Jarno Lähteenmäki

Scenario based security evaluation: Generic OpenFlow network

School of Electrical Engineering

Espoo 16.11.2014

Network Technology, special assignment supervisor and advisor:

D.Sc. (Tech.) Mikko Särelä

Aalto University
School of Electrical Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

Author: Jarno Lähteenmäki

Title: Scenario based security evaluation: Generic OpenFlow network

Date: 16.11.2014 Language: English Number of pages: 5+35

Department of Communications and Networking

Professorship: Networking Technology Code: S.38

Supervisor and instructor: D.Sc. (Tech.) Mikko Särelä

Demand for network programmability was recognized when development of protocols slowed down due to network inflexibilities in 1980s. Research speeded up and many proposals were made to solve architectural issues during 2000s. Academic world put up an initiative to build up new programmable network architecture later 2000s. OpenFlow was born.

In modern public network infrastructures the security of the network architecture is crucial to archive data confidentiality, integrity and authenticity, yet high availability. Many studies have shown that there are many security vulnerabilities and issues on current OpenFlow implementations and even in OpenFlow specification itself. Many proposals have been made to enhance these known issues. In this research, the scenario based security evaluation of the generic OpenFlow network architecture was carried out using technology publications and literature. The security evaluation framework was used in security assessment.

Proposed risk mitigation patterns were found to be effective on most of the cases for all 13 identified and evaluated scenarios. Lack of mandatory encryption and authentication in OpenFlow control channel were most critical risks on general level. OpenFlow specification should provide clear guidance how this should be implemented to guarantee inter-operability between different vendors. Short term solution is to use IPSec. Second critical issue was that bugs and vulnerabilities in OpenFlow controller and switch software are causing major risks for security. Proper quality assurance process, testing methods and evaluation are needed to enhance security on all phases of the software production.

Current OpenFlow implementations are suffering poor security. Tolerable level can be reached by utilizing small enhancements. There are still many areas which need to be researched to archive solid foundation for software defined networks of the future.

Keywords: OpenFlow, security, evaluation, scenario

Contents

Abbreviation

Abbreviation	Definition
API	Application Programming Interface
ARP	Address Resolution Protocol
ASIC	Application Specific Integrated Circuit
CPU	Central Processing Unit, Processor
DoS	Denial of Service
DPID	Data Path identification, OpenFlow specification
DTLS	Datagram Transport Layer Security
ETSI	European Telecommunications Standards Institute
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IPSec	IP Security
MiM	Man-in-the-Middle Attack
MAC	Media Access Control
NTP	Network Time Protocol
OFP	OpenFlow Protocol
QA	Quality Assurance
REST	Representational state transfer architecture
SDN	Software Defined Networking
SEF	Security Evaluation Framework
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transmission Layer Security
UDP	User Datagram Protocol
URI	Uniform Resource Identifier

1 Introduction

Distributed packet based network technology has history of more than 50 years. First traditional networks were developed on 1960s and 70s. Programmable network research started on 1980s when development of protocols slowed down due to inflexibilities on existing network models. Since the launch of first packet based networks there have been security related issues. When user base was low there were no large interest to enhance the security. At the end of 2009 academic put up co-operation initiative to build up new programmable network architecture and OpenFlow was born.

While carriers are rolling out Software Defined Networks (SDN) with increasing speed, the security of whole architecture has had challenges to keep up with the pace. Multiple OpenFlow specifications have been released since the first release on year 2009. Many new features have been presented but hardly none which would enhance the security of the OpenFlow protocol itself. OpenFlow has become a mainstream control-data interface protocol. Due to relatively young age of OpenFlow, there are still quite many areas which are under development. Especially security related functions.

SDN has got a footage largely on network communities on 2010s. Advantage of using SDN is realized with extensive programmability and flexibility. At the same time it bring totally new attack vectors for intruders. Like the definition of term SDN states, network is controlled by a software based applications.

All software are prone for defects, bugs, flaws and risks but these can be managed by utilizing effective tools. Some of these problems remains in the application for a long time without an implication and some might trigger a problem quite rapidly or frequently. To handle these problems, software products should be evaluated and tested. By using scenario based evaluation of the OpenFlow network architecture, an organization, planning to deploy an OpenFlow network, can identify risks beforehand. Security evaluation framework provides iterative way to enhance security to level that meets requirements of the organization. Furthermore, to extend a risk handling, an organization can implement more advanced checking tools which can even monitor control traffic on real time if needed by utilizing solutions that networking and security researchers have developed.

In this research, security of the OpenFlow based SDN architecture was assessed using scenario based security evaluation framework. Scientific publications related to security, networking and programming areas were used as a source material. The security evaluation framework was used as a methodology to assess different scenarios.

The research problem in this study was to find out what should be taken into account from security point of view when an organization is planning to roll out OpenFlow network. This study does not provide overall picture of all possible scenarios but pinpoints some of the most crucial areas of interest instead.

In this paper, software problem categories, software evaluation methods and OpenFlow specification are described in section two. In third section, the OpenFlow security analysis is carried out by first taking a look to the related work done by

other researchers and then evaluating found issues using security evaluation framework. In fourth section, recommendations are presented that can be used for making OpenFlow based network more secure. In section five the paper is concluded.

2 Theoretical background

SDN and especially OpenFlow based network technologies have evolved rapidly over the past years to become a feasible network technologies. So far, there have been no studies which addresses what kind of security and vulnerability issues must taken into account when planning to implement OpenFlow networks in real production environment. This study pinpoints some of the major areas.

Confidentiality, integrity, authenticity and availability are the basic properties of a secure network communication [1]. Security is not one action that can be done to ensure that these properties exists. Instead, security must be executed on all levels and all phases of the design, implementation and execution of the system. There are evidences that poor design and lack of software quality assurance can result huge disasters by enabling catastrophic software failures [2]. As networks have become a critical infrastructure for many environment, like hospitals, nuclear power-plants and in-vehicle control-systems, it has become a crucial that software quality is set to high standard.

Following sections give a brief theoretical introduction to the context. First, software problem categories and software evaluation methods are presented. Then, OpenFlow specification is described.

2.1 Software problem categories

According to McGraw [3], software based security problems can be categorized into four classes. First three, defects; bugs; flaws, represents different kind of occurrence pattern and mitigation method. The fourth class, risk, is caused as an effect of a flaw and a bug. Some of these are quite easy to observe, but there are cases which are never found due to probabilistic nature of problem occurrence [4].

First, defect is a vulnerability, that was born in implementation or design phase of the software production. A software might contain a defect many years. Defects are challenging to detect.

Second, bug is a problem in software code, that was born in implementation phase. A software might contain bugs which are never executed and never found. Bugs are relatively easy to discover and some testing automation can be used for bug hunting.

Third, flaw is a problem which exist in the software code and was born in design phase. It is at deeper level in the code than a bug. Flaws are more challenging to find. Software code containing a flaw might get executed but not triggered. On most cases some special conditions need to be set to trigger the flaw.

Fourth, risk is the realization of an event of a bug or a flaw with some probability and impact. Measure of risk takes into account the possible damage a bug or flaw causes. [3]

Though we talk here about software related problems, all these categories have an affect also to network quality. Traditional networks have software based components underneath which might contain same kind of problems. When we move to area of SDN networks the role of software products becomes more crucial due to centralized

control plane. Before we move to more SDN related items we next take a look at the software assessment and evaluation methods.

2.2 Software evaluation methods

Software security evaluation is crucial part of any software release management process. Different methods are suiting for different needs. It is important to choose right method for the task to archive feasible results. Here are described some of the methods.

2.2.1 Security evaluation methods

Architectural evaluation of software security can be divided into four classes. (1) Mathematical methods, (2) simulation based evaluation methods, (3) experience-based assessment and (4) scenario-based evaluation. [5] In this case study, we are focusing on scenario-based evaluation and specifically we are using the security evaluation framework (SEF) for analysis.

First, in a mathematical method, the software is modelled as formulas and statistics. The model can reflect the software quality requirements accurately only on some restricted domain. In that domain the security can be evaluated definitely. It is nearly impossible to model all possible domains. One reason is that software security is measured with absence of security risks mentioned in previous section. Mathematical model instead provides a way to evaluate a presence of an event. Other reason is that flaws, bugs and defects are put into the code in implementation phase and normally evaluation is done against reference architecture.

Second class, simulation based evaluation, can be used for partial penetration tests. Simulated software environment is never the exact copy of actual software but mimics only most important part of the software instead. Simulation can be used to test against *known* vulnerabilities and problems.

Third class, experience-based assessment, is based on specialists experience of the software security. Security architects reviews the application design, coding style and testing methods.

Fourth class, scenario-based evaluation can provide good level of accuracy and systematic process. In this method some specific quality parameters are assessed by setting up a scenarios that represents different risks. Scenarios are assigned into different profiles that are used to evaluate the actual software architecture. [5]

2.2.2 The Software Evaluation Framework

SEF [5] describes a process which can be used for analysing given architecture towards defined scenarios. The process contains six phases.

First, the process starts with defining an evaluation goal. Generally goal is one of quantitative assessment, qualitative assessment or trade-off assessment.

In second phase, security scenarios are created. A 5-tuple scenario contains a specific security requirement, a threat towards the system, a precondition (enabler) that must be met before scenario can occur, a behaviour of the system when specific

scenario is realized and a pattern which is a set of actions which are required to mitigate the risk to prevent a threat to realize if applicable.

In third phase, security profiles are created. One way to divide scenarios is risk based approach. The risk rating method based on OWASP's RRM model [6] is in formula 1 where P is the probability of the event i and I is the impact of the event when it has occurred. Values for P and I are ranging from 0 to 9. In RRM, relative level is low when value is from 0 to 3, medium when value is from 3 to 6 and high when value is from 6 to 9. The R can be calculated from these values. Security objectives for networks, as mentioned earlier, are confidentiality, integrity, availability and authenticity.

$$R_i = P_i * I_i \tag{1}$$

In fourth phase, the full scenario table can be created using threat, patter, security objectives and risk information. Risk and probability values are estimated using experts opinion. [5] In this phase it is crucial that person who is doing this subjective analysis has certain level of expertise.

In fifth phase, the actual evaluation is done by comparing required security properties with provided scenarios. Possible scenario values for different factors are listed in Technical impact factors -table (table 1) and vulnerability factors -table (table 2). Impact of the scenario is average of the technical impact factors of a scenario. Probability of the scenario is average of vulnerability factors of a scenario. In this study risk based evaluation was used.

Table 1: Technical impact factors [5]

Impact of Confidentiality (IC)	[2]	non-sensitive data disclosed
	[6]	sensitive data disclosed
	[7]	critical data disclosed
	[9]	all data disclosed
Impact of Confidentiality (II)	[1]	minimal corrupt data
	[6]	seriously corrupt data
	[7]	extensive corrupt data
	[9]	all data totally corrupt
Impact of Availability (IAV)	[1]	minimal services interrupted
	[6]	serious services interrupted
	[7]	extensive services interrupted
	[9]	all services lost
Impact of Authenticity (IAU)	[1]	fully traceable
	[7]	possibly traceable
	[9]	completely anonymous

In sixth and the last phase, the evaluation results are scrutinized if given system architecture meet security requirements or not. Architecture can be revised and evaluated again until required level of security is met.

Table 2: Vulnerability factors [5]

Easy of Discovery	[1]	practically impossible
	[3]	difficult
	[7]	easy
	[9]	automated tool available
Ease of Exploit	[1]	theoretical
	[3]	difficult
	[5]	easy
	[9]	automated tool available
Publicity	[1]	unknown
	[3]	hidden
	[7]	obvious
	[9]	public knowledge

The ultimate goal of the security evaluation framework is to enhance security of the software systems. Even though the framework is targeted to software applications it is also usable to evaluate any other systems like network architectures. Later in chapter 3.4 we use the SEF model for evaluating OpenFlow based network environment. Results are used for enhancing the architecture.

2.3 Threat vectors

A threat vector is a way or a path to penetrate to the system or cause some malicious effect by attacker. OpenFlow based network is attractive for attackers due to centralized control-plane which enables also potential control of the whole network for the attacker [7]. Here are described seven individual threat vectors categorized by Kreutz et al. [7] and additional two categories which are proposed by this study.

First, (1) forged or faked traffic flow attack vector can be initiated by malicious user or by a faulty device. The purpose of the deliberate attack is to cause a denial of service attack. This threat vector is not specific for OpenFlow, but it is also possible in traditional network.

Second, (2) attacks on vulnerabilities in devices can cause a slowness to packet forwarding, cloned or dropped packets. Also some data can be rerouted for data theft purposes. This threat vector is not specific for OpenFlow but programmability amplifies the impact of an attack.

Third, (3) attacks on control plane communication between a controller and a switch is specific for OpenFlow protocol and is not present in traditional networks. Lack of encrypted and authenticated communication and some weaknesses in SSL/TLS protocol enables this vector.

Fourth, (4) attacks on and vulnerabilities in controllers is OpenFlow specific. Attacker using this vector may get a control of the whole network.

Fifth, (5) lack of mechanisms to ensure trust between the controller and management applications is specific for OpenFlow protocol. Upper-layer application may

contain bugs and vulnerabilities that enables this vector.

Sixth, (6) attacks on and vulnerabilities on administrative station is not OpenFlow specific. In OpenFlow controlled network by installing malicious code to the administrative station attacker can gain control of the controller and thus control a part or whole network.

Seventh, (7) lack of trusted resources for forensics and remediation is not OpenFlow specific. While all layers of the network might be compromised, there has to be separate place for storing logging and tracing information that cannot be changed.

Eighth, (8) interoperability problems and vulnerabilities enabled by the insufficient OpenFlow specification. This threat vector is challenging to categorize on detailed level. Individual solutions might be secure as such but when used in mixed environment those might appear to be vulnerable.

Ninth, (9) information disclosure of the actual OpenFlow based network itself. This kind of threat reveals information of the network itself and it can be used for attacks toward the network.

These nine threat vectors are referred in this report. After taking a look to these attack paths we move our focus to the network vulnerability types.

2.4 Network vulnerability types

Since the launch of first data networks, many vulnerabilities have been discovered and general types have formed. There are well-known network attack and vulnerability types which are referred in the report. Used types are explained here.

2.4.1 Man-in-the-middle attack

In a man-in-the-middle (MiM) attack an attacker can reroute a traffic from the source to destination via a middle box by spoofing underlying network address or name. This attack is well known type of vulnerability in network security field. It can be used for revealing the actual information of the traffic. Protocols that are lacking an encryption or good implementation of secret key exchange are more easily vulnerable to this attack. [8]

2.4.2 Denial of Service attack

In a Denial-of-Service (DoS) attack one source host sends malformed or amplified data packets to the target host to cause disturbance to the targeted service. Target can be single service, host or whole network of some organisation or country. A variant of DoS is Distributed DoS (DDoS) attack where multiple hosts are used as a source to attack on single target. [9]

2.4.3 ARP spoofing

ARP spoofing, also known as ARP cache poisoning, is network intrusion method where attacker steals a MAC address of a gateway or a host. Methods to cheat a

target host is to send false ARP reply messages to the host that is under attack. [10]

2.4.4 Information disclosure

When network behaves on some predictable pattern, when forwarding a traffic, it is possible to probe from outside how network is acting. Attacker might be able to get an information that can be used to target more powerful attack. [11]

These four are mentioned because these are referred from elsewhere in the report. There are actually more known network vulnerabilities than these mentioned. We leave these practical vulnerabilities for a while and take a look at the OpenFlow specification.

2.5 OpenFlow specification

OpenFlow specification is an open standard released by Open Networking Foundation [12]. It was born on 2000s when academic world started studying how campus network could be built using programmable network technology. First formal release was made at the end of year 2009. In this study, we are focusing on latest specification available, 1.4.0 version.

Generic SDN architecture contains three layers as described in figure 1. OpenFlow specification covers two bottom layers of the architecture - control layer and infrastructure layer.

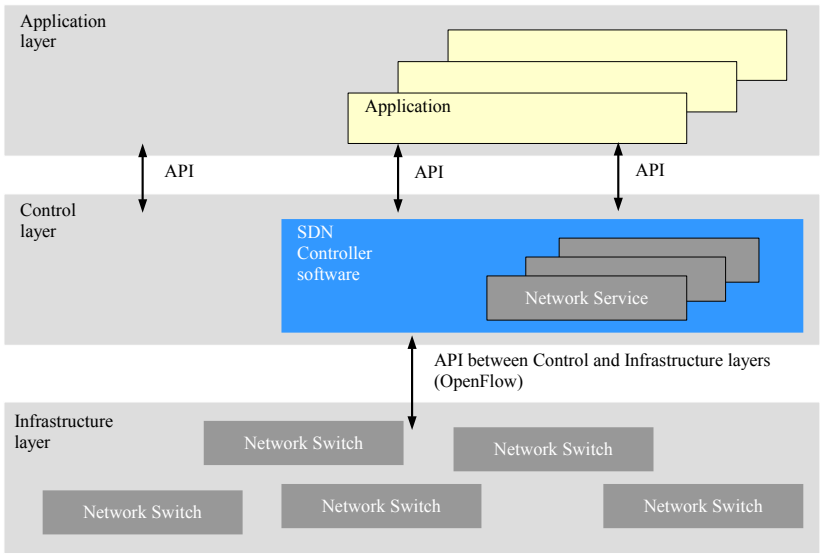

Figure 1: Generic Software-Defined Network Architecture [13]

OpenFlow specification defines software interfaces between network devices, that takes care of forwarding plane, and controllers, that takes care of the control plane.

On switch end, there is OpenFlow channel software component which handles all requests from and to controller. Connection from a switch to primary controller is defined as main connection. Connections from a switch to other secondary controllers are defined as auxiliary connections. The main control session between a OpenFlow switch and the controller uses TCP protocol on port 6653. The use of transmission layer security (TLS) is optional but preferred. Switch and controller should also support mutual certificate based authentication. [12]

Each OpenFlow enabled switch has one or more flow tables that contains a flow handling information programmed by controller. When a packet arrives to the network device it is checked against flow tables.

If matching rule is found from a flow table, switch can modify the packet and update fields if needed. It can update an action set, that are executed just before the packet leaves the switch. The switch can also update the metadata if needed. It is used to pass information between flow tables in the switch.

If matching flow entry is not found from the table, there is three options the switch can do depending on the default rule of the flow table. First, the switch can forward the packet to the controller via OpenFlow channel. Second, it can drop the packet. Third, it can continue to the next flow table. [12] The flow handling and per-table packet processing by the OpenFlow specification is shown in figure 2.

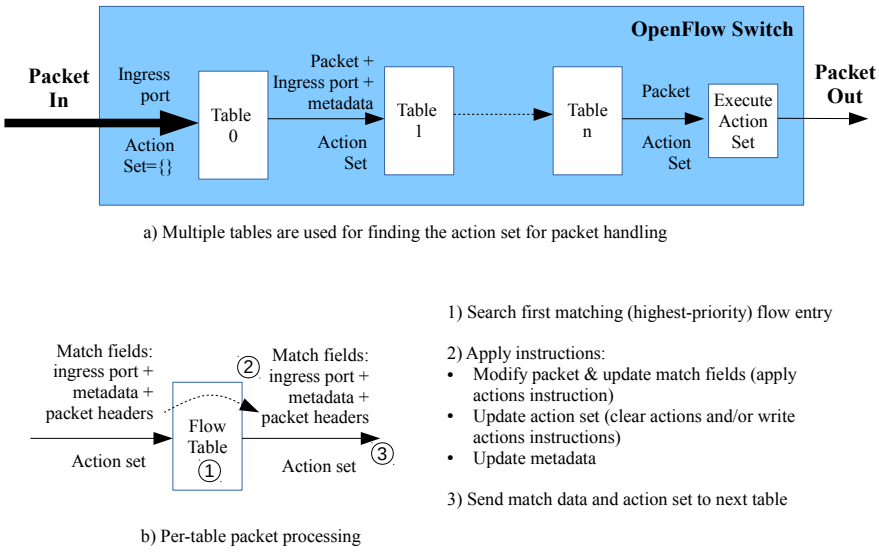

Figure 2: Packet flow through the processing pipeline [12]

Each communication packet in the OpenFlow communication protocol (OFP) contains a header and a message. Header defines OFP version, message payload type, message length and communication packet identifier. [12] The OpenFlow header structure is shown in figure 3. OpenFlow specification defines many message types. Here are listed five most commonly used.

```
struct ofp_header {
        uint8_t version;
        uint8_t type;
        uint16_t lenght;
        uint32_t xid;
};
OFP_ASSERT(sizeof(struct ofp_header) == 8);
```

Figure 3: OpenFlow-header defined in specification [12]

- *Packet-In:* When OpenFlow switch sends a packet to the controller it uses *Packet-In* message type. This kind of situation occurs when flow processing stops in table-miss flow entry and action is defined, so that switch sends the packet to the controller.
- *Features:* Controller can query switch capabilities by using Features message type. Controller typically use this right after communication channel has been established between the controller and the switch.
- *Configuration:* Using Configuration message type controller can set and query switch configuration.
- *Modify-State:* Controller uses Modify-State message type to add, modify or delete flow entries in the switch's flow table. This message type is also used for changing switch port properties.
- *Packet-Out:* Packets, that are destined to be sent out from the switch port, are passed from controller to the switch using Packet-out message type.

OFP header and message handling is implemented in software components. In controller it is purely software application and in switches either using software or application specific integrated circuit (ASIC). OpenFlow specification does not define how the actual implementation should be done.

In this study we are referring to generic OpenFlow network environment and architecture. Fundamental parts of this structure is shown in figure 4.

2.6 Summary

In this chapter we went through theory and specification part of the study. Software problems are categorized into different classes to help theoretical handling of issues. To minimize probability of vulnerabilities, bugs, defects or risks, it is good practise to evaluate the software and system architecture. Security evaluation framework can be used for assessing system using scenario based approach. In SEF method, threats are categorized into scenarios which have mitigation patters, technical impact factor and occurrence probability. Using these informations, scenarios are evaluated and examined if it can meet required security level or not.

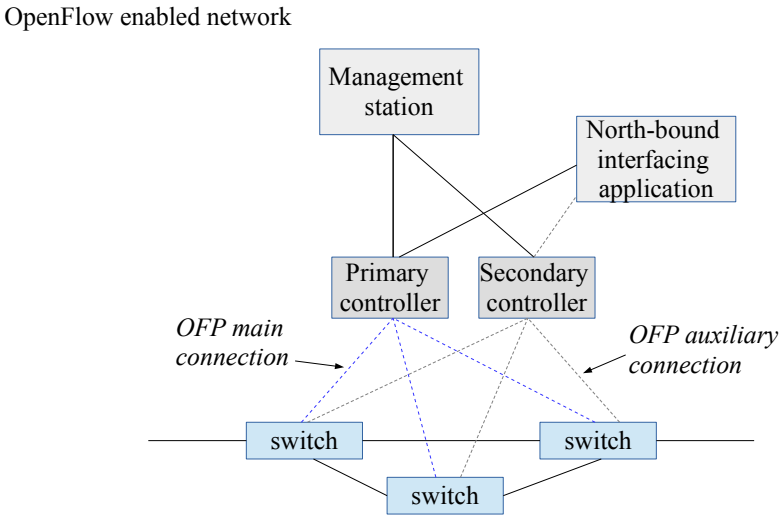

Figure 4: Generic OpenFlow based network structure

OpenFlow specification was released on year 2009 by Open Networking Foundation. Due to relatively young age of the specification, there are still much to improve in security related features in specification. OpenFlow itself is the communication protocol between controller and a switch.

After taking a look at the theory of software quality and OpenFlow specification, we proceed to practical research and what have been done in the past.

3 Security evaluation

Network programmability and security of it have been under investigation on several researches. During this study, some of the past research results were examined. Those are presented in related work section. After that, OpenFlow protocol level security issues were examined using OpenFlow 1.4.0 specification document and vulnerability reports. In third section we also take a short look at the code verification and debugging applications that could be used for enhancing quality assurance (QA) process. In final section the actual evaluation is carried out.

3.1 Related work

Since the release of OpenFlow 1.0 back in 2009, many reports have been published regarding security issues of OpenFlow specification. Most of the examined reports were focusing on issues other that OpenFlow security itself, nevertheless many reported those and concluded how immature OpenFlow security currently was. Here are presented some of those which where found to have most affect.

3.1.1 Hardware programmability and controller performance

Sezer et al. [14] reported their research with publication named "Are we ready for SDN? Implementation challenges for software-defined networks". They studied if SDN technology was ready for a prime time or not. From security point of view they raised a couple of items.

First, modern networks requires capacities from 100Gb per second to 1Tb per second. To archive such a high speed network, forwarding packets must be implemented using a custom ASIC. OpenFlow requires high level of programmability from network equipment to archive flexibility. General purpose CPU chips contains programmability but does not provide enough bandwidth.

Second, Sezer et al. also noted that controller and network node scalability is an issue. When number of nodes increases in the network the statistical behaviour results also increased latency between a node and controller. When splitting the controller into multiple instances for scalability reason it also brings new challenge in form of inter-controller communication. To be able to keep full network view all the time, controllers need to exchange a lot of state information.

Sezer et al. concluded their report by stating that future networks will follow the past trends. Expanding capacity requirements will push programmability into every device on the network. Meanwhile, technology standardization bodies are developing methods to overcome scalability issues and improving security at the same time. Nevertheless, the challenge is that industry working groups will need central coordination to archive interoperability and to take advantage of existing standards. [14] Though Open Networking Alliance has quite strong support from software and hardware vendors it should build liaison to other industry working groups like IETF and ETSI. At the present day there is no industry which can work in a vacuum.

3.1.2 OpenFlow vulnerabilities

Benton et al. [15] performed an assessment of OpenFlow vulnerabilities. They raised multiple security issues of OpenFlow protocol.

The optionality of TLS was raised as most crucial vulnerability. This single vulnerability yields many different attack vectors. First, networks that are using only plain-text communication in switch to controller communication are vulnerable to man-in-the-middle attack. If switch to controller network is secure and dedicated the risk is lower but still it exists. Especially when control traffic is carried via third party network for example from remote office or in campus-style network this becomes an issue. The challenge is that man-in-the-middle attacks are hard to detect and it can be undetected for a long time. An attacker can seize a control of any subsequent switch it detects after collecting a data for picturing a full network. [15]

Second, some vendors have implemented a listener mode to the switches which enables unauthenticated connections to the switches. This is not protocol level issue but organizations that are implementing OpenFlow networks are facing this kind of issues.

Third, in some controller software the implementation of switch authentication is not flawless. If authentication is not implemented properly an attacker can gain an information that can be used for denial of service attack and potentially tear down some part of the network by sending a false information using a switch source address to the controller.

Fourth, there might be situations where switch fails to either remove a rule requested by controller or adds faulty rule. Currently there is no direct way to check anomalies between controller state information and a real switch flow table.

Fifth, if flow-rule planning is not done carefully it is possible to cause a DoS situation by sending more Packet-In messages to the controller than it can handle. Also Flow-Mod messages can cause excessive amount of traffic for the controller.

Sixth, controller vulnerabilities that enables an attacker to use a vulnerability of one interfacing application to control the whole network.

Their research was not very wide, pointing out only some of the current issues. Nonetheless, the conclusion summarises quite well the current situation of OpenFlow development. Benton et al. [15] says in their conclusions chapter:

> OpenFlow risks repeating the errors of other insecure network management protocols (e.g. Telnet, SNMPv2, TFTP) where physical security was initially the only security and adoption of transport security lagged.

3.1.3 Threat vectors and secure control platform

Kreutz et al. [7] researched security of the SDN architecture. They presented a model for threat vectors as described in chapter 2.3. They also presented a general design of the secure SDN control platform and introduced several mechanisms that offers a solution for the threat vectors.

Secure control platform had two major advancements. It archived a fault tolerance by using replication. It also enabled an interface that can be used for masking

malicious or faulty applications.

For fault tolerance, proposed method was to replicate controllers so that those are using different controller software. Second method was self-healing mechanisms that can be use for replacing vulnerable or faulty component by it self and return to normal operational state much faster after a failure.

For masking malicious of faulty applications, proposed method was dynamic device association where OpenFlow switch could dynamically change from one controller to another when it notices that first controller is not in secure state. Furthermore, a method to isolate infected switch from the network was proposed. In this method other switches can propose a switch to the blacklist.

By presenting solutions for a secure SDN core and showing several threats Kreutz et al. argued that there is a strong need for further discussion in the SDN community to speed up solving multiple critical issues. So far, SDN has repeated same mistakes that have been present in traditional networks for a long times. With a support from database, programming and systems communities these problems can be fixed. [7]

3.1.4 Lowering effect of an attack

Klöti et al. [11] did a security analysis of OpenFlow protocol and network setups. In their study they found that different attack vectors can be eliminated using certain techniques.

To prevent DoS attacks they proposed rate limiting the control channel between a switch and controller. This damp the load to the controller in event of DoS attack. By using selective event filtering and packet dropping, controller can first select to receive only part of the messages from switches, then identify the pattern of attack and then install new packet-drop flow rule to damp only the attack traffic. [11]

Klöti et al. concluded their paper by recommending numerous prevention and mitigation techniques. They admitted that OpenFlow applications and standards are not secure enough, thus, Open Networking Foundation should work on this area to fulfil the gap. [11] It is crucial to have secure by design bedrock for the rest of the applications build on top of OpenFlow foundation.

3.1.5 Experimenting attacks to OpenFlow infrastructure

Romao et al. [16] did a practical analysis of OpenFlow security. Their goal was to show how OpenFlow weaknesses can be exploited.

They experimented different attacks like communication channel interruption using ARP spoofing, port mirroring with OpenFlow using man-in-the-middle attack between controller and a switch and DoS attack to switch flow table by sending plain ICMP echo-request packets with different destination addresses. They managed successfully to execute all attacks they tried. [16]

They proposed four enhancements to make OpenFlow behaviour less drastic. First, OpenFlow switches should include fallback controller with minimalistic rule set that can be used when the connection to main controller is lost. Second, all

controller to switch communication should use dedicated physical or virtual connection so that intruder has no way to communicate with switch management port. Communication channel must use TLS encryption. If hardware vendor has not implemented TLS in the software this feature should be demanded from the vendor. Third, all OpenFlow enables switches should have methods to detect and prevent ARP spoofing. Fourth, controller software should has feature to detect network anomalies. Despite these enhancements, in their conclusion Romao et al. stated that OpenFlow is not ready for production networks. [16]

3.1.6 Evaluation of controller softwares

Shalimov et al. [17] did a analysis of open source OpenFlow controllers. Tested softwares were NOX, POX, Beacon, Floodlight, MuL, Maestro and Ryu.

They analysed scalability, reliability, performance and security of different controllers. They intentionally sent packets with malformed OpenFlow header or malformed OpenFlow messages to the controller. None of the tested controllers passed all test cleanly. Some of the controllers crashed, some did not notice that the packet was malformed, some noticed malformed packet though.

As a conclusion they pointed that many of tested controllers contained vulnerabilities that could be used for the malware attacks. Controller software community should take security seriously. [17] Most open source SDN controller projects have to improve QA processes to ensure that software problems are found during the development cycle.

3.1.7 Flow rule verification

To extend security capabilities in OpenFlow network, Khurshid et al. [18] presented a VeriFlow method which brings new layer between controller and network switches. Main purpose of this new layer was to enable real-time network-wide checking of invariants. This debugging tool can detect faulty OpenFlow-rules, that have been issued by upper-layer application. At the best case VeriFlow can detect and prevent such rules to reach flow-table of network devices and therefore prevent potential security breaching flow-rules. During the evaluation of VeriFlow, Khurshid et al. did a benchmark for the tool. Flow rule checking brings extra delay when inserting new rules or removing old ones. In ten host network extra delay was 15 ms compared to controller-switch interface without VeriFlow.

VeriFlow is using a mathematical testing model to test if individual flow-rule meet security requirement or not. Method was described in chapter 2.2. Therefore we can state that VeriFlow can be fully accurate only on some finite domain and with pre-defined set of matching rules. It might miss some faulty rules if it is out of scope. Although VeriFlow is very powerful tool, it should not be used as a principal way to ensure network security but as a security assurance tool to prove in statistical way that upper-layer applications, including controller software, does not issue faulty rules at the first place.

3.1.8 Summary

Many research results concluded that security of OpenFlow specification and implementations are not rock solid. Problems are multifaceted. Most of the problems could have been prevented with secure by design practices. Also there are aspects that emphasizes the meaning of OpenFlow specification itself. Identified problems are analysed using a scenario based evaluation in chapter 3.4. Now we move to OpenFlow specification part of the study.

3.2 OpenFlow protocol level security issues

OpenFlow is relatively recent specification. The version 1.0 was released at the end of 2009. Ever since, there have been substantial number of releases that provides more features but hardly none which would directly enhance the security of OpenFlow architecture itself. We did use OpenFlow specification version 1.4.0 in this study.

Controller facing applications are not part of this study because it is not considered as part of the OpenFlow scope. These application level features thoroughly brigs new kind of security issues and is worth for further study. In following chapters, security related topics are presented to give an overall picture of the current situation of OpenFlow security.

3.2.1 Switch level

OpenFlow specification does not define how to handle malformed or corrupted packets [12]. Implementing vendor must choose if packet is dropped or forwarded in these cases. Forwarding behaviour is archived if packet header checksum is omitted.

OpenFlow switch might be enabler for DoS attack if hardware constrains are not taken into account. Although modern datacenter core switch gear might contain very powerful CPUs like typically used in servers [19], smaller edge switches towards access devices includes much less computing power [20]. As mentioned in chapter 3.2.3 the TLS communication between the controller and switch is optional but preferred. When encrypted communication is used, performance issue on access switches is raised. Switch has to have enough performance on slow-path to encrypt and decrypt *Packet-In*, *Packet-Out* messages and other controller communication like *Configuration* and *Modify-State* messages, furthermore it must reserve performance for platform monitoring, logging and other housekeeping operations.

If OpenFlow switch contains performance bottlenecks it is possible enabler for Denial-of-Service attack by using threat vector 1 (described in 2.3). This vulnerability can be probed using information disclosure threat vector (9).

3.2.2 Controller level

When using a multi controller environment, operational misconfigurations and security issues becomes more likely due to fact that OpenFlow specification does not guide how controller to controller traffic should be handled. To archive best and most secure control platform, controllers should be replicated and diversified [7].

To archive diversity at least two different controllers software should be used. This yields doubled testing and integration work. This also makes QA process more complex.

[21] reports an Open Floodlight vulnerability that can be used for connecting to the controller so that legitimate switch is blocked and booted as defined in protocol. This vulnerability is using threat vector 3 (described in 2.3), attack on control plane communication between controller and a switch. In this attack, attacker manages to get switch identification information *datapath_id* (DPID). The actual DPID can be revealed using very simple procedure. Open Floodlight controller provides REST API by default which shows information of all switches and their information including DPID. Access to the REST API URI is unauthenticated and is using plaintext HTTP communication method. The URI in Floodlight is http://<controller IP>:8080/wm/core/controller/switches/json. The controller IP is pretty easy to search using a basic network scanning for TCP port 6633. Of course attacker has to have access to the management network first. Most probable way to achieve it is to use threat vector 6, target vulnerable administrative station by inserting malware and using a backdoor.

3.2.3 Controller-switch interface

OpenFlow specification states that encryption in communication from switch to controller is optional [12]. Proposed method is TLS/TCP on main connection and optionally DTLS/UDP on auxiliary connection. Specification does not state which version of TLS should be used. It is known that TLS and DTLS versions 1.0 are vulnerable and should not be used in high security applications [22]. This type of attack is using a threat vector 3.

Furthermore, specification does not require certificate based mutual authentication of the switch and the controller. This raises a risk, that must be considered case by case how to mitigate. There is two alternative options to address this. First option is to considering recommended certificate based authentication as a mandatory. Despite the fact that these actions makes control channel more robust against man-in-middle attacks there is still some scenarios where communication level can be disturbed. Other option to mitigate the risk is to use a IPSec protocol between the switch and a controller.

When certificate based authentication is used, it is possible that authentication fails for example due to signature verification, certificate expiration or device time offset. Specification does not provide any guidance how these situation should be handled. Lack of definition yields most probably a vendor specific implementations and thus inoperable cross-vendor solutions. Safest way to handle these situations would be to initiate a date-time update on the switch for example using NTP and then retry controller connection. It should be avoided to fall back to unencrypted connection because that will lead to unsecure implementation by enabling new attack vector for intruders.

Crucial point for interoperability in certificate based authentication is the use of same fields on the both ends. Current OpenFlow specification does not provide

definition which fields should be used. When implementing switch and controller software a flexibility to choose an arbitrary certificate format should be retained. This lowers the possibility for operational misconfigurations and thus possible security vulnerabilities.

Another certificate related lack in specification is absence of guidance what exchanging certificates in both directions actually means. Real three-way handshake should be implemented to prevent a possibility of man-in-the-middle attacks. How this should be implemented in DTLS case remains for further study.

When authentication is not used correctly in switch to controller interface, attacker can send a forged packets to the controller using an address of the OpenFlow switch. When packet reaches the controller it assumes that is packet is from the switch and handles it accordingly. This attack type is using a threat vector 3.

3.3 Code verification, debugging and security breach analysis

OpenFlow protocol is implemented as a software code in a switch and controller that is referred here as a OpenFlow client application and server application accordingly. The software is written using some arbitrary programming language. Thus, all problem categories mentioned in chapter 2.1 can occur in final implemented code.

It is good practice to use code verification and testing tools to minimize software risks. OpenFlow enabled application can be automatically tested using NICE tool. It can help to uncover bugs by doing a model checking and symbolic execution. [23] Another option is to use Anteater. It checks network environment for network invariants in the data plane in real time. Benefit of this approach is that it can cover all protocols and errors caused by faulty switch firmware or problems in controller to switch control channel. [24]

Debugging OFP based network problems returns practically to analysis of flow rule insertions and removals from controller to switches. To be able to track what kind of rules are inserted and removed, rules must be logged. This feature can be implemented to path between a controller and a switch using controller extension or separate middle-box. [25] This logging feature can also be used for audit trail analysis if log entries are stored to hamper proof storage so that it cannot be changed.

3.4 Security evaluation

Goal of this evaluation was to do a qualitative assessment of generic OpenFlow enabled network environment. This evaluation can give an indication if presented architecture supports required security requirements. Security evaluation framework was used for assessment as described in chapter 2.2.2.

Threats were gleaned from research reports studied during this work. Each scenario was mapped to threat vector. Then security objectives and probabilities were defined using author's previous security and network knowledge and recommendations made by other report authors. In table 3 are described six high risk scenarios out of 13 total scenarios, their security requirements, probabilities and calculated

risks as a summary. All detailed scenarios and values for technical impact factors and vulnerability factors are shown in appendix A. Selection criteria for most successful pattern is a mitigation method that can lower probability most effectively. In this study patterns were chosen using results from related work and author's previous knowledge of networks and security.

Table 3: The Full Security Table

				SO *)	Pr **)	
S	**Threat**	**TV**	**Pattern**	**Total**	**Total**	**Risk**
6	Abuse of switch listener mode	2	Disable switch listener mode	7	6	43
4	Lack of mandatory encryption in OpenFlow protocol	3	Secure switch to controller network and use IPSec to secure transport	5	8	38
5	Lack of proper authentication in OpenFlow protocol	3	Secure switch to controller network and use IPSec to secure transport	5	8	38
7	Software bugs and defects in management station which enables unauthorized access to the system	6	Isolate management station from other network and utilize patching process	8	4	33
8	Man in the middle attack on communication path between management station and controller	5	Disable mac learning on management network switches and use static mac entries. Secure management network physically.	5	5	25
9	Man in the middle attack on communication path between controller and OpenFlow switch	3	Disable mac learning on controller network switches and use static mac entries. Secure controller network physically.	5	5	25

Legends:
TV = Threat Vector
S = Scenario
*) SO = Security Objectives
**) Pr = Probability

3.4.1 Risk distribution

Each scenario has unique technical impact factors and vulnerability factors. The graph in figure 5 represents relationship between those two factors in this evaluation before security enhancements. As we can see, scenarios are distributed quite evenly to the range. High risk scenarios have high impact and probability values. Top six scenarios with highest risk value are marked with red square.

The SEF framework gave us a result which indicates that security of given architecture was not adequate. There were many scenarios which contains so high probability value that it is statistically almost sure that such scenario is realized during the lifetime of the network. After evaluating the security of proposed architecture, mitigation patterns were applied to the model. In next section patters are described and results analysed.

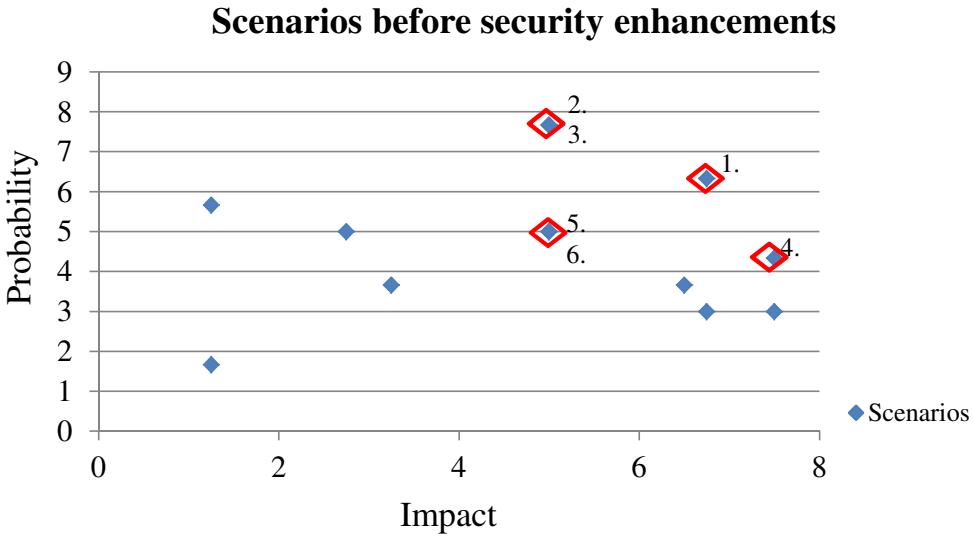

Figure 5: Comparing scenarios with probability and impact before security enhancements

3.4.2 Threat mitigation patterns

There are alternative ways to mitigate risk of each scenario. Top six risk scenarios are listed in table 4 after applying mitigation patterns. Full list can be seen in appendix B. Scenario numbers are referring to the table in appendix.

Abuse of listener mode found to be most critical security risk (scenario 6). This issue is valid only for some OpenFlow enabled switches from vendor Hewlett-Packard (HP). This risk is easy to mitigate by disabling listener mode configuration in switches [26]. This is not OpenFlow specification related issue but vendor specific implementation issue instead. Important here is that other vendors should be precautious when implementing this kind of features. Feature should be disabled by default in device configuration.

Lack of mandatory encryption and proper authentication were found to be second and third critical security issues (scenarios 4 and 5). All OpenFlow implementations should consider securing switch to controller communication path using SSL [15]. Another method proposed by the author of this study is to use IPSec if available. Communication channel encryption is OpenFlow specification related issue. ONF should focus more on these security related items and include clear guidance how vendors should implement control plane security.

Table 4: The Full Security Table after enhancements

				Security Objectives *)					Probability **)				
S	**Threat**	**TV**	**Pattern**	**IC**	**II**	**IAV**	**IAU**	**Total**	**EOD**	**EOE**	**Pub**	**Total**	**Risk**
7	Software bugs and defects in management station which enables unauthorized access to the system	6	Isolate management station from other network and utilize patching process	7	7	7	9	8	3	3	3	3	23
2	Software bugs in OpenFlow client which enables vulnerability	2	Enhance software quality with sophisticated testing automation and simulation driven methods	7	6	6	7	7	3	3	3	3	20
8	Man in the middle attack on communication path between management station and controller	5	Disable mac learning on management network switches and use static mac entries. Secure management network physically.	6	6	1	7	5	3	3	3	3	15
9	Man in the middle attack on communication path between controller and OpenFlow switch	3	Disable mac learning on controller network switches and use static mac entries. Secure controller network physically.	6	6	1	7	5	3	3	3	3	15
1	Software bugs in controller which enables vulnerability	4	Enhance software quality assurance process with standardized testing and simulation driven methods	9	7	7	7	8	1	3	1	2	13
13	Vulnerability on controller interface enables attack on control plane communication	3	Utilize real-time flow rule analysis software	7	6	7	7	7	1	3	1	2	11

Legends:

TV = Threat Vector S = Scenario

*) IC = Impact on Confidentiality
II = Impact on Integrity
IAV = Impact on Availability
IAU = Impact on Authenticity

**) EOD = Easy of Discovery
EOE = Easy of Exploit
Pub = Publicity

Security of management station was found to be fourth critical (scenario 7). Software bugs and defects provides back-door for an attacker to the management network. From management station the attacker can connect to the controller. This study recommend that management station must be isolated from other network. Also all unauthorized access from management network to outside should be blocked and logged. Furthermore, patching process must be defined and implemented to ensure that all vulnerabilities are mitigated as soon as those are discovered. This is OpenFlow implementation specific issue.

MiM attacks on control communication path between both switch to controller and management station to controller was found to be fifth critical risk (scenarios 8 and 9). There are several ways to secure these communication paths to be more resilient for MiM attacks. This study recommend that MAC address learning should be disabled from switches and use static MAC entries in the network between management station and controllers. This effectively lowers the possibility of traditional MiM attack. It is also important to isolate control and management networks physically so that attacker cannot have an physical access to devices or cables. This is considered as OpenFlow implementation specific issue.

Software bug in OpenFlow client or in controller is high risk issue (scenarios 2 and 1). By utilizing proper software release process with QA control and au-

tomated testing procedures, software communities can achieve enhanced software quality. Furthermore, simulation driven testing can be used for exclude all known vulnerabilities before releasing software. This is not OpenFlow specification related issue but issue that organization should consider.

Controller vulnerability related risk can be lowered using real-time flow rule analysis software (scenario 13). VeriFlow is one of those as presented in chapter 3.1.7. VeriFlow does not provide full protection against non-authoritative rule insertions but it protects against mathematically predictable patterns in pre-definable domain.

There are some areas where current OpenFlow specification is ambiguous (scenario 12). This raises a possibility for interoperability problems in multi-vendor environment. One of these situations is when an OpenFlow switch receives a malformed packet. First way to dampen impact is to utilize rate limiting packets destined to controller. This effectively dampens impact of possible attack done using a malformed packets. This is also controller software quality issue. All controller applications should survive and be resilient against this kind of risk.

Another case where rate-limiting can be used for dampening attacks is where switch receives more traffic than it can handle (scenario 11). Controller application should be aware of hardware constrains and be able to insert rate-limit rules accordingly. Rule insertion, and rate-limiting, can be done on the edge of the network where traffic is originating.

Information disclosure of OpenFlow network occurs when attacker probes the network by sending different kind packets and using for example flow initiation time, jitter time and latency time information to picture the structure of the network (scenario 10). One way to reduce the difference between flows first packet and subsequent packets is to ensure that controller capacity is adequate [17]. Controller should also be closely located from OpenFlow switch so that packet can be passed as fast as possible to controller. Another way to reduce possibility of information disclosure is to use explicit rule insertion where controller predefines all flows beforehand. This approach is rarely feasible due to flow table size constrains.

Flow insertion or removal can fail for some reason from time to time (scenario 3). Software bugs, control channel temporal distortions or hardware malfunction causes that controller and switch states are not synchronized. This study recommend, as a long term solution, to include flow table synchronization feature to OpenFlow specification where a switch and controller can initiate table check and update the table if differences. Short term solution is to use flow rules with short lifetimes. Downside of this latter method is that it increases control traffic and yields more jitter to the flows.

3.4.3 Re-evaluated scenarios

After applying previously listed patterns to the architecture the security evaluation process was re-executed. The graph in figure 6 represents relationship between probability and impact after applying security enhanced patterns to the architecture. By comparing figures 5 and 6 we can see that probability of of high risk scenarios have reduced. Thus, we can state that system architecture is more secure after

applying risk mitigation patterns. After applying these enhancements, there is still issues with medium risk. Further study and analysis is needed to enhance the architecture even more.

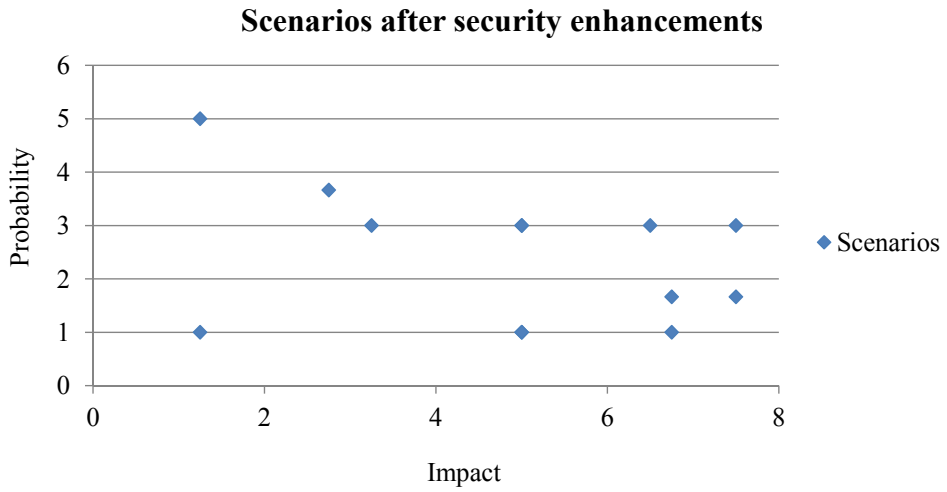

Figure 6: Comparing scenarios with probability and impact after security enhancements

Earlier work on the field was presented in this chapter. After that was results compared against theoretical frameworks and evaluated using SEF model. Next we move to the results and recommendations part of the study.

4 Results and recommendation

In the previous chapter we went through some of the earlier research done so far and carried out the actual security evaluation. Results pointed out that there are still much to improve in security before OpenFlow networks could be implemented without significant risks. Some of the major risk enablers and attack types are shown in the figure 7. There is practically none part of the architecture that could dodge a threat. In this section, paramount view is offered to the security improvements in studied research reports. Finally we also present some new results that patch up some of the observed loopholes.

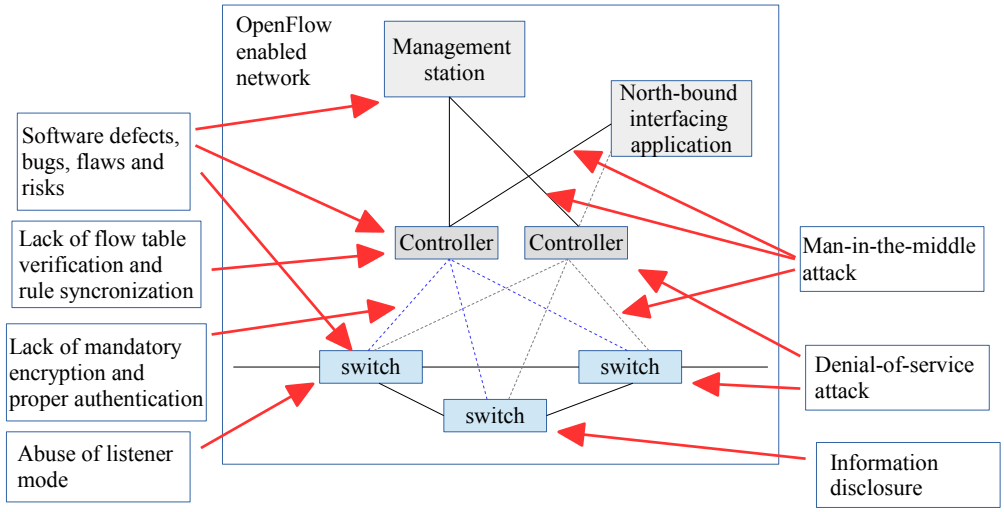

Figure 7: Threats and attack types in OpenFlow network

4.1 Implementation recommendations

During this study many observations were made that were related to implementation level. Here are listed these recommendations.

Current OpenFlow specification does not state which version of SSL/TLS should be used for switch to controller interface. It is known that TLS version 1.0 is vulnerable [22]. Thus, it is recommended that TLS version 1.1 should be used.

Some OpenFlow switch models have a debug or listener mode that can be used without authentication [27]. It might be enabled by default, it should be disabled

in production network.

One option to make network more stable would be to implement a fallback controller to the switch that could be used if primary controller fails. This fallback controller would include small number of essential flow rules. To prevent basic ARP spoofing in OpenFlow enabled network switch should include and enable prevention mechanisms in place by default. [16]

Controller network should be isolated from actual user traffic as well as possible. There should be no backdoors from external network to the controller segment. The need for physical separation needs to be concluded on case by case basis.

Administrative stations should be patched with high priority to minimize risk for vulnerabilities. Administrative stations should also be hardened by using security best practices [28]. All connections from administrative stations should be blocked to the external networks so that possible attacker is not able to retrieve any additional malware or viruses to the system. All secret keys should be kept in secure storage device to prevent a data theft which could possible used for attack.

To lower the possibility of ARP spoofing and, thus, a man-in-the-middle attacks, it is recommended that static MAC and ARP entries are used in controller networks' switches and routers. If management network is also SDN/OpenFlow enabled, separate controller should be dedicated for that purpose.

4.2 New requirements for OpenFlow specification

OpenFlow specification is continuously evolving industrial standard. ONF has been releasing new revisions every three to six months. During this study, some new requirements were found that are recommended to be included in specification in the future.

First, control communication channel encryption should be made mandatory. Also clear technical minimal level should be defined to exclude technologies which are known to be vulnerable. Currently TLS 1.1 sets the minimum standard. To enable uniform implementation of certificate based authentication, specification should include clear guidance how it should be implemented. Authentication should be mandatory.

Second, current OpenFlow specification does not provide any message capability for flow table rule synchronisation between controller and a switch. If erroneously some rules are not removed in the table, e.g. after the network outage or software bug, it is evident that controller and switch rules are getting out of synchronisation. Long term solution would be to include a flow table checksum and flow table synchronisation message features to the OpenFlow specification.

Third, OpenFlow version 1.4.0 specification explicitly states that it does not define what is the expected behaviour when controller or switch receives a malformed or corrupted packet. It is known that these packets are causing problems for many current controller software [17]. OpenFlow specification should define how controller and switch should handle such packets. As long as OpenFlow specification does not give guidance, the expected behaviour should be that the packet is ignored, logged and rate-limited.

4.3 Other considerations

When using flow-rule verification tools, like VeriFlow, the flow initiation time raises due to added computing need. [18] This grows the probability of information disclosure (threat vector 9).

OpenFlow switch platforms that are using general purpose CPU architecture can perform faster software version updates from lower OpenFlow version to higher. When OpenFlow flow handling is hard coded to the hardware in form of ASIC chips the OpenFlow version upgrade is not so straightforward. This yields slowness to the protocol version upgrades and, thus, slowness to the introduction of security related features to the network. If ASIC chips are used, those should include feature for re-programmable firmware. This item is worth for further research.

Another observation was that by splitting controllers into multiple instances it yields much inter-controller traffic and, thus, brings new scenario to evaluate.

From quality of OpenFlow specification point of view we can also state that Open Networking Foundation should consider more close relationship with other industry standardization bodies. For example IETF and ETSI have strong background of security related specifications.

Now we have presented results and recommendations of this study. It was clear that current level of OpenFlow security is not on sufficient level. With proper security planning and risk mitigation patterns, security can be enhanced to bearable level. Next, we conclude this study by presenting a summary.

5 Summary

OpenFlow as a concept has become a viable alternative to accomplish modern and cost-effective network environment. Especially data centres and carrier networks are gaining by enabling high level of automation, link load balancing and centralized control. However, programmability brings new security related problems and threats.

OpenFlow is a concept which includes architecture and communication protocol definitions released by Open Networking Foundation. OpenFlow architecture covers infrastructure and control layer part of the generic SDN architecture. OFP is well defined as a protocol and it is used between OpenFlow switches and controllers. It includes capabilities to function in synchronous or asynchronous mode, in TCP or UDP based transport layer and protocol also provides option for controller redundancy. Specification does not cover topics that are not directly part of the protocol itself, for example definition how session authentication and encryption should be done.

OpenFlow based network service development is moving towards traditional software development. All software applications are prone to vulnerabilities, bugs, flaws and, thus, risks. To minimize probability and impacts of these problems, applications and architectures should be evaluated to enhance security.

The software evaluation framework is scenario based assessment method. It builds on profiles that represents different kind of threat vectors to the system. These profiles are prioritized by rating occurrence probabilities and impact factors using experts opinion. For each profile, mitigation patterns are planned carefully to decrease either probability or impact. This process is repeated iteratively as long as required security and risk level is archived.

During this assessment security evaluation framework was used. Total of 13 individual threat profiles were identified and evaluated. Defined patterns were found to be effective on most of the cases. Overall security could be enhanced if proposed patterns are implemented.

Most of the problems can be avoided at the first place if design is evaluated properly with security in mind. Importance of QA process should not be under estimated. Most of the bugs in production system can and should be avoided by utilizing appropriate code verification, testing and analysis tools. Though SDN and OpenFlow brings totally new flexibility and programmability for the network services it should be remembered that same basic problems and rules are applying that are found from traditional software and network systems.

OpenFlow network facing application issues were scoped out from this study. This is important field and it is worth for further study. Another area is practical security assessment of controller softwares. There are studies which are showing that there are severe security and stability problems in publicly used controllers.

References

[1] Christos Douligeris and Dimitrios Nikolaou Serpanos. *Network Security: Current Status and Future Directions*. Onl. Piscataway, NJ, USA: IEEE Press, 2007. ISBN: 9780470099742.

[2] Mark Dowson. "The Ariane 5 Software Failure". In: *SIGSOFT Softw. Eng. Notes* 22.2 (Mar. 1997), pp. 84–. ISSN: 0163-5948. DOI: 10.1145/251880.251992. URL: http://doi.acm.org/10.1145/251880.251992.

[3] Gary McGraw. *Software Security: Building Security In*. Pearson Education, Inc., 2006, pp. 9–14. ISBN: 0-321-35670-5.

[4] Norman Fenton, Paul Krause, and Martin Neil. "A probabilistic model for software defect prediction". In: *IEEE Trans Software Eng* (2001).

[5] A. Alkussayer and W.H. Allen. "A scenario-based framework for the security evaluation of software architecture". In: *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*. Vol. 5. July 2010, pp. 687–695. DOI: 10.1109/ICCSIT.2010.5564015.

[6] *OWASP Risk Rating Methodology*. 2014. URL: https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology (visited on 08/24/2014).

[7] Diego Kreutz, Fernando M.V. Ramos, and Paulo Verissimo. "Towards Secure and Dependable Software-defined Networks". In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. HotSDN '13. Hong Kong, China: ACM, 2013, pp. 55–60. ISBN: 978-1-4503-2178-5. DOI: 10.1145/2491185.2491199. URL: http://doi.acm.org/10.1145/2491185.2491199.

[8] Bruce Schneier. *Applied Cryptography Second Edition: protocols, algorithms, and source code in C*. John Wiley & Sons, Inc., 1996. ISBN: 0-471-11709-9.

[9] M. Handley and E. Rescorla. *Internet Denial-of-Service Considerations*. RFC 4732. RFC Editor, Nov. 2006. URL: http://www.rfc-editor.org/rfc/rfc4732.txt.

[10] M. Al-Hemairy, S. Amin, and Z. Trabelsi. "Towards more sophisticated ARP Spoofing detection/prevention systems in LAN networks". In: *Current Trends in Information Technology (CTIT), 2009 International Conference on the*. Dec. 2009, pp. 1–6. DOI: 10.1109/CTIT.2009.5423112.

[11] R. Kloti, V. Kotronis, and P. Smith. "OpenFlow: A security analysis". In: *Network Protocols (ICNP), 2013 21st IEEE International Conference on*. Oct. 2013, pp. 1–6. DOI: 10.1109/ICNP.2013.6733671.

[12] *OpenFlow Switch Specification version 1.4.0*. California, USA, 2013. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf (visited on 03/08/2014).

[13] *Software-Defined Networking: The New Norm for Networks, OpenFlow White Paper*. 2012. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf (visited on 07/29/2014).

[14] S. Sezer et al. "Are we ready for SDN? Implementation challenges for software-defined networks". In: *Communications Magazine, IEEE* 51.7 (July 2013), pp. 36–43. ISSN: 0163-6804. DOI: 10.1109/MCOM.2013.6553676.

[15] Kevin Benton, L. Jean Camp, and Chris Small. "OpenFlow Vulnerability Assessment". In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. HotSDN '13. Hong Kong, China: ACM, 2013, pp. 151–152. ISBN: 978-1-4503-2178-5. DOI: 10.1145/2491185.2491222. URL: http://doi.acm.org/10.1145/2491185.2491222.

[16] D. Romao et al. *Practical Security Analysis of OpenFlow*. 2013. URL: https://os3.nl/_media/2013-2014/courses/ssn/projects/practical_security_analysis_of_openflow_report.pdf (visited on 08/13/2014).

[17] Alexander Shalimov et al. "Advanced Study of SDN/OpenFlow Controllers". In: *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*. CEE-SECR '13. Moscow, Russia: ACM, 2013, 1:1–1:6. ISBN: 978-1-4503-2641-4. DOI: 10.1145/2556610.2556621. URL: http://doi.acm.org/10.1145/2556610.2556621.

[18] Ahmed Khurshid et al. "Veriflow: Verifying Network-wide Invariants in Real Time". In: *SIGCOMM Comput. Commun. Rev.* 42.4 (Sept. 2012), pp. 467–472. ISSN: 0146-4833. DOI: 10.1145/2377677.2377766. URL: http://doi.acm.org/10.1145/2377677.2377766.

[19] *Cisco Nexus 7000 Series Supervisor Module Data Sheet*. Tech. rep. 2013. URL: http://www.cisco.com/c/en/us/products/collateral/switches/nexus-7000-series-switches/Data_Sheet_C78-437758.pdf (visited on 08/17/2014).

[20] *Cisco Nexus 5000 Series Architecture: The Building Blocks of the Unified Fabric*. Tech. rep. 2009. URL: http://www.cisco.com/c/en/us/products/collateral/switches/nexus-5020-switch/white_paper_c11-462176.pdf (visited on 08/17/2014).

[21] Jeremy M Dover. *A denial of service attack against the Open Floodlight SDN controller*. Tech. rep. URL: http://dovernetworks.com/wp-content/uploads/2013/12/OpenFloodlight-12302013.pdf (visited on 08/17/2014).

[22] N.J. Al Fardan and K.G. Paterson. "Lucky Thirteen: Breaking the TLS and DTLS Record Protocols". In: *Security and Privacy (SP), 2013 IEEE Symposium on*. May 2013, pp. 526–540. DOI: 10.1109/SP.2013.42.

[23] Marco Canini et al. "A NICE Way to Test OpenFlow Applications." In: *NSDI*. 2012, pp. 127–140.

[24] Haohui Mai et al. “Debugging the Data Plane with Anteater”. In: *Proceedings of the ACM SIGCOMM 2011 Conference*. SIGCOMM ’11. Toronto, Ontario, Canada: ACM, 2011, pp. 290–301. ISBN: 978-1-4503-0797-0. DOI: 10.1145/2018436.2018470. URL: http://doi.acm.org/10.1145/2018436.2018470.

[25] S. Hommes et al. “Automated source code extension for debugging of OpenFlow based networks”. In: *Network and Service Management (CNSM), 2013 9th International Conference on*. Oct. 2013, pp. 105–108. DOI: 10.1109/CNSM.2013.6727816.

[26] *Production-ready SDN with OpenFlow 1.3*. Tech. rep. Hewlett-Packard, 2013. URL: http://h17007.www1.hp.com/docs/interop/2013/37958_HPN_Openflow_Brief_042913_lo.pdf (visited on 03/08/2014).

[27] *HP Switch Software: OpenFlow Supplement*. Tech. rep. 2012. URL: http://h20566.www2.hp.com/portal/site/hpsc/template.BINARYPORTLET/public/kb/docDisplay/resource.process/?spf_p.tpst=kbDocDisplay_ws_BI&spf_p.rid_kbDocDisplay=docDisplayResURL&javax.portlet.begCacheTok=com.vignette.cachetoken&spf_p.rst_kbDocDisplay=wsrp-resourceState=docId%3Demr_na-c03170243-3%7CdocLocale%3D&javax.portlet.endCacheTok=com.vignette.cachetoken (visited on 08/07/2014).

[28] Andrés Steven et al. *Security Sage’s Guide to Hardening the Network Infrastructure*. First edition. Massachusetts, USA: ´Syngress Publishing Inc., 2004. ISBN: 1-931836-01-9.

A Full Security Table before security enhancements

In table 5 is described all evaluated scenarios their security requirements and calculated risks. Also detailed security objectives and probability values are shown.

Table 5: The Full Security Table before security enhancements

				Security Objectives *)					Probability **)				
S	**Threat**	**TV**	**Pattern**	**IC**	**II**	**IAV**	**IAU**	**Total**	**EOD**	**EOE**	**Pub**	**Total**	**Risk**
1	Software bugs in controller which enables vulnerability	4	Enhance software quality assurance process with standardized testing and simulation driven methods	9	7	7	7	8	3	3	3	3	23
2	Software bugs in OpenFlow client which enables vulnerability	2	Enhance software quality with sophisticated testing automation and simulation driven methods	7	6	6	7	7	3	5	3	4	24
3	Lack of flow table verification and rule syncronization between controller and a switch	8	Use flow rules with short lifetime	2	1	1	1	1	1	3	1	2	2
4	Lack of mandatory encryption in OpenFlow protocol	3	Secure switch to controller network and use IPSec to secure transport	6	6	1	7	5	7	9	7	8	38
5	Lack of proper authentication in OpenFlow protocol	3	Secure switch to controller network and use IPSec to secure transport	6	6	1	7	5	7	9	7	8	38
6	Abuse of switch listener mode	2	Disable switch listener mode	7	7	6	7	7	7	9	3	6	43
7	Software bugs and defects in management station which enables unauthorized access to the system	6	Isolate management station from other network and utilize patching process	7	7	7	9	8	7	3	3	4	33
8	Man in the middle attack on communication path between management station and controller	5	Disable mac learning on management network switches and use static mac entries. Secure management network physically.	6	6	1	7	5	3	9	3	5	25
9	Man in the middle attack on communication path between controller and OpenFlow switch	3	Disable mac learning on controller network switches and use static mac entries. Secure controller network physically.	6	6	1	7	5	3	9	3	5	25
10	Information disclosure of OpenFlow network	9	Ensure sufficient controller capacity with redundant controllers to keep session initalization time minimum	2	1	1	1	1	9	1	7	6	7
11	Hardware constrains enabled DoS threat	1	Utilize rate limiting on switches for packets destined to slow-path	2	1	9	1	3	3	5	3	4	12
12	Interoperability problems due to insufficient OpenFlow specification of how to handle malformed or corrupted packets	1	Utilize rate limiting on switches for packets destined to slow-path. Implement Packet-In filter in controller to detect malformed packet properly.	2	1	7	1	3	3	9	3	5	14
13	Vulnerability on controller interface enables attack on control plane communication	3	Utilize real-time flow rule analysis software	7	6	7	7	7	3	3	3	3	20

Legends:

TV = Threat Vector S = Scenario

*) IC = Impact on Confidentiality
II = Impact on Integrity
IAV = Impact on Availability
IAU = Impact on Authenticity

**) EOD = Easy of Discovery
EOE = Easy of Exploit
Pub = Publicity

B Full Security Table after security enhancements

In table 6 is described all evaluated scenarios their security requirements and calculated risks after applying risk mitigation patterns. Also detailed security objectives and probability values are shown.

Table 6: The Full Security Table after security enhancements

				Security Objectives *)					Probability **)				
S	**Threat**	**TV**	**Pattern**	**IC**	**II**	**IAV**	**IAU**	**Total**	**EOD**	**EOE**	**Pub**	**Total**	**Risk**
7	Software bugs and defects in management station which enables unauthorized access to the system	6	Isolate management station from other network and utilize patching process	7	7	7	9	8	3	3	3	3	23
2	Software bugs in OpenFlow client which enables vulnerability	2	Enhance software quality with sophisticated testing automation and simulation driven methods	7	6	6	7	7	3	3	3	3	20
8	Man in the middle attack on communication path between management station and controller	5	Disable mac learning on management network switches and use static mac entries. Secure management network physically.	6	6	1	7	5	3	3	3	3	15
9	Man in the middle attack on communication path between controller and OpenFlow switch	3	Disable mac learning on controller network switches and use static mac entries. Secure controller network physically.	6	6	1	7	5	3	3	3	3	15
1	Software bugs in controller which enables vulnerability	4	Enhance software quality assurance process with standardized testing and simulation driven methods	9	7	7	7	8	1	3	1	2	13
13	Vulnerability on controller interface enables attack on control plane communication	3	Utilize real-time flow rule analysis software	7	6	7	7	7	1	3	1	2	11
12	Interoperability problems due to insufficient OpenFlow specification of how to handle malformed or corrupted packets	1	Utilize rate limiting on switches for packets destined to slow-path. Implement Packet-In filter in controller to detect malformed packet properly.	2	1	7	1	3	3	5	3	4	10
11	Hardware constrains enabled DoS threat	1	Utilize rate limiting on switches for packets destined to slow-path	2	1	9	1	3	3	3	3	3	10
6	Abuse of switch listener mode	2	Disable switch listener mode	7	7	6	7	7	1	1	1	1	7
10	Information disclosure of OpenFlow network	9	Ensure sufficient controller capacity with redundant controllers to keep session initalization time minimum	2	1	1	1	1	7	1	7	5	6
4	Lack of mandatory encryption in OpenFlow protocol	3	Secure switch to controller network and use IPSec to secure transport	6	6	1	7	5	1	1	1	1	5
5	Lack of proper authentication in OpenFlow protocol	3	Secure switch to controller network and use IPSec to secure transport	6	6	1	7	5	1	1	1	1	5
3	Lack of flow table verification and rule syncronization between controller and a switch	8	Use flow rules with short lifetime	2	1	1	1	1	1	1	1	1	1

Legends:

TV = Threat Vector S = Scenario

*) IC = Impact on Confidentiality
II = Impact on Integrity
IAV = Impact on Availability
IAU = Impact on Authenticity

**) EOD = Easy of Discovery
EOE = Easy of Exploit
Pub = Publicity