are replication and diversity to make the network more resilient, which is also explained in [12]. These possibilities will be covered in the next section, improving the network.

Attacks on control plane communications should difficult to achieve in practice if the switch is configured correctly. In order to target the OpenFlow communication, access to the control network, which is physically separated from the data plane, is necessary. If an attacker is able to gain access to the control plane network they must first get access to the switch, which also should not be possible. The switch should only function as a forwarding device and it should not be possible to connect to potential management connections from the data plane.

The implementation lacks a method to ensure proper trust between all components. The switch and controller is authenticated before establishing a connection, so trust between the two parties is established. The processing units and Snort are not authenticated before connecting, but could be implemented with certificates.

Some logs are produced by Snort. However, better logging should be implemented if a similar set up were to be used. Logging all traffic and storing it helps with detecting how intrusion and intrusion attempts were made, which can help build a more secure system for future use.

5.3 Improving the Network

In this section possible improvements of the network will be discussed. The discussion will mostly focus on adding additional units to create redundancy, as suggested in [12]. However, the placement of the IDS will also be discussed.

5.3.1 Adding Switches

The physical setup suffers from a lack of redundant switches. If the switch is compromised or fail, all traffic on the network is compromised or interrupted. To bypass this issue, it could be possible to add additional switches, as illustrated in 5.1. The intrusion detection system can either be moved to a different machine or duplicated in all of the switches, see 6.5 Moving the IDS.

Inserting an additional switch has several complicating factors. First of all, each PU should be directly connected to both switches, putting some restraint on required ports. Second, to be able to isolate a compromised switch it would not be sufficient to use the OpenFlow protocol. A compromised switch should not be trusted to block itself from communicating. There are two feasible solutions to this issue. One possibility is to block communication with the switch in all connected nodes. This would require a message to be sent to all connected nodes when an intrusion is detected, so that the affected nodes shut down their network interface to the compromised switch. At the same time, this message would go through the compromised switch which can just drop that message, so it is not a reliable method. The other method is to cut the power to the switch. Since the switch is considered compromised, it can not be a message that shuts it down since it can be ignored by the switch. Manually cutting the power is the only way to guarantee that the switch turns of.

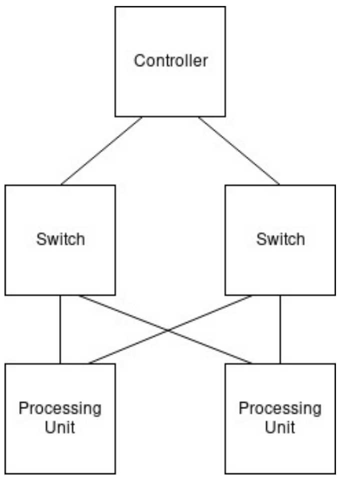

Figure 5.1: Adding a redundant switch

5.3.2 Adding Controllers

The physical set up lacks redundant controllers. Adding additional controllers would improve the stability of the network. Adding one extra controller should make it possible to determine if one of the controllers is faulty. By using three controllers it should be possible to determine which controller is faulty. The reasoning is based on employing the controllers with equal priority and using an oligarchy decision model. That is, if a majority of the controllers push the same rule it is determined to be a correct rule. If three controllers is used and one pushes a different rule that controller should be investigated. As well as providing an error check mechanism, more controllers also means that the system is more reliable. If a connection to one controller is down, the switch can still utilize the others and query them for flow entries. In a system with only one controller, the single connection to one controller becomes a single point of failure, which is much undesired in a safety critical system.

While simple in theory, using several controllers poses some practical challenges.

First, there could be synchronization problems for new flow entries. The switch would have to wait until it has received at least a majority of the flows from the controller. If there are a lot of requests to the controllers the responses could arrive out of order. Even though the issue should be easily handled with a buffer, chances are response times would deteriorate. How much of a problem a time increase on new flow is depends on the system, but it should anyhow be known.

Second, the controllers must support the same Openflow protocol version. While not a big problem, it does decrease the possible choices for controller software, since different controllers support different Openflow versions.

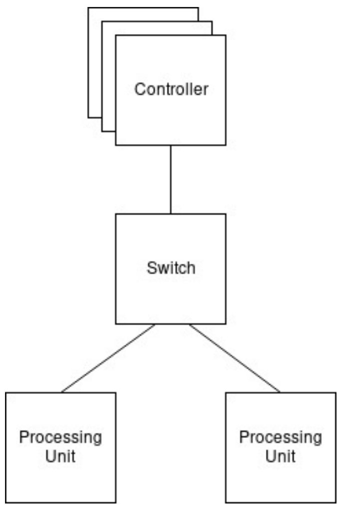

Figure 5.2: Adding two redundant controllers

5.3.3 Adding Processing Units

Adding additional processing units should improve the networks resiliency. If there are two processing units and one goes down 50 percent of the network's capacity is lost. If there on the other hand are 20 processing units only 5 percent of the capacity is lost when one goes down. By using a sufficient number of processing units and implementing self-healing it could be possible to obtain not only graceful degradation, but even automatically restore the network after an attack. At the same time, adding more processing nodes does not automatically increase the capacity of the system if a node goes down. Depending on what type of system and program running, everything can break as soon as one node goes down, if the system is reliant on that node. A method to redistribute the work done by a node that goes down and connect it to the rest of the system is needed to not lose capacity, but to examine how that works is outside the scope of this thesis.

5.3.4 Moving the IDS

In the implementation of the network the intrusion detection system, Snort, has been placed in the same machine as the switch. However, this is only one of several reasonable locations for the IDS.

Placing the IDS in the switch has the advantage of letting it observe all traffic on the network without having to send data to a dedicated computer, which would increase the load on the network. However, if several switches were to be used Snort would need to run in each switch. This would increase the configuration somewhat and also distribute the logs of the system, which also is a minor inconvenience.

Placing the IDS in the controller is one of the options suggested by the Ryu manual [25]. This solution was considered undesirable as it exposes the computer running Ryu to all traffic on the data plane, which potentially could aid an attacker get access to the controller if an exploit is found in Snort. It does also increase the general load of the network, while not really providing anything positive. It is also possible to let the IDS listen to only the control plane communication, but this would not detect any attacks since Snort is built for normal network traffic. In the future, there might be an IDS aimed at SDNs which would analyze control plane traffic and try to determine if a flow is valid or forged.

Using a dedicated machine to run the IDS is the second option suggested in the Ryu manual and likely the best option if several switches are used. An illustration can be seen in 5.3. Having a dedicated computer with the sole purpose to analyze network traffic would increase the load on the network, but would bring a lot of positive properties to the system. First, it would be possible to adapt the processing power of the hardware to mitigate overloading and have Snort analyze the traffic fast so it can alert the system in time. Second, it would make it easier to configure, since all logic would be in one place instead of spread out in different switches. Third, it creates an extra step to take control of the IDS without exposing the controller to additional risk. The machine should have one interface with the switch in the data plane and one with the controller in the control plane.

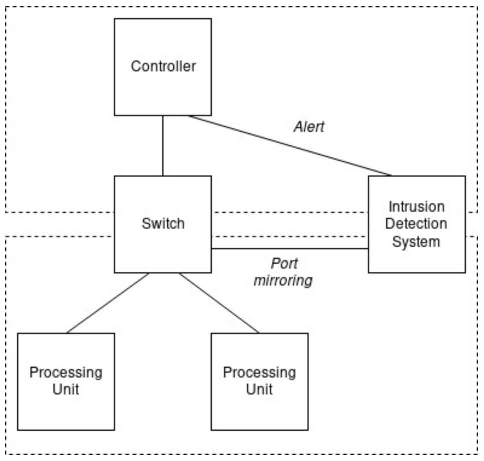

Figure 5.3: Locating the IDS in a dedicated machine

This has the advantage of letting Snort observe all traffic between the nodes without exposing the controller to additional risk.

Chapter 6

Conclusion

6.1 Answers to the Research questions

6.1.1 Network Security

The security of the network does not depend on whether the network is software defined or not. Rather it depends on the components in the network and security policies enforced. A chain is only as strong as its weakest link, which in this case appears to be the possibility for an attacker to redefine the network infrastructure by clever crafting of flows.

SDN seem to be unsuitable for a small safety-critical system. In a small system with limited complexity it should be possible to efficiently enforce network policies using traditional networking methods. SDN could have a bigger role to play in large, complex systems exposed to a lot of change.

Regarding the sensor networks, implementing SDN should not be highly prioritized if the aim is to achieve high security to a comparatively low cost. Money would likely be better spent preventing an attacker from breaching the network in the first place, and creating an IDS which discovers if it is breached. SDN could possibly be further studied in the future if the complexity of the networks increase or the likelihood of a breach is high despite efforts taken to prevent it.

6.1.2 Isolation of Nodes

Although it is possible to isolate nodes in a software defined network, the method demonstrated in this thesis is not sufficient. If an attacker were capable of entering the network they would likely be able to circumvent blocks ordered by the controller. In fact, rather than improving the security, compared to a traditional network, the setup creates a shortcut for blocking all nodes in the network. By using one node to spoof the IP addresses of the other nodes and sending illegal packets, all nodes will be blocked and the network rendered unusable.

With that said, it should be possible to build an SDN capable of blocking nodes based on network interface. Open vSwitch has IDs for its ports and ports could be switched off. The main issue preventing implementation of blocking based on ports is how the controller can know which port should be blocked. Information about incoming port is not mirrored to the IDS.

6.1.3 Galvanic Isolation

The question about galvanic isolation is not directly related to the other two. The network implementation demonstrated that media converters and optic fiber can be used without considerably interfering with the network communications. However, this could depend on the hardware used and should be evaluated, together with the other components, if the requirements of the network are very high.

6.2 Future Work

This section presents ideas for how this work can be continued. Concepts implemented that can be improved and new ideas which can utilize this work as inspiration or a starting point are presented.

6.2.1 Improving the isolation

Looking at another method of isolating nodes is worth investigating. The solution that block nodes on MAC address has a major shortcoming, which is discussed in the discussion chapter. Examining if blocking nodes by shutting down ports in the switch is a practical way, and if there are any direct downsides by doing so. We believe that method provides a much safer and reliable isolation but it should be looked at in more detail.

6.2.2 Investigating an error check for flows

As discussed in ways on how to improve security in a controller, one method is to compare flows from many different controllers before they are added to a switch's flow table. We have not been able to find an implementation of this, and maybe there is a reason for it. If it is investigated in detail and if it is determined a practical solution, then it can be implemented to further improve the security of software defined networks.

6.2.3 Testing attacks on an SDN

We have discussed potential attacks and ways to protect the system from them, we did not have time to actually test them and verify that the system is protected against them in practice. We did read papers which performed attacks on an SDN, but only a few attacks. A review and test of all attack vectors and ways to secure the system against them would be beneficial to verify the theory.

6.2.4 A work distribution algorithm

When isolating a node, the functionality of the whole system is most likely going to be disturbed. As an example, if a program is split into many different nodes with their own responsibilities, then all nodes are needed for the program to work. If a node is isolated because it has been compromised, then the whole program will not function. To solve this, it should be investigated if it is possible to distribute the isolated node's responsibility to another node in the network, to maintain the functionality of the program even if a node is isolated. If this could be solved in a practical way, it would be be easier to build systems which are both more secure and reliable.

References

[1] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE COMMUNICATIONS SURVEYS & TUTORIALS*, vol. 16, no. 3, pp. 1617–1634, 2014.

[2] S. W. Shin and G. Gu, "Attacking software defined networks: A first feasibility study," in *ACM SIGCOMM workshop on Hot topics in software defined networking.* SIGCOMM, 2013, pp. 165–166.

[3] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "Sphinx: Detecting security attacks in software-defined networks." in *NDSS*, vol. 15, 2015, pp. 8–11.

[4] A. Zaalouk, R. Khondoker, R. Marx, and K. M. Bayarou, "Orchsec: An orchestrator-based architecture for enhancing network-security using network monitoring and sdn control functions." in *NOMS*, 2014, pp. 1–9.

[5] ——, "Orchsec demo: Demonstrating the capability of an orchestrator-based architecture for network security," *Academic Demo, Open Networking Summit*, 2014.

[6] R. Khondoker, P. Larbig, D. Senf, K. Bayarou, and N. Gruschka, "Autosecsdndemo: Demonstration of automated end-to-end security in software-defined networks," in *2016 IEEE NetSoft Conference and Workshops (NetSoft).* IEEE, 2016, pp. 347–348.

[7] S. Choudrey and K. Hiltunen, "Moln för försvarsmakten," 2015.

[8] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," https://arxiv.org/pdf/1406.0440.pdf, 2014.

[9] SDxCentral, "What are sdn southbound apis?" https://www.sdxcentral.com/networking/sdn/definitions/southbound-interface-api/, accessed: 2019-04-10.

[10] A. Zhu, "Openflow switch: What is it and how does it work?" http://www.cables-solutions.com/whats-openflow-switch-how-it-works.html/, accessed: 2019-08-06.

[11] M. C. Dacier, S. Dietrich, F. Kargl, and H. König, "Network Attack Detection and Defense (Dagstuhl Seminar 16361)," *Dagstuhl Reports*, vol. 6, no. 9, pp. 1–28, 2017. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2017/6912/

[12] D. Kreutz, F. M. V. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," *IEEE COMMUNICATIONS SURVEYS & TUTORIALS*, vol. 16, no. 3, pp. 1617–1634, 2014.

[13] SDNSecurity.org, "An overview of misuse/attack cases," http://www.sdnsecurity.org/vulnerability/attacks/, accessed: 2019-08-04.

[14] J. Bailey and S. Stuart, "Deploying sdn in the enterprise," *ACM Queue*, vol. 60, no. 1, pp. 45–49, 2017.

[15] Z. Yan and C. Prehofer, "Autonomic trust management for a component-based software system," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 6, pp. 810–823, 2010.

[16] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in distributed systems: Theory and practice," *ACM Transactions on Computer Systems (TOCS)*, vol. 10, no. 4, pp. 265–310, 1992.

[17] Y. Xu and Y. Liu, "Ddos attack detection under sdn context," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications.* IEEE, 2016, pp. 1–9.

[18] J. Bengtsson and H. H. Teodor Sommestad, "It-säkerhetskrav i försvarsmakten," FOI, Tech. Rep., December 2014.

[19] L. Kohnfelder and P. Garg, "The threats to our products," Microsoft Corporation, Tech. Rep., 1999.

[20] 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.

[21] S. Finucane, "Why open vswitch," https://github.com/openvswitch/ovs/blob/master/Documentation/intro/why-ovs.rst/, accessed: 2019-06-09.

[22] S. Lowe, "Layer 3 routing with open vswitch," https://blog.scottlowe.org/2012/10/31/layer-3-routing-with-open-vswitch/, accessed: 2019-09-05.

[23] *SNORT Users Manual 2.9.13*, The Snort Project, 2019, accessed: 2019-07-05.

[24] J. Dries, "An introduction to snort: A lightweight intrusion detection system," http://www.informit.com/articles/article.aspx?p=21778&seqNum=9/, accessed: 2019-07-25.

[25] Ryu, "Snort intergration," https://ryu.readthedocs.io/en/latest/snort_integrate.html/, accessed: 2019-07-08.

[26] G. Thomas, "Incorporating media converters," *Contemporary Controls: the EXTENSION*, vol. 7, no. 6, 2006.

[27] G. P. Agrawal, *Fiber-optic communication systems.* John Wiley & Sons, 2012, vol. 222.

[28] L. Zhu, M. M. Karim, K. Sharif, F. Li, X. Du, and M. Guizani, "Sdn controllers: Benchmarking & performance evaluation," *arXiv preprint arXiv:1902.04491*, 2019.

[29] "Hardware - northbound networks," https://northboundnetworks.com/collections/hardware/, Northbound Networks, accessed: 2019-06-09.

Printed by Tryckeriet i E-huset, Lund 2019

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2019-728
http://www.eit.lth.se