Atte Räty

Security aspects of service chaining in software-defined networking environments

Master's Thesis in Information Technology

December 5, 2018

University of Jyväskylä

Department of Mathematical Information Technology

Author: Atte Räty

Contact information: atte.t.raty@student.jyu.fi

Supervisor: Timo Hämäläinen

Title: Security aspects of service chaining in software-defined networking environments

Työn nimi: Palveluiden ketjuttamisen tietoturvakysymykset ohjelmallisesti hallituissa tietoverkoissa

Project: Master's Thesis

Study line: Information Technology

Page count: 38+0

Abstract: Software-defined networking is a new computer networking paradigm that brings many new possibilies with it. One of these possibilities is the increased dynamicity of chaining together network services (also known as network service chaining). This way network operators can shape the path the network traffic has to take to reach its destination. This also means that the network operators can easily manipulate the traffic using various services. While this will certainly be useful for various network operators and service providers it may also open up new vectors of attack for malicious actors or change how old attack vectors are utilised. In this paper I will explore the concept of SDN-based service chaining and the security concerns that it may cause from the perspective of a web hosting provider.

Keywords: SDN, software-defined networking, NFV, network function virtualisation, security, service chaining, networking

Suomenkielinen tiivistelmä: Ohjelmistojohtoinen tietoverkkojen hallinta on uusi lähestymistapa tietoverkkojen hallintaan ja se tuo mukanaan uusia mahdollisuuksia. Eräitä näistä mahdollisuuksista ovat monipuolisemmat ja dynaamisemmat mahdollisuudet verkkopalvelujen ketjuttamisen. Käyttäen palvejulen ketjuttamista, tietoverkon hallinnoijat voivat kontrolloida reittiä, jota verkkoliikenteen on käytettävä päästäkseen haluamaansa kohteeseen. Ja koska tietoverkon hallinoija voi kontrolloida tätä reittiä, hän voi myös kontrolloida mitä verkkolii-

kenteelle tehdään tällä reitillä. Tämä ei kuitenkaan tuo mukanaan pelkkiä hyötyjä, vaan tämä voi avata myös uusia väyliä ja hyökkäysmuotoja vihamielisille tahoille. Tulen tässä tutkimuksessa käsittelemaan ohjelmistojohtoisen tietoverkkojen hallinnan mahdollistamaan palvelujen ketjuttamiseen liittyviä tietoturvakysymyksiä verkkopalveluja ylläpitävän toimijan näkökulmasta.

Avainsanat: SDN, ohjelmistojohtoiset titetoverkot, NFV, tietoverkkopalveluiden virtualisointi, tietoturva, palvelujen ketjutus, tietoverkot

Contents

1 Introduction

The modern world is filled with different kinds of computing devices ranging from smartphones to data centers full of computers acting as servers. The use of these computing devices has become an everyday activity to a great deal of people and a critical part of the way companies do business. Much of the functionality of these computing devices is dependent on network connectivity. This means that network infrastructure is very important (or even critical) to the functioning of these computer systems and, by extension, the modern society. Yet the management of traditional networks is still often cumbersome and resource-intensive (Kreutz et al. 2015; Chiosi et al. 2012). Software-defined networking (SDN) is an approach to computer networking that seeks to make the management of computer networks simpler, centralised and software-driven.

The basic principle of software-defined networking is the separation of the data and control planes of network traffic. Control plane refers to the management of how network traffic is routed whereas data plane refers to the actual routing of the network traffic. A central idea in software-defined networking is the centralisation of the control plane's responsibilities (network management and control) to an SDN controller. The SDN controller is a piece of software that is used to manage all the networking equipment (whether virtual or hardware) centrally (Kreutz et al. 2015).

One of the main features of an SDN controller is the abstraction of network configuration. Working with the abstractions provided by the controller, network configuration changes become a lot simpler. The abstractions also serve to allow the controller to expose programmable interfaces (application programming interfaces, APIs) for use by different systems and users (Kreutz et al. 2015). It is easy to see how the simplified network configuration and programmability makes many network management concepts and ideas easier and more feasible to implement and try out.

One of these concepts is service chaining. Service chaining is the concept of chaining network services together in such a manner that network traffic will follow a specified route through various services to gain access to the services ultimately requested by the client.

The chained services can consist of, for example, firewalls, load balancers, caching servers and web servers (Blendin et al. 2014b). Many of the functions of the chained services are typically performed by hardware middleboxes in traditional networks. With the help of SDN, the process of chaining these services together can be made easier. A network service provider can, for example, have a setup where the quality of streamed video can be changed on-the-fly and even during playback of the said video without any interaction by the end user (Blendin et al. 2014a).

In chapter one I will introduce the topic of my thesis and my research question. In chapter two I will delve into network function virtualisation, what it means and some analysis of it. Chapter three is dedicated to introducing the reader to the concept of SDN and analysis of its benefits and challenges. Chapter four will focus on the concept of service chaining. It should give the reader a clear picture of what service chaining is and some analysis of it. Chapter five is where most of the original work is done. It is dedicated to theoretical security analysis of network function virtualisation, service chaining and SDN environments. On top of analysing all these concepts on their own the security of a system consisting of all three is also analysed. Chapter six introduces an experiment plan that can be used as a basis for future work to perform practical security testing and analysis. Chapter seven is used to present the conclusions of my thesis work.

1.1 Research question

New technologies and approaches typically raise new kinds of problems and concerns. In this paper, I will look into the security aspects of service chaining in SDN environments. I will try to find both the security benefits and vulnerabilities that the approach may bring and analyse them. My research question is: “What kinds of security benefits and vulnerabilities result from service chaining in software-defined networking environments?”

To try to answer this question I will look into relevant literature and compare the security aspects of this setup to those of more traditional network setups. I will also plan the creation of an experimental setup consisting of a firewall, an intrusion detection system (IDS), a caching server and a web server that is meant to model a possible setup used by a web

hosting provider. I hope this setup will help me and others explore the security aspects of service chaining in SDN environments.

On top of planning the creation of an experimental setup I will also plan some security experiments to be run against this setup. However performing this security testing will be left as future work.

The results of my findings can be used by network operators and service providers to evaluate the risks and benefits associated with service chaining in software-defined networking environments. The results of performing the experiments can also be used to identify weak spots in existing network setups and as an indicator of what might be worth further research.

2 Network function virtualisation

Network function virtualisation (NFV) is the concept of replacing networking hardware (providing networking functionality) with virtualised equipment performing the same functionality. The virtualised equipment typically consists of virtual machines. This is desirable because network hardware is often specialised for certain tasks. With virtualisation it can be replaced with the combination of software and general-purpose hardware resulting in a setup that is typically cheaper and more flexible (Chiosi et al. 2012; Blendin et al. 2014b). In practice network function virtualisation can mean, for example, replacing dedicated hardware switches with computers running virtual switches on top of the common x86 processor architecture (Han et al. 2015; Chiosi et al. 2012).

2.1 The benefits of network function virtualisation

NFV has many benefits over traditional specialised networking hardware. One of the most significant advantages it offers is economical, as already mentioned. Dedicated networking hardware is expensive and resource intensive to procure, deploy and maintain (Chiosi et al. 2012). With NFV all of these dedicated hardware middleboxes can be replaced with traditional commodity server hardware. While one likely needs more commodity hardware to reach the performance of dedicated hardware, commodity hardware is also cheaper and more readily available because the market for it is larger and more competitive.

Networks created using dedicated networking hardware are typically cumbersome to modify or change (Chiosi et al. 2012). Making the backbone of networking software-based allows for better flexibility in the management of the network. This is especially true if the software is running on virtual machines as one can easily fine-tune the configuration (the virtualised hardware) the software is running on. The increased flexibility allows one to better keep up with new innovations and trends and include new technologies into the network. The increased flexibility also makes it easier to have complicated and detailed network setups and topologies.

The combination of the flexibility of the network setup and the hardware running it results

in a good scalability of network capacity. When it is easy to both setup and procure new hardware it makes scaling up to the growing demands of the network easy. When the management and configuration of the network is easy, it is also easy to scale the network down. All of this can even be as simple as configuring the amount of resources the virtual machines have available to them or how many virtual machines are running so you may not even need to get new hardware.

Dedicated hardware also generally requires specific skill sets to setup, configure and run. With NFV the skills required will shift to more typical system administrator and virtual machine administration skills which are more readily available (Chiosi et al. 2012). This also means it is easier to learn to manage the networking equipment as one does not need to learn multiple vendor specific configuration schemas and practices.

2.2 The challenges of network function virtualisation

The most obvious challenge with network function virtualisation is performance. It does not seem plausible for software-based and virtualised solution to beat dedicated hardware at what the hardware is designed to do. The catch is that NFV solutions do not have to beat dedicated hardware's performance. They just need to get close enough so that the value proposition of NFV still comes out ahead in the comparison. This can be through simple monetary value (as commodity server hardware is typically cheaper than dedicated networking hardware even if you have to get more of it) or through the other added benefits of NFV solutions.

Network function virtualisation also faces the challenge of interoperability (Chiosi et al. 2012). The NFV solutions need to be able to run on hardware provided by many different vendors and on many different virtualisation solutions (namely, hypervisors). The support for these platforms needs involvement from the vendors that are competing with each other and trying to provide a case for why their solution is favorable to the competition. NFV solutions also have to be run together with the already existing network setups that rely on dedicated hardware as moving to a completely NFV-based solution at once is often impractical. This requirement is especially tricky since some of these existing setups are legacy systems performing critical functions in critical systems.

Then there is also the problem of security. It is reasonable to assume that transferring the functionality from hardware to software increases the attack surface for attackers. Obviously the current hardware runs software to perform its duties but moving to purely software-based solution means adding at least one additional layer of software being used and more software and complexity commonly results in more potential security vulnerabilities. For example, the technogolies used for virtualisation (hypervisors) are known to have faced security issues in the past (Corporation 2015).

Stability is also going to be one of the major hurdles for NFV since the networking backbone absolutely needs to be stable in the modern world. Virtualising the hardware results in extra complexity and thus more potential for stability affecting failures. With dedicated networking equipment the stability mostly depends on the reliablity of the hardware, the stability of the operating system the hardware is running and the stability of the software controlling the networking functionality. With NFV the reliability burden of the hardware is delegated from the dedicated hardware to commodity hardware and on top of this the host operating system (the one running the virtualisation software) and the virtualisation solution. That means that two more complex software systems are added to the mix so it stands to reason that the stability of the whole solution will be affected.

3 Software-defined networking

3.1 How software-defined networking works

The operation of software-defined networking can be roughly divided in three parts. These three parts are handling of the data plane, handling of the control plane and the communication between the SDN controller and the networking devices.

The control plane is handled solely by the SDN controller. It provides all the intelligence of the network and acts as the primary decision maker for the network. It also provides the view of the network to controllers and administrators of the network and enables them to control the network. In essence, it is the brains of the network. One could liken this relationship to how a CEO (SDN controller) of a publicly traded company acts according to the guidelines and requests of the company's stakeholders (network administrators) but does not bother them with the unnecessary details of operating the company (configuring networking solutions).

The handling of the data plane is left to the networking devices (whether hardware or virtual). They handle transferring the network traffic according to the instructions of the SDN controller. If they are not sure about how they should proceed (for example, how to handle a new type of traffic) they consult the SDN controller. The handling of control plane in an SDN environment is not all that different from how it is done in traditional networks.

The communication between the SDN controller and the networking devices happens using a networking protocol. This communication consists of instructions and queries for instructions. Currrently the networking protocol of choice for software-defined networking is commonly OpenFlow though SDN is in no way tied to any specific protocol.

3.2 The structure of an SDN controller

The SDN controller, too, can be divided into three distinctive parts. These parts are the northbound APIs, the southbound APIs and the core controller. A user interface for the

controller could coinceivably be called a fourth part but it is built on top of the northbound APIs and does not really affect how the SDN controller functions so it is not treated as a disctinct part of the controller in this paper.

The northbound APIs refer to the interfaces the SDN controller exposes to the outside world. It is these APIs that enable the programmability (and all the related benefits, such as extensiblity) of the SDN controller. As such, the northbound APIs are a huge enabler for the SDN controller. These APIs are used to control the operating of the network and can be used to, for example, make configuration changes.

The southbound APIs refer to the interfaces that enable the SDN controller to connect to a multitude of different networking devices and services and to create the abstractions that make it easier to reason about and operate with these devices and services. In other words, the southbound APIs are the things that enable the SDN controller (and by extension, the concept of software-defined networking) to function. Southbound APIs are the part of an SDN controller that handles communicating with the networking devices and services using the networking protocol of choice.

The core controller refers to the core functionality of the SDN controller, in other words "everything between the northbound and southbound APIs". This is the part where all the functionality associated with an SDN controller resides. The core controller is what makes all the control plane decisions. The northbound and southbound APIs are merely the ways of communicating with the core controller.

The following example will demonstrate how the different parts of the SDN controller interact with each other. A user makes a change in the routing configuration using a web-based user interface. This invokes an action in the northbound API which in turn causes a change in the core controller. This change in the controller gets passed to the southbound APIs that communicate the change to the networking devices using OpenFlow.

3.3 OpenFlow

OpenFlow is the de facto standard networking protocol used with software-defined networking. It is mainly utilised in the communication between the SDN controller and the networking devices and services. OpenFlow was originally intended to be a way for researchers to experiment with new protocols in and along side of existing networks using existing hardware (switches) (McKeown et al. 2008).

OpenFlow is based on an Ethernet switch that provides a flow-table, for "flows of network traffic", and a standardised interface to modify this flow-table. (McKeown et al. 2008). Most modern Ethernet switches, even at the time of writing of the original article (2008), already contained these flow-tables and provided functionalities for operating with these flow-tables that shared similarities between different vendors and implementations (McKeown et al. 2008). These factors combined made it an obvious platform for the authors of OpenFlow to build OpenFlow on top of.

When using flow-tables to handle network traffic, the traffic is separated into different flows based on its properties. Each flow can then be configured to perform specific operations on the traffic. In practice this can mean, for example, that traffic coming from different networks can be routed differently or that traffic coming from specific networks can be handled with differing levels of trust. In other words, flow-tables are a way of enhancing the basic functionality of networking devices.

Flow-tables are also used to enable the experimentation on already existing networking hardware without disrupting its normal operations by using the flow-table and its associated interfaces to divide the network traffic to isolated production and research flows (McKeown et al. 2008). This allows researchers and network operators to freely experiment with new networking mechanisms without having to deploy specific devices or building seprate networks for experimentation.

3.4 The benefits of software-defined networking

The most obvious benefit provided by software-defined networking is also its core principle, the separation of the control and data planes. This makes handling and reasoning about network setup and configuration easier as one can think of the two as separate traffic flows.

The hardware abstractions provided by an SDN controller and protocols make configuring the network a lot easier. With the abstractions networking devices can be classified into certain classes of devices that can all be controlled at once using the same commands, regardless of the vendor or operating system of the device. Thus the network becomes vendor and operating system independent.

One of the greatest benefits of software-defined networking is the centralisation of network management to an SDN controller. This is a great benefit because it makes network configuration changes easy and quick. Instead of having to configure a multitude of devices, one can just configure a rule in the SDN controller that then gets propagated to all the relevant devices and services.

With the aforementioned ease of configuration trying out new things becomes a lot easier and simpler to do. This way new technologies can be integrated as part of the network or the network setup can be made more complex. One can even run isolated production and testing networks side by side thanks to the ease of configuration.

Not only does not the centralisation of the network management to the controller ease the network configuration, it also means that the controller has visiblity over the whole network. This can be very useful from the point of view of a network administrator.

The combination of network management being centralised to an SDN controller and this controller exposing APIs for traffic management means that the network becomes programmable (moreso than it was before). This makes developing new networking services quite a bit easier and faster to do. There have even been programming languages, such as Frenetic, developed for the specific purpose of programming the network using SDN controllers (Foster et al. 2013).

3.5 The challenges of software-defined networking

Centralisation of the control of the network to an SDN controller can be both beneficent and a negative effect. With the centralisation the SDN controller becomes the network's single point of failure. In case something goes wrong and the SDN controller goes down or starts malfunctioning, it will affect the entire network. Granted, this can be partly worked around using a secondary SDN as a failover or in parallel to the primary one but this brings more configuration complexity to the equation.

The controller being a single point of failure for the network is also a problem securitywise. If an attacker gets access to the SDN controller he basically gets access to the whole network. For this reason, SDN controller security is very important.

The concept of SDN relies on every device cooperating. If there are individual devices that don't support the technologies required for an SDN environment (for example, OpenFlow protocol) the SDN controller loses its ability to manage the whole network. While this might be acceptable with individual devices it can easily counteract and defeat the purpose of the whole SDN environment if the number of problematic devices rises too high. Luckily for SDN, it seems like most networking hardware vendors are committed to supporting SDN environments and many of the already existing devices can support OpenFlow with a simple firmware upgrade.

While SDN lessens the need for deep technical knowledge and understanding it does not completely do away with it. If something goes wrong and the network or SDN controller needs to be debugged or troubleshot one still needs deep technical understanding. Besides, the SDN environment adds a new skill set that will be required to operate an SDN-based network. Doubly so if one wants to take full advantage of the SDN APIs as that means one has to have both programming and networking know-how.

Because every new situation/occurrence needs to be handled by the SDN controller, some performance overheard is to be expected. This is especially true if the SDN controller is busy and thus can't immediately process all the requests it receives. This also means that the performance of the SDN controller is vital to the performance of the overall SDN network.

Similarly to the possible problem of performance, scalability might also prove problematic with SDN architecture. One SDN controller can only serve so many clients (mainly networking devices) before being overwhelmed by the load so either the (possibly virtualised) hardware the SDN controller is running on needs to be upgraded or new controllers need to be introduced and, as already mentioned, this increases the complexity of the network configuration by a non-trivial amount and may introduce new kinds of problems (for example, what to do when a device is simultaneously being controlled by two SDN controllers).

3.6 The relationship between network function virtualisation and software-defined networking

Network function virtualisation and software-defined networking are two different concepts. They both have to do with computer networking and are complimentary to each other (and thus often mentioned and used together) but they operate at different conceptual levels of computer networking. NFV is something that appears at the hardware level and deals with the networking hardware whereas SDN is more of a network management concept. An example of a setup combining NFV and SDN is an SDN environment where all the networking equipment (routers, switches, ...) are software-based virtualised solutions (using, for example, Open vSwitch) but are being controlled by an SDN controller using the OpenFlow protocol.

The use of NFV can ease the creation, maintenance and management of SDN environments. These kinds of things help to streamline the part of networking that SDN does not concern itself with, the networking devices themselves. SDN, on the other hand, takes care of most everything besides the devices itself. It is easy to see how these two concepts compliment each other.

It is also important to note that even though network function virtualisation and software-defined networking are complimentary concepts both of them can be used completely separately from each other. It doesn't matter to an SDN controller whether the networking hardware is really dedicated networking hardware or virtualised hardware and similarly virtualised hardware doesn't care how it is being managed in the sense of network administra-

tion.

4 Service chaining

Service chaining is the concept of organising network traffic into service chains. These service chains are chains of multiple network services built to provide a specific service (Medhat et al. 2017; Lin et al. 2016). This means that the administrator of the network can easily control what operations are performed on the traffic prior to it reaching a specific service.

Service chaining on its own is not a new concept as network services have been chained together for a long time (simple example of which is traffic coming from router or switch to a firewall which either blocks it or lets it through to a web server). These traditional service chaining setups are often based on traditional middleboxes and are rather static and built with one specific network in mind (John et al. 2013; Leivadeas et al. 2016). All of these factors make traditional service chaining setups inflexible. With the introduction of new networking technologies and paradigms such as SDN and NFV service chaining can be built to be more dynamic. These new advancements result in increased flexibity, scalability, granularity and configurability. The service chaining discussion in this paper will be focused on this new and more dynamic "next generation" service chaining.

4.1 The benefits of service chaining

Dynamic service chaining enables network operators to swiftly adjust to changes in the functioning of the network because of the ease of making changes. For example, if a caching server suddenly starts malfunctioning the administrators can direct the traffic right to the web server (ignoring caching) or to an alternate caching server or even spin up a completely new instance of the caching server and direct the traffic there.

Being able to dynamically adjust your service chains can also help cope with sudden changes or surges in network traffic. For example, under sudden increase of network traffic load, the incoming traffic can be split and directed to different service chains to more evenly distribute the traffic. Changes like this one can also be very granular. For example, a banking web application under heavy load can direct all the domestic traffic to a separate service chain while

all of the foreign traffic is handled by different service chains. This means that the banking service functions as normal to the domestic customers (the prioritised customers in this example) while still providing some level of functionality for foreign customers whereas in a more static network setup the service's functioning would have been compromised for everyone or completely unavailable for foreign customers (if the bank decided to drop foreign network traffic).

Scalability can be managed by directing traffic to different service chains as needed. These different service chains can offer varying performance and capacity, ignore some optional services or even be hosted in a physically different location (for example, extra on-demand processing power in a cloud environment).

Service chains enable easy experimentation with network traffic and services. The network operators can adjust how much traffic a service gets, how the traffic gets to the service or even where the service is hosted. A network operator can, for example, put two competing products in otherwise identical service chains and split the incoming network traffic between these two service chains to compare these competing solutions. Or he can include certain services in one chain but leave them out of other chain to evaluate the effectiveness of certain services (for example, caching or load balancing).

Service chains make the integration of new services to the collection of services easy. These new services can, at first, be exposed to reduced amounts of network traffic in production environments. This network traffic load can then slowly be scaled up while the health, stability and functioning of the service is monitored. This allows the controlled introduction of new services to the network. These new services can also be included in service chains that only receive part of the incoming traffic. This can be used to not only evaluate the performance and stability of the new service but to also perform A/B testing to gauge the users' impressions of the change. A/B testing refers to the practice of randomly referring a portion of the users to a new version of a service or an application in order to evaluate the users' impressions of the change (Hynninen and Kauppinen 2014).

4.2 How services are chained together

4.3 Software-defined networking and service chaining

Software-defined networking is an enabling technology for (dynamic) service chaining. Thanks to the ease of configuring networking devices and the programmability provided by SDN APIs the creation, configuration and management of service chains is made easy, efficient and practical.

The ease of configuration of networking devices and services is crucial for dynamic service chaining. If creating and managing the service chains would be hard and cumbersome dynamic service chaining would not be practical and thus (probably) not achieve significant use.

Service chaining can also be seen as a beneficiary techology to software-defined networking as it is a clear example of a technology that SDN enables and enhances.

5 Security aspects of service chaining in NFV-enabled SDN environments

5.1 Evaluating security

To be able to reason about a system's or architecure's security effectively one needs to understand the kinds of threats it is facing. One can easily come up with potential security problems of a system but to properly evaluate the security one needs to also understand how feasible the attacks are to pull off. It is very difficult (if not completely impossible) to build a completely secure system so eventually one has to settle for "good enough" instead of piling on multitudes of "what if" scenarios. A prime example of this is modern cryptography. Whenever a cryptographic method relies on some kind of a seed or a private key it is possible to crack the encryption. But if cracking the encryption (on average) takes a time counted in dozens of years or more it is not reasonable to prepare for that attack.

To be able to evaluate the security of a system one needs to first understand the threats related to the individual parts of the system. And as a system is often desired to be greater than the sum of its parts so can the security implications extend from beyond those presented by the individual parts. Some new security implications may arise from how the parts are connected to each other or just from the mere fact that they are connected to each other.

All of this means that in order to evaluate the security of service chaining in SDN environments utilising network function virtualisation one needs to first evaluate the security of SDN networks, network function virtualisation and service chaining. Then one needs to evaluate the security implications of combining these concepts and systems with each other. In this paper, this will happen on a more general and conceptual level instead of focusing on specific implementations of these concepts.

5.2 Threats for SDN controllers

The SDN controller contains all of the network's intelligence and it is also in control of the network. This makes it an extremely appealing target for a possible adversary.

The first and most obvious threat faced by an SDN controller is the take over of the system it is running on. There are multitude of ways of achieving this. This kind of a compromise can lead to the total compromise of the SDN controller and thus the network. This will not be explored in further detail as this is not specific to software-defined networking (and this is a separate area of research that has been and continues to be active).

Threats focusing on the SDN controller itself (as opposed to the system it is running on) can be roughly divided into two categories: threats utilising the northbound APIs and threats utilising the soutbound APIs. That is to say, the threats can either utilise the APIs the SDN controller exposes to the outside world or the APIs that it uses to communicate with the networking devices.

5.2.1 Southbound APIs

Southbound APIs, like most APIs, are suspectible to denial of service (DoS) attacks. Denial of Service attacks are attacks whose purpose is to deny some service from legitimate users by using up all of resource the server or servers hosting the service. There resources can be anything from network bandwitdh to CPU time to harddrive space. If an attacker can somehow get the networking devices to perform resource intensive operations (in this context, that could for example be complex routing calculations) he could hog up all of the computer power of the SDN controller. Realistically, moden hardware is good enough to handle most anything that could be directed to SDN controller by a single networking device without much trouble. That is unless there is a flaw in the SDN controller implementation or design that allows the attacker multiply the power of the single resource intensive operation.

While an SDN controller can easily handle resource intensive operations from one networking device at a time it may fail against several (or even all of the) networking devices doing it simultaneously in something known as a distributed denial of service (DDoS) attack. DDoS attacks are much like normal DoS attacks with one major twist: they're performed by mul-

tiple devices at the same time. These are much trickier to defend against (as compared to normal DoS attacks) as they can be so powerful that it is simply too expensive to handle them through raw computing power.

SDN controller handles input from the networking devices and this input can be at least somewhat controlled by an attacker and this opens the SDN controller up to an injection attack. Injection attacks are attacks where the attacker gets the system (or some other system) to execute arbitrary code or commands through the use of malicious input (such as inputting specific SQL statements to a search field for an SQL injection). In SDN controller's case, this malicious input can be specific requests from the networking devices (such as the SDN controller being unable to handle non-standard IP addresses or malformed requests securely). These requests can result from either the attacker directing such malicious requests to the networking device which then forwards them to the SDN controller or the attacker taking over the networking device and creating such malicious requests. Injection attacks are a major part in why the SDN controller should not assume that the calls to the southbound APIs come from trusted sources.

Another threat posed by malicious input from the networking devices is that of malicious flow or instructions. If an attacker can get the SDN controller to create a flow that, for example, redirects all traffic from certain IP address ranges to a network of his choice, it opens the users of the network up for new attacks and threats. Such a routing rule could be created as a result of improper validation or authentication of the requests coming from the networking devices.

5.2.2 Northbound APIs

The threats faced by the nortbound APIs are similar to those faced by southbound APIs. The biggest difference is that the attacker has the potential to interact with the API more directly instead of having to work through networking devices. Northbound APIs are also easier to restrict and secure because their functionality is more of a convenience than necessity for the network. Southbound APIs are also limited by the capabilities of the networking devices in use while northbound APIs can be built a lot more freely. For example, it is rather hard to

change or update the authentication mechanism of API calls with southbound APIs since that would require changes to how the networking devices operate and while the same changes in northbound APIs would also require changes to the API clients it is safe to assume that the clients of the northbound APIs are generally easier to modify (for example, slight changes to an API client library compared to networking device firmware update).

5.3 Networking devices in SDN network

Networking devices in an SDN environment are mostly similar to normal networking devices with a major difference being that they are configured to support and use the SDN-enabling networking protocol of choice (for the purposes of this paper, the protocol of choice is presumed to be OpenFlow). Therefore, the threats they face are mostly similar to the threats faced by normal networking devices.

For the purposes of this paper, threats related to networking devices are roughly divided into three categories: threats that aim to prevent the functioning of the networking device (or part of its functionality), threats that aim to modify the funtionalities of the device and threats that aim to bypass or fool the device. It is important to recognise that these categories overlap with each other.

Networking device can be prevented from properly functionting, apart from physically disabling the device, with a denial of service attack. If the device is unable to provide the desired functionality to the clients using it, it is effectively disabled. Another way of achieving the same result is by cutting the device off the network so it can not communicate with its clients and other devices.

If an attacker can break ("hack") into the device itself or the SDN controller controlling it he can modify the device to function in malicious ways. These ways can be clearly malicous, such as redirecting all the traffic to a network controlled by the attacker, or they can be more subtle. When a device is functioning in subtly malicious ways it can appear to operate normally (at least on quick inspection) but might in reality be doing something on top of its normal functionality (such as mirroring/logging all traffic to another device controlled by the attacker) or misbehaving in subtle ways (for example, filling networking packets sent to a

specific IP address with malicious or incorrect data).

Being able to completely bypass or fool a security-focused networking device to let the traffic through can be extremely valuable for an attacker. If a firewall, for example, is not handling the traffic in an appropriate way it makes performing attacks against other networking devices easier as you can pass them any kind of traffic you want. Of course, the effects of this should be limited if the other devices are properly configured but this is not always the case.

5.4 Threats for network function virtualisation

The threats faced by virtualised networking devices are largely the same as those faced by traditional hardware networking devices. On top of that virtualised devices are subject to threats resulting in weaknesses and vulnerabilities in the virtualisation solution and in the underlying hardware running the virtualisation solution so they have a slightly increased attack surface compared to dedicated hardware solutions.

5.5 Threats for service chaining

Service chaining does not pose threats that are different from the threats faced by the individual services. The exception to this statement are weaknesses and vulnerabilities in the service chaining solution or in the way service chaining is implemented and handled.

A threat that is not specific to service chaining but is relevant in the context is how the different services are interfacing with each other and third parties. If this is not handled securely it can open the services and the whole system up for an attack.

Service chaining may also result in an increased level of trust between the services being chained but this is more of a user error than a threat posed by the technology or concept. If the services treat each other as trusted and don't perform proper validation, sanitation and verification on the messages and the data being passed between the services this may result in multiple of the services getting breached as a result of a breach in one of the services.

5.6 Threats for the combination of SDN, NFV and service chaining

The combination of service chaining, software-defined networking and network function virtualisation does not pose any additional threats than those posed by each of these individually. They are separate concepts and implementations that are not really visible to each other as the combination of their functionality can also be provided without any of them. The main value provided by their combination is to be found in convenience, effectiveness, ease of use and centralised control and management.

6 Proposed experiment

6.1 Note on experiment

The experiment was originally planned to be performed for this thesis. However, due to unexpected personal issues of the author the experiment itself will not be performed. Instead of the experiment being performed a rough test plan is laid out in this chapter. Performing the planned experiment is future work.

Parts of the experiment were performed as the author was a part of setting up OpenStack setup on Linux servers.

6.2 Setup

The setup to be used in the experiment is meant to resemble a setup that could be used to host a web site. The setup includes a web server, a caching server, a firewall, an intrusion detection system (IDS) and an SDN controller. For the sake of simplicity, the setup will not replicate some of the more advanced and complex parts of real-world setups (such as load balancers).

6.2.1 Firewall

Firewall in networking contexts is a tool that is used to filter network traffic. It can be software-based (such as iptables found in most GNU/Linux distributions) or a dedicated hardware device running specialised software (such as commercial enterprise firewalls or pfSense). Firewalls work by filtering network traffic based on its properties (such as destination and source IP and MAC addresses, port numbers or network protocols). Firewalls are used extremely commonly and some typical use cases are restricting access to and from a network (for example, denying any connections to and from public networks).

There are many different types of firewalls based on the way they operate. They can be either stateful or stateless. Stateless firewall refers to a firewall that doesn't hold any history

or information of previous network sessions and as such every network packet is handled as if it was an individual packet with no connections to previous network sessions. (Yang, Lee, and Ko 2008; Gouda and Liu 2005). Stateful firewall, on the other hand, refers to a firewall that decides the fate of a network packet based on what was done to previously network packets from the same source (IP address or domain, for example)s and thus holds information on the state of network connections that have attempted to pass through the firewall before (Gouda and Liu 2005). A common example of a rule in stateful firewall is that no incoming packets from a foreign network are permitted into the private network unless they are related to a connection initialised by something inside the private network that the firewall is protecting.

In this setup, pfSense will be used. pfSense is an open source firewall & router software that is to be installed on dedicated device (which can be virtualised). It is based on FreeBSD operating system of the BSD family of operating systems. pfSense is a stateful firewall. In the setup, pfSense will be the default gateway connecting the private network to public internet.

6.2.2 Intrusion detection system

Intrusion detection system (IDS) is a system used to monitor a network and detect network intrusions (unauthorised access to the network). They work by monitoring the network traffic and creating models of what the traffic for the particular network is normally like. If the IDS then notices something out of the ordinary it will get looked into and possibly flagged as an intrusion.

Intrusion detection systems rely on a big body of data depicting the normal behavior of the network. Combining this data with various statistical analysis techniques (such as anomaly detection) they are able to spot possible abnormalities in network traffic.

Snort is the IDS of choice for this experimental setup. Snort is open source software.

6.2.3 Caching server

Caching servers are used to boost the performance of various applications. They work by storing frequently requested data to a fast storage (referred to as the cache) from where it can be retrieved faster than from typical storage solutions. Caching servers are located between the user of the application and the application itself and the traffic to the application is directed through the cache server so if the requested data is cached it can be immediately returned without the application needing to perform any operations.

Squid is the caching software used for the caching server in this experiment. It is an open source caching and forwarding HTTP proxy. It will be used to cache static HTML (Hyper-Text Markup Language) sites (as that is what the web server will be hosting).

6.2.4 Web server

A web server is a HTTP (and HTTPS) server serving web pages for its clients (typically web browsers). A web server's job is to handle the serving of the web pages and everything related to it. This means that it has to listen for and react to HTTP traffic and respond with proper HTTP responses. Of course, it is not quite that simple and a web server has to concern itself with things like encryption (HTTPS), access rights, redirections and handling huge amounts of simultaneous users. Additional information about the functioning of web servers is provided as needed.

The web server that is used for the experiment is called NGINX. NGINX is an open source web server that can also additionally function as an HTTP cache, a load balancer or a reverse proxy.

In this setup the web server is only used to serve simple static HTML content.

6.2.5 SDN controller

OpenDaylight is an open source SDN controller chosen for the experimental setup.

6.2.6 OpenStack

OpenStack is an open source tool created for hosting cloud computing solutions (infrastructure-as-a-service). It allows and eases the creation and management of multiple virtual machines running on a single physical machine. It is used, much like SDNs, to abstract away the underlying hardware and implementation details and thus create vendor agnostic environments.

OpenStack is used for the creation and management of the experimental environment on a single physical machine. Virtual servers are created using OpenStack as needed. All of the tools listed above will be hosted as their own virtual servers.

6.3 Related tools

6.3.1 Kali Linux

Kali Linux is an open source security-focused GNU/Linux distribution based on Debian. It is built for digital forensics and penetration testing. This means that it comes with many tools for attacking different kinds of computer systems. These are the kinds of tools that are used by both "white hat" security researchers and "black hat" attackers in the real world. This makes it an useful tool for evaluating the security of computer systems.

Kali Linux is used to simulate various attacks against the experimental environment and to evaluate the security of it.

6.4 Testing methodology

The testing that will be performed is security testing of networking between SDN controller and different networking devices.

Security testing of the SDN controller itself is a whole topic of its own and is not within the scope of this thesis. Similarly the security testing of different networking devices and in-depth security testing of different networking protocols are not within the scope either.

6.5 Different attacks to be performed

6.5.1 ARP cache poisoning

Address resolution protocol is a networking protocol typically used to map hardware level MAC addresses to IP addresses. ARP is used when there are no intermittent routers or gateways between two networking devices. ARP is required in such networks because on top of knowing the IP address of other devices the MAC addresses also need to be known due to link layer protocol resources (for example, Ethernet in a case of Ethenet cables).

When a networking device needs to transmit traffic to a previously unknown IP address it broadcasts an ARP message to the network requesting the device with the IP address to respond. Upon receiving a response from a device it will map its MAC address to its IP address. Now it can send traffic to that device.

One important functionality of ARP is an ARP cache. This cache is used to cache the IP-MAC address pairs so that everytime a device needs to send a packet to another device it does not have to broadcast for the IP and wait for the response. This is used both to improve performance and to avoid unnecessary traffic.

ARP cache poisoning is an attack where the ARP cache of a device is poisoned with a fake ARP entry by modifying an IP-MAC address pairing in ARP cache. This will cause the traffic of the poisoned entry to be misrouted to a different device that can be attacker controlled. This can be used to eavesdrop, perform a man-in-the-middle attack (6.5.2) and to send the traffic to an attacked controlled device.

6.5.2 Man-in-the-Middle attack

A man-in-the-middle attack (MitM) refers to an attack where the attacker manages to gain access to network communications between two or more communication points. In practice this means that the attacker can listen to, inspect and in some cases even modify the network traffic between two networking endpoints.

A practical example of this is that an attacker can modify an HTML page shown to an user to contain malicious JavaScript that wasn't contained in the original HTML document sent

by a web server.

Another practical example is the attacker listening for and capturing authentication related information such as tokens and keys. The latter example renders the authentication useless as the attacker can pretend to be a legitimate authenticated user and do everything the user is authorised to while appearing as the user in question.

6.5.3 DHCP spoofing

Dynamic Host Configuration Protocol (DHCP) is a protocol used to dynamically assign network devices IP addresses and some other network configuration related values. In most common scenarios there is a single master DHCP server (though having multiple is also possible) and introducing additional DHCP servers can cause serious problems to the network. In an SDN environment it makes sense for SDN controller to act as this DHCP master server.

Obviously some client devices and the controller itself can (and probably) will have static IP adresses but it is also quite possible that some of them might have dynamic IPs.

DHCP spoofing refers to the act of introducing a rogue DHCP server to the network. This will cause problems because DHCP clients typically use automatic discovery for the DHCP server they will use (instead of a specifically configured one). In such a scenario there will be multiple DHCP servers that can and most likely will assign conflicting IP addresses. This may result in two completely separate network clients having the same IP address. Such a scenario will make the correctly routing of traffic based on IP address very troublesome and thus cause major disruptions in the networking devices using DHCP protocol.

This kind of an attack can be simply tested by introducing a second DHCP server to a network already using DHCP.

6.5.4 Malicious SDN client device

Can an SDN client device controlled by malicious actor disrupt and endanger the functioning of other SDN controller connected client devices by sending maliciously modified and crafted OpenFlow messages? Such an attack could render the whole concept of SDNs worth-

less as it would completely undermine their security. Of course such a vulnerability can likely be mitigated and fixed by updates to SDN controllers and the networking protocols it and its clients use for communication.

6.5.5 DoS attack of an SDN contoller

How well does the SDN controller deal with a DoS service on the controller itself?

The first point of note is that this obviously depends on the hardware, network bandwitdh and the operating environment of the controller. However for the sake of this experiment let us assume that the SDN controller does not have enough brute computing and networking capacity to deal with the DoS attack and that it is not hosted in an environment that can help it against the attack.

The main interest here is whether the SDN controller's clients can remain functional and retain some quality of service under a DoS attack on the SDN controller. An unmitigated DoS attack on the SDN controller itself may obviously render the controller itself at least somewhat non-functional but can the clients still remain functional with their existing configurations. If this is possible it would be a very positive milestone. Obviously in a such a situation new configuration changes would be slowed down to a halt or even made completely impossible under the DoS attack on the controller is mitigated or has stopped.

6.5.6 DoS attack of an SDN client

Can SDN controller handle a DoS attack on a client device? What if there is a DoS attack on a client device managed by an SDN controller? How does the SDN controller react to that?

The SDN controller can mitigate the attack by changing some flow rules for the client device. Another possibility of dealing with this is distributing resources in such a way that the client device can handle the DoS attack but in the case of a serious DoS attack this might not be a viable option. Distributing the resources differently might also affect the quality of other services hosted by clients of the SDN controller which might be a complete non-starter depending on the environment.

A type of DoS attack called DDoS attack can be simulated using performance testing tools such as Java-based JMeter to bombard the client device with a large amount of network traffic.

7 Conclusions

This thesis serves as an introduction to the concepts of network function virtualisation, service chaining and software-defined networking. On top of introducing these concepts some theoretical security analysis is performed on them as different concepts and as a system consisting of all three concepts.

The proposed experiment plan includes a realistic depiction of a complete computing and networking system resembling that of some web hosting services. Building this environment will provide one with a suitable albeit basic system to perform security testing against.

All of the parts of the proposed experiment system are introduced and briefly described so that one has the basic knowledge of all these components.

The selection of different kinds of network attacks for the experiment is obviously limited. The attacks were chosen based on the following criteria: potential damage, ease of performing, simplicity and the knowledge and experience of the author.

The limited selection of different kinds of attacks should not be considered sufficient for testing production systems but is meant as a good starting point. Using the experiment laid out in this thesis one can get a good basic understanding of the security of the descibed target system. Many of these attacks serve as basis for more advanced and complicated attacks. Thus failing to deal with these attacks means that the system most probably will fail against more advanced and motivated attackers.

Most of the different attack types described in the experiment plan section are also very broad. This is done on purpose. Listing very specific attacks would not be fruitful as new attacks are continuously invented and old attacks are improved. The selection of specific forms of these types of attacks is left for the performer of the experimenter to decide.

Bibliography

Blendin, J., J. Rückert, N. Leymann, G. Schyguda, and D. Hausheer. 2014a. "Demo: Software-Defined Network Service Chaining". In *2014 Third European Workshop on Software Defined Networks,* 139–140. doi:10.1109/EWSDN.2014.15.

———. 2014b. "Position Paper: Software-Defined Network Service Chaining". In *2014 Third European Workshop on Software Defined Networks,* 109–114. doi:10.1109/EWSDN.2014.14.

Chiosi, Margaret, Don Clarke, Peter Willis, Andy Reid, James Feger, Michael Bugenhagen, Waqar Khan, et al. 2012. *Network Functions Virtualisation – Introductory White Paper.* Published: A joint white paper by several ISPs published by ETSI. Visited on June 18, 2016. http://portal.etsi.org/NFV/NFV_White_Paper.pdf.

Corporation, The MITRE. 2015. *CVE-2015-3456.* Published: A Common Vulnerabilities and Exposures (CVE) listing by The MITRE Corporation. Visited on June 18, 2016. https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2015-3456.

Foster, N., A. Guha, M. Reitblatt, A. Story, M. J. Freedman, N. P. Katta, C. Monsanto, et al. 2013. "Languages for Software-Defined Networks". *IEEE Communications Magazine* 51, number 2 (): 128–134. ISSN: 0163-6804. doi:10.1109/MCOM.2013.6461197.

Gouda, M. G., and A. X. Liu. 2005. "A Model of Stateful Firewalls and Its Properties". In *2005 International Conference on Dependable Systems and Networks (DSN'05),* 128–137. doi:10.1109/DSN.2005.9.

Han, B., V. Gopalakrishnan, L. Ji, and S. Lee. 2015. "Network Function Virtualization: Challenges and Opportunities for Innovations". *IEEE Communications Magazine* 53, number 2 (): 90–97. ISSN: 0163-6804. doi:10.1109/MCOM.2015.7045396.

Hynninen, P., and M. Kauppinen. 2014. "A/B Testing: A Promising Tool for Customer Value Evaluation". In *2014 IEEE 1st International Workshop on Requirements Engineering and Testing (RET),* 16–17. doi:10.1109/RET.2014.6908673.

John, W., K. Pentikousis, G. Agapiou, E. Jacob, M. Kind, A. Manzalini, F. Risso, D. Staessens, R. Steinert, and C. Meirosu. 2013. “Research Directions in Network Service Chaining”. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For,* 1–7. doi:10.1109/SDN4FNS.2013.6702549.

Kreutz, D., F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. 2015. “Software-Defined Networking: A Comprehensive Survey”. *Proceedings of the IEEE* 103, number 1 (): 14–76. ISSN: 0018-9219. doi:10.1109/JPROC.2014.2371999.

Leivadeas, A., M. Falkner, I. Lambadaris, and G. Kesidis. 2016. “Resource Management and Orchestration for a Dynamic Service Chain Steering Model”. In *2016 IEEE Global Communications Conference (GLOBECOM),* 1–6. doi:10.1109/GLOCOM.2016.7842225.

Lin, P. C., Y. D. Lin, C. Y. Wu, Y. C. Lai, and Y. C. Kao. 2016. “Balanced Service Chaining in Software-Defined Networks with Network Function Virtualization”. *Computer* 49, number 11 (): 68–76. ISSN: 0018-9162. doi:10.1109/MC.2016.349.

McKeown, Nick, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. “OpenFlow: Enabling Innovation in Campus Networks”. *SIGCOMM Comput. Commun. Rev.* 38, number 2 (): 69–74. ISSN: 0146-4833, visited on July 30, 2016. doi:10.1145/1355734.1355746. http://doi.acm.org/10.1145/1355734.1355746.

Medhat, A. M., T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz. 2017. “Service Function Chaining in Next Generation Networks: State of the Art and Research Challenges”. *IEEE Communications Magazine* 55, number 2 (): 216–223. ISSN: 0163-6804. doi:10.1109/MCOM.2016.1600219RP.

Yang, Hae-Yong, Kyung-Hoon Lee, and Sung-Jea Ko. 2008. “Communication Quality of Voice over TCP Used for Firewall Traversal”. In *2008 IEEE International Conference on Multimedia and Expo,* 29–32. doi:10.1109/ICME.2008.4607363.