UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

SECURITY ANALYSIS OF ONOS SOFTWARE-DEFINED NETWORK PLATFORM

Technical Report

OLUWADAMILOLA ADENUGA-TAIWO
Faculty of Business and Information Technology
University of Ontario Institute of Technology
Oshawa, Canada
Oluwadamilola.adenugataiwo@uoit.ca

SHAHRAM SHAH HEYDARI
Faculty of Business and Information Technology
University of Ontario Institute of Technology
Oshawa, Canada
Shahram.Heydari@uoit.ca

SUMMARY

Software Defined Networking (SDN) enables organizations strengthen their network architectures, reduce running costs and enable sophisticated network functions. The adoption of software defined networks by top industry players like CISCO, IBM and Ericsson implies its increased application in mainstream networks and real world applications. Security challenges in SDN networks are quite similar to the traditional networks where many attacks occur at the control layer. This issue is further escalated by the fact that information is synchronized between the data layer, which houses the network devices; and the control layer, in which the SDN controller operates. The control layer contains policies for operating the data layer, providing a single point of failure on the network as a whole. This technical report examines a number of security challenges within the control layer of ONOS SDN platform – an open source initiative under the Linux foundation for carrier-grade software-defined networking, which is quickly gaining popularity within the industry. In particular, we examine ONOS vulnerability Northbound and Southbound DOS and MITM attacks in a test environment and provide observation, analysis and some defensive measures.

I. INTRODUCTION

The major characteristic of the Software Defined Networks (SDN) is the physical separation of the control plane from the forwarding plane. The centralized control function maintains the network state and provides forwarding instructions to the data plane. In recent years, SDN has received significant attention from both the academic community and the industry as it offers "flexibility of implementing network infrastructure, programmability of controlling network device, and the reliability of providing network service (Yu-Jia Chen et al., 2014). SDN found its major application with the arrival of cloud computing and virtualization technologies in the data center (Sandra Scott- Hayward et al, 2016).

Openflow is the first standard communications interface defined between the control and forwarding layers of an SDN architecture. It allows access to the forwarding layer, which contains devices such as switches and routers, on both physical and virtual infrastructures (ONF, 2014). "The Openflow controller sends commands to manipulate all Openflow switches, maintains network topology information, and monitors the overall status of entire network" (Yu-Jia Chen et al, 2014). The successful adoption of Openflow by both researchers and industry is a major drive for the SDN movement. SDN-enabled networks have been successfully deployed in various scenarios (e.g., Google's backbone network, Microsoft's public cloud etc.). In addition, network virtualization software has significantly progressed from trial evaluations to production deployments (e.g., VMWare NSX) (Jafar Haadi Jafarian et al, 2012).

One of the major contributions or drivers for the innovation of the SDN technology is the difficulty of large data centers to support the dynamic nature and requirements of server virtualization. This implies the ability to effectively migrate new virtual machines into a large network and apply network services to them quickly enough. By separating the Network control from forwarding functions, SDN provides improved solutions to traditional problems in networking, most especially, routing. Additionally, SDN offers more benefits such as

- Improved network visibility and reduced operational due to some of the network management tools today being more difficult to use, resulting in higher operational costs
- The ability to use open application program interfaces to connect the application layer to the network resources.
- Support for network segmentation for specific parts of the network, or departments which would allow other parts of the network to run independently

(Lee Doyle, 2013)

The SDN controller sends commands to manipulate all switches and network devices, maintains the network topology information and also monitors the overall status of the network. The network devices forward incoming network packets according to a flow table, also maintained by the controller.

While these trends are promising, one area that has received lesser attention is that of security challenges in SDN. The SDN architecture offers a lot of advantages to networking domain, but its weaknesses mostly lie in lack of

security. Nonetheless, it offers support for security solutions, which is an area that is currently being explored by many researchers. Some of these challenges or vulnerabilities are discussed below

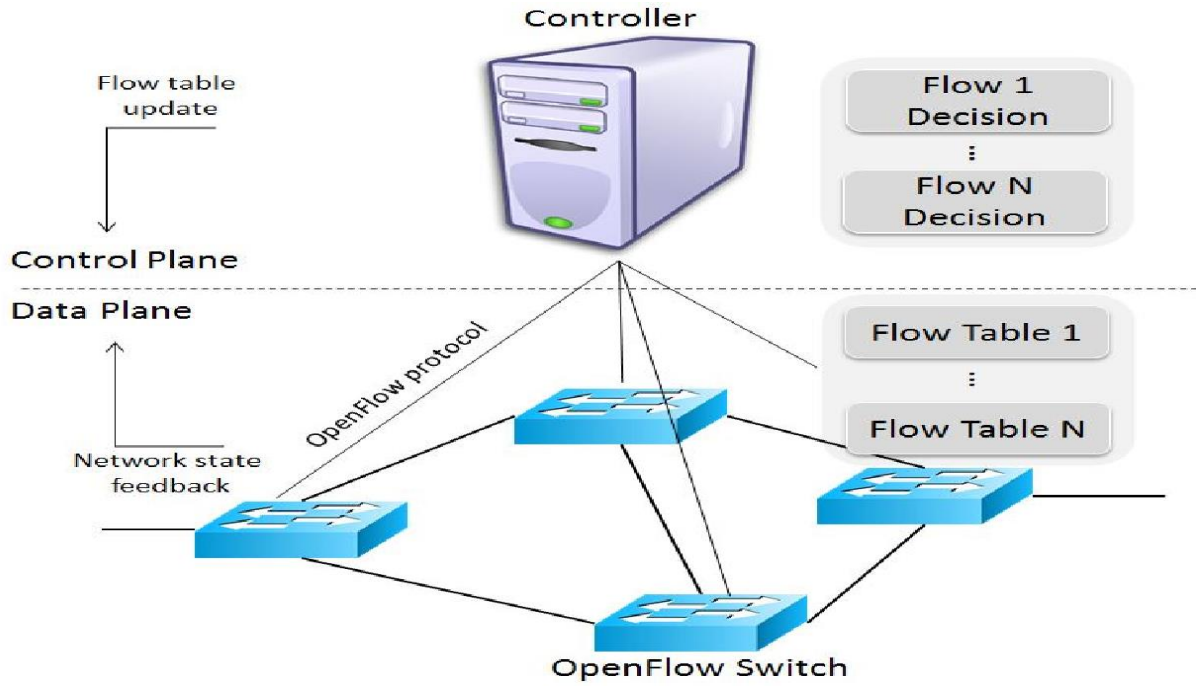

Figure 1: Open flow architecture

1) **Unauthorized Access**: An important characteristic of SDN is in its centralized control, or more accurately described as logically centralized control, which in many implementations is distributed. The functional architecture of SDN makes it possible for several controllers to access the data plane of the network. Similarly, third party apps may link to a pool of controllers. The controller provides an abstraction to applications so that the applications can read/write network state, which grants a level of network control. An attacker could gain access to network resources and manipulate the network operation if it impersonates a controller/application. (Sandra Scott- Hayward et al, 2016)

2) **Data Modification**: The controller is able to program the network devices to control the flow of traffic in the SDN. An attacker can effectively take control over the entire system if it can hijack the controller. From this privileged position, the attacker can insert or modify flow rules in the network devices, which would allow packets to be steered through the network to the attacker's advantage. With regards to the communication channel, the Openflow switch specification (ONF, 2014) describes the use of TLS with mutual authentication between the controllers and their switches. However, it is an optional security feature, and the version of TLS is not specified. The lack of TLS adoption by major vendors can allow a man-in-the-middle attacker to impersonate the controller and launch various attacks; for example, manipulate the Openflow messages sent by the controller or inject reset messages to tear down the connection. A man-in-the-middle attack occurs when the attacker has the ability to intercept messages between two victims to monitor and alter or inject messages into the communication channel, particularly when there is no authentication between the communication endpoints.

3) **Denial of Service (DOS)**: An attacker could flood the controller with packets requiring a flow rule decision and render it unavailable for legitimate users, using the communication path between the controller and the network device. A similar DOS attack could be performed at the infrastructure level with flooding of the flow table for which limited memory resources are available. These Denial of service attacks mostly affect

networks that use reactive rules. Networks that make use of proactive rule insertions have a lesser risk as long as "no traffic can generate arbitrary packet-in events" (Kevin Benton et al, 2013)

4) **Configuration Issues**: As vulnerabilities are detected on a network, network security policies and protocols are continuously developed and many of those policies will apply to the layers and interfaces of the SDN framework. However, there is reduced protection from such policies if they are not implemented or they are disabled without understanding the security intricacies of the deployment scenario. In an SDN-based network, it will be beneficial for network operators to implement security policies such as TLS. Incorrect use of security features due to misconfigurations can impact all layers in the architecture. By disabling a secure connection, network functions can be negatively affected when several potential vulnerabilities are introduced in the framework. With SDNs, networks are easily programmable, which allow for the creation of dynamic flow policies. "It is, in fact, this advantage that may also lead to security vulnerabilities. With policies from multiple applications or installed across multiple devices, inconsistencies must be detected to resolve policy conflicts." (ONF, 2014)

5) **Controller Vulnerabilities:** In a SDN environment, the control plane can dynamically enforce flow rules when the data plane requires, which enables efficient control of the network. As mentioned above in the DOS section, this type of reactive-mode control can cause serious problems when there are too many requests from the data plane to the control plane. Requests from the data plane can flood the control plane. In addition, the data plane flow tables can also be flooded by rules for handling the requests. (Seungwon Shin et al, 2013)

6) Implementing mechanisms like TLS could increase the security of the messages between the controllers and the switches, however, it wouldn't help detecting switches that incorrectly alter rules. Also, tracking the flow table state changes for each switch by recording the 'flow-removed' messages, requires extra processing on the controller, especially when there temporary network outage (Kevin Benton et al, 2013). A network outage could be caused by this mismatch between the controller's version of the network rule-state and the actual rule-state. The only countermeasure is by inspecting all the flow tables from each switch, which could be computationally intensive for the controllers and the switches.

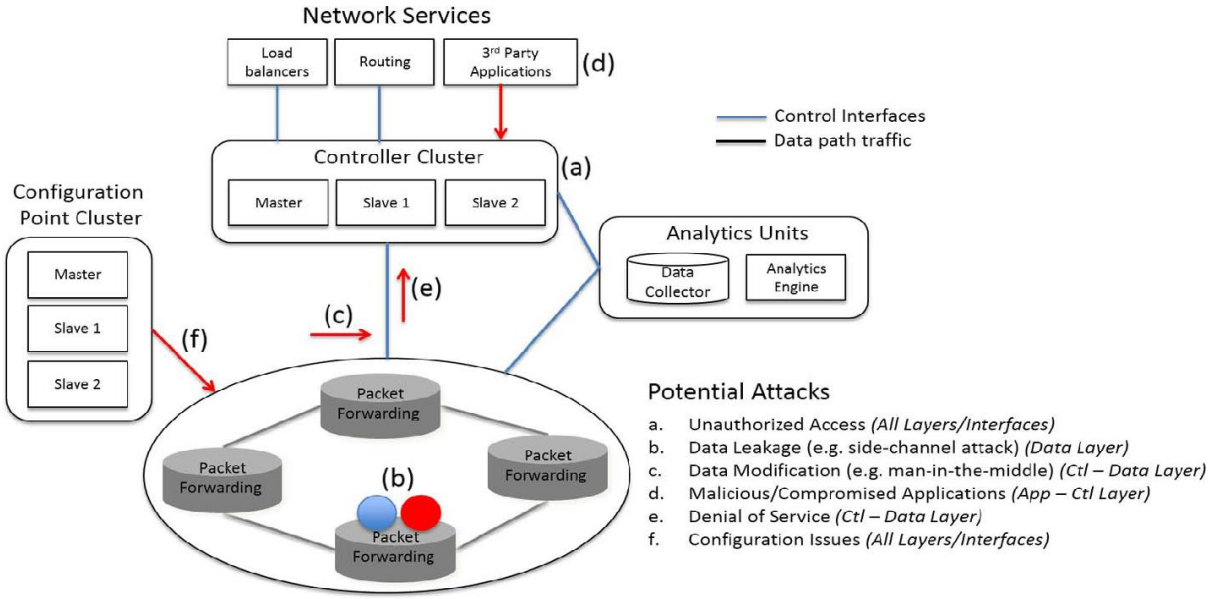

Figure 2. Potential SDN attacks and vulnerabilities

(Sandra Scott- Hayward et al, 2016)

The tenets of a secure communications network are the confidentiality, integrity and availability of information, which are supported by authorization, accounting and encryption techniques. The combination of these properties imply a network or IT assets being protected from malicious attack or intentional damage. Some of the challenges and vulnerabilities in a SDN-enabled network have been explored in this section, and this reveals how critical it is to include security in the design of an SDN infrastructure.

Another problem currently existing in software-defined network devices is the operating system compatibility, as some vendors make use of proprietary operating systems which may not function on other networking devices. The Open Networking Operating System (ONOS) is the result of an effort to address this problem. It is an open source utility hosted by the Linux foundation, and the primary purpose is to have a SDN operating system for vendors or service providers that offers high network scalability, performance and uptime

II. LITERATURE REVIEW

A. Open network operating system

ONOS is a project motivated by the requirements of a large operator network. The paper by (Pankaj Berde et al, 2014) discusses the experiences in building ONOS. They examined two ONOS prototypes, one which implemented 'a distributed, but logically centralized, global network view; scale-out and fault tolerance' (Pankaj Berde et al, 2014) and the second focused on improving overall performance.

As for the security of ONOS, (Scott-Hayward, 2015) describes the operations of the ONOS controller, with regards to authentication, authorization and accounting (AAA). ONOS does not explicitly implement AAA functionality, 'the applications will have to register with a *core service* for a unique App ID to use ONOS services' (Scott-Hayward, 2015). This paper also offers some security recommendations for the ONOS controller, such as applying secure controller settings, designing future controllers considering software security principles, e.g. encryption.

B. Openflow vulnerabilities

Openflow is the main communications protocol responsible for arbitrating the flow of information and control functions between different SDN controllers and the forwarding plane (network devices). (Jafar Haadi Jafarian et al, 2012) explores the challenges of 'defining a security mediation layer between the Openflow application layer and the data plane'. The authors designed a security enforcement service which is embedded in the control layer to mediate all flow rules and control protocol requests initiated by Openflow apps in the same network.

Openflow operates using a set of rules issued from the controller, either configured by the administrator, or in an automated manner by the controller itself. The controller is notified when a new data packet arrives at the switch, which prompts it to send a decision to instruct the switch to look for specific matches in the traffic and take a particular action, if there's a match. These set of rules are referred to as **flow rules.** The switch stores these rules in what is regarded as **flow tables.** (Kiran Vemuri, 2014**).** The controller also installs flow-based rules on the switches in two ways:

- **Reactive approach**: This allows the controller application to make dynamic decisions (on the spot) about incoming traffic. Due to the fact that the controller has to install flow-based rules based on events, It has significant overhead (CPU, memory etc.) and high performance overhead (delay), also considering that there is limited communication channel between controller and switches.
- **Proactive approach**: This approach permits the controller to install rules in switches ahead of time for all the flows coming into the switch

(Masoud Moshref et al, 2014)

As for vulnerabilities in the Openflow protocol, the authors in (Kevin Benton et al, 2013) provide a brief overview of the vulnerabilities present in hardware and software vendors deploying Openflow protocol. The original specification for Openflow required that the channel between controllers and switches to be secured using TLS. However, the subsequent specifications made it optional due to cumbersome configuration steps and this created

an incentive for administrators to operate without TLS. In reality, the only controller with full TLS support is the open vSwitch. This situation obviously opens up security holes in Openflow networks and creates an enabling environment for vulnerabilities.

The use of static IP addresses might pose a problem for networks, because it gives attackers a vantage point to remotely scan network and quickly identify targets, which is a precursor to many attacks. Dynamic IP addresses can be used as a solution but they are insufficient to provide proactive mitigation because these IP addresses are traceable. The authors in (Jafar Haadi Jafarian et al, 2012) introduce a moving target technique called the Openflow random host mutation which 'mutates the IP addresses of end-hosts randomly and frequently so that the attacker's premises about the static IP assignment of the network fails' (Jafar Haadi Jafarian et al, 2012). This work showed to an increase in unpredictability and also rapid mutation while attempting to determine the assignment of unused address ranges under multiple test constraints. The authors showed they could use this technique to invalidate the gathering of information by external scanners by up to 99%, and can save up to 90% of network hosts from zero-day worms.

In light of frequent network state and traffic changes, it is challenging to build robust network protection mechanisms, such as firewalls. To address this challenge, the authors in (Hongxin Hu et al, 2014) introduce FLOWGUARD, which is 'a comprehensive framework, to facilitate not only accurate detection but also effective resolution of firewall policy violations in dynamic Openflow-based networks' (Hongxin Hu et al, 2014). It operates by checking network flow paths in a bid to detect violation in policy when network states are updated. They also implemented FLOWGUARD in floodlight and the results favor FLOWGUARD in managing performance overhead to enable real-time monitoring of SDNs.

C. SDN controller security

Research in this area is basically concerned with the overall security of the control layer in the SDN environment.

The extent to which network security management (as related to controllers) is improved in SDN-based networks, is discussed in (Dmitry Ju et al, 2012). The authors categorized their analysis by infrastructure, software stack, and network protocols with the majority of the discussion focused on protocol security, which the authors define as authentication and confidentiality. It was concluded that the security of the switch–controller and controller–controller communication protocols require further work. The security of switch–controller communication has inspired further research. Consideration of controller–controller communication security is less explored and should be an important consideration in any wide deployment of SDN.

Centralized control and network programmability pose new threats, and new responses are required. These were the conclusions given by the research presented in (Diego Kreutz et al, 2013). Out of seven threat vector identified in this study, three were specific to SDN and relate to controller software, control-data interface and the control application interface. The authors proposed a number of techniques which include the 'replication of controllers and applications to provide alternative management/control in the case of hardware or software faults, diversity of controllers for robustness against software bugs and vulnerabilities in any single controller, and secure components such as trusted computing bases (TCB) to protect the confidentiality of sensitive security data.' (Diego Kreutz et al, 2013).

The vulnerability of centralized controller would be considered a key issue, but other issues of concern are the integration of 3rd party applications with the SDN control layer. The research in (Scott-Hayward, 2015) selects five controllers for security analysis, including OpenDaylight, Open Network operating system, Ryu, which are open-source; and SE-floodlight and ROSEMARY which are research driven and security focused controllers. The paper discusses the extent to which SDN controllers support the set of attributes, namely: security, robustness and resiliency. The results show that none of the above controllers have support for all these features/attributes and that should be a design consideration for future controller developments.

To address the challenge of the lack of key security features on SDN controller, the authors in (Phillip Porras et al, 2015) present a prototype security enhanced version of the popularly used Floodlight controller. This enhancement adds a security-enforcement kernel layer, which has functions directly applicable to other Openflow

controllers. ‘The Security-enforcement Kernel adds a unique set of secure application management features, including an authentication service, role-based authorization, a permission model for mediating all configuration change requests to the data-plane, inline flow-rule conflict resolution, and a security audit service (Phillip Porras et al, 2015). The authors carried out a functionality assessment and performance evaluation of the implementation to demonstrate the robustness and scalability of the system. Floodlight has also been used for different test cases/experiments. The authors in (Vainius Dangovas et al, 2014) used its modules and were able to enhance its architecture to register and authenticate switches to the controller, to authenticate and bind the hosts to the switches, to provide the authentication of users and lastly, manage data flows and user/hosts mobility.

III. PROPOSED METHODOLOGY

The studies above illustrate that SDN can be used for security purposes. However, they only present conceptual ideas or research prototypes. Thus, it is very difficult to understand whether SDN really helps in developing practical security functions. My work begins investigating this issue, and we attempted to discover the effectiveness, feasibility and efficiency of security functions based on SDN techniques.

The open networking foundation (ONF) has identified the southbound communication between the controller and the forwarding devices (e.g. switch) as vulnerable to flow command spoofing alongside several other attacks.

Also, as the OpenFlow specification states, "when using plain TCP, it is recommended to use alternative security measures to prevent eavesdropping, controller impersonation or other attacks on the OpenFlow channel" (ONF, 2014). However, the specification gives no indication of what sort of alternative security measures are needed to prevent those attacks. This is contrary to popular belief that the Openflow protocol is relatively safe.

In this research we examine the ONOS controller to evaluate its security, both on the North and southbound interfaces of the controller. Following the explanations above, the security on the Openflow protocol will be reviewed. We also explore means to enable security on the southbound interface, which utilizes the protocol, if not already enabled.

The overall goal is increase good understanding of functionalities in the ONOS environment, taking into consideration security vulnerabilities, features and enhancements.

IV. SECURITY ANALYSIS OF ONOS CONTROLLER

The test environment consists of the ONOS controller connected to two switches via TCP and SSL southbound connections. Switches are connected to each other using a virtual GRE tunnels and there is a host connected to each switch. All the components are installed on a separate virtual machine instance to create better abstraction for deploying configuration. The figure below is a diagrammatic representation of the set-up.

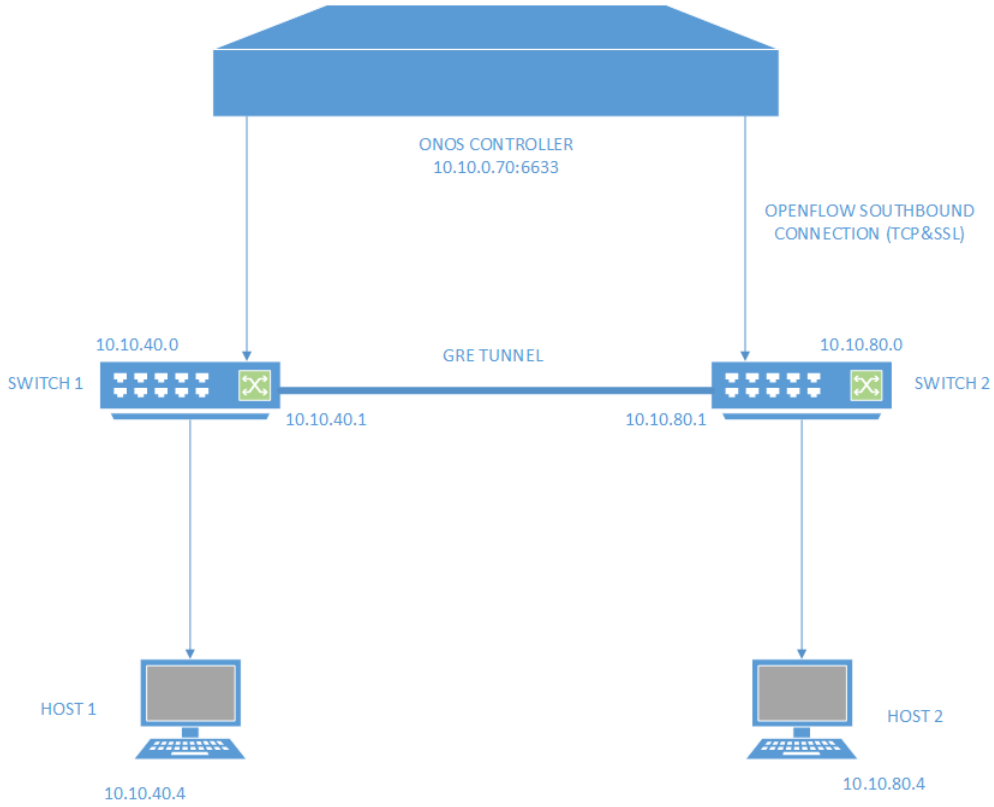

Figure 3: ONOS test environment

The following attack scenarios were investigated in this research:

A. Man in the middle attack

The set-up for this experiment is one ONOS controller embedded in a virtual machine (Ubuntu 16.4) and two switches (All running CentOS 6.5), to simulate an ARP spoofing attack against the controller and a given switch. Basically, this will be an attempt to compromise security whereby the attack machine will transmit forged ARP replies over the local network to link the MAC address of the attack system to the IP address of both the client(switch) and the server(controller), as would be reflected in both their ARP tables. Subsequently, the attack machine will be able to receive messages being transmitted. The aim here is to recover flow messages, authentication information, or other types of information that would be of interest. Since both TCP and SSL connectivity are present between the switch and controller, we attempt the attack in both scenarios and check for any differences or peculiarities.

B. Denial of service attack

We also examine the impact of launching a denial of service attack against the ONOS controller. This scenario has the following setting:

- A distributed denial of service attack whereby we set-up other computers/terminals as botnets to send traffic to the machine running the ONOS in a coordinated manner
- Transmission of malformed or modified packets (with a larger size than the regular packet size) to the controller service, both in an attempt to cause unavailability of the controller. This will make a good case for how the ONOS controller handles large service request quantities or large traffic as the case may be.
- Randomizing the source ports of the packets, which will make the attack harder to defend against in a real world scenario.

C. Experiment Scenarios

1) NORTHBOUND INTERFACE MAN IN THE MIDDLE ATTACK

The experiment carried out a successful man in the middle attack to sniff traffic coming to the ONOS controller northbound interface for application (utilizing the Graphical user interface). Figure 4 shows a remote host attempting to access the ONOS controller GUI

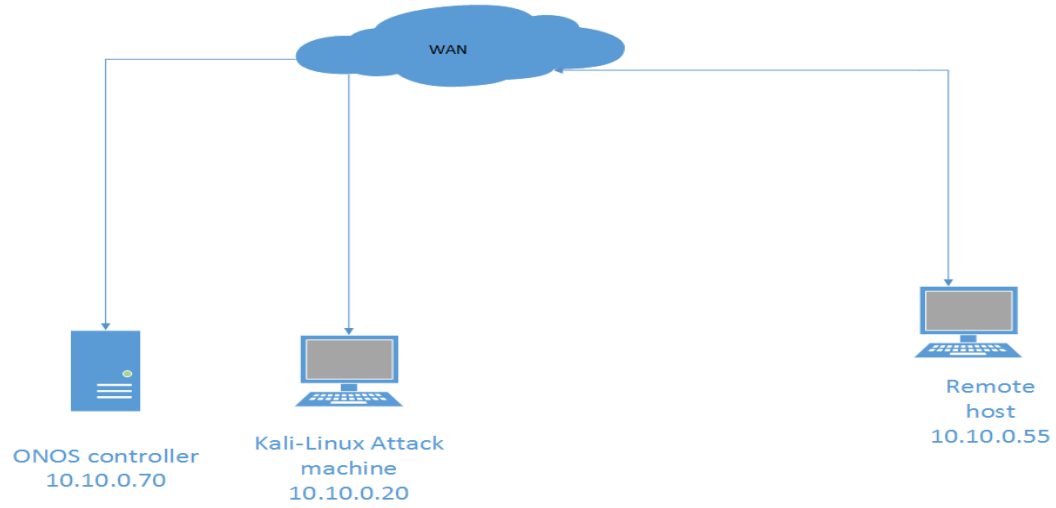

Figure 4. North-bound MITM scenario

- **Remote Machine**

A remote machine was set up, which will act as a system attempting to establish a connection to the GUI listening on port 8181. In a real life scenario, this can be a system administrator attempting to carry out management tasks such as viewing the environment topology, state of devices, viewing and adding new flows, etc.

- **Attacker machine**

In this attack scenario, Ettercap software embedded in Kali-Linux was used. The Ettercap software is mainly used for MITM attacks through different means including ARP spoofing, DNS spoofing etc. ARP spoofing was used for this MITM attack scenario.

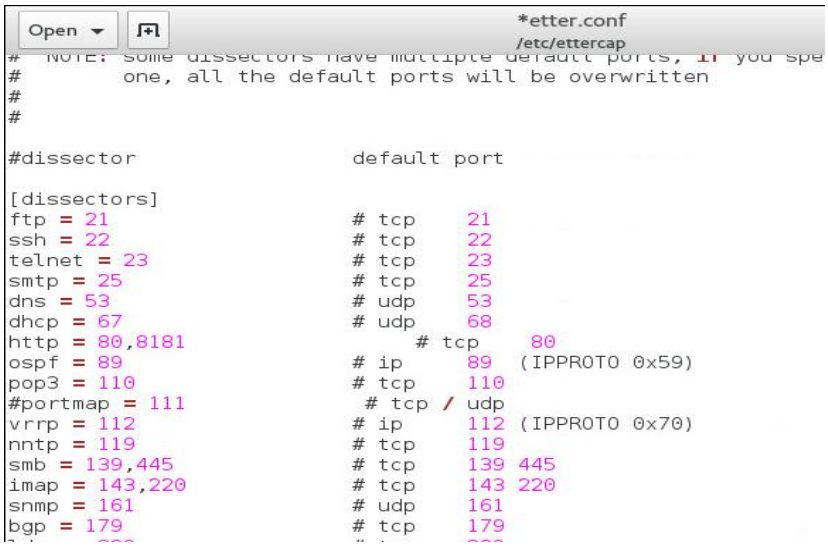

Figure 5. Ettercap port configuration

IP packet forwarding was enabled to allow Ettercap to forward incoming connections from the remote machine to the machine hosting the ONOS controller. Without this provision, as soon as the MITM attack started, the ONOS controller would lose its connection.

Authentication was created for the remote system to access the ONOS GUI, as per Figure 6.

```
Open                                                                    Save
#!/bin/bash
# ----------------------------------------------------------------------------
# Forms ONOS cluster using REST API of each separate instance.
# ----------------------------------------------------------------------------

[ $# -lt 2 ] && echo "usage: $(basename $0) ip1 ip2..." && exit 1

# Scan arguments for user/password or other options...
while getopts u:p: o; do
    case "$o" in
        u) user=$OPTARG;;
        p) password=$OPTARG;;
    esac
done
ONOS_WEB_USER=${ONOS_WEB_USER:-onos} # ONOS WEB User defaults to 'onos'
ONOS_WEB_PASS=${ONOS_WEB_PASS:-rocks} # ONOS WEB Password defaults to 'rocks'
```

Figure 6. Adding a new user

2) SOUTHBOUND INTERFACE MITM ATTACK

In this experiment, a man in the middle attack was attempted on the southbound interface on the SDN environment. This involved sniffing traffic between each device pair in the network and also poisoning the ARP cache of the devices along these routes. A software called Cain and Abel (Cain & Abel, 2014) was used for this purpose, which is capable of sniffing traffic between network devices and retrieving the contents of packets in transit, even in SSL connections.

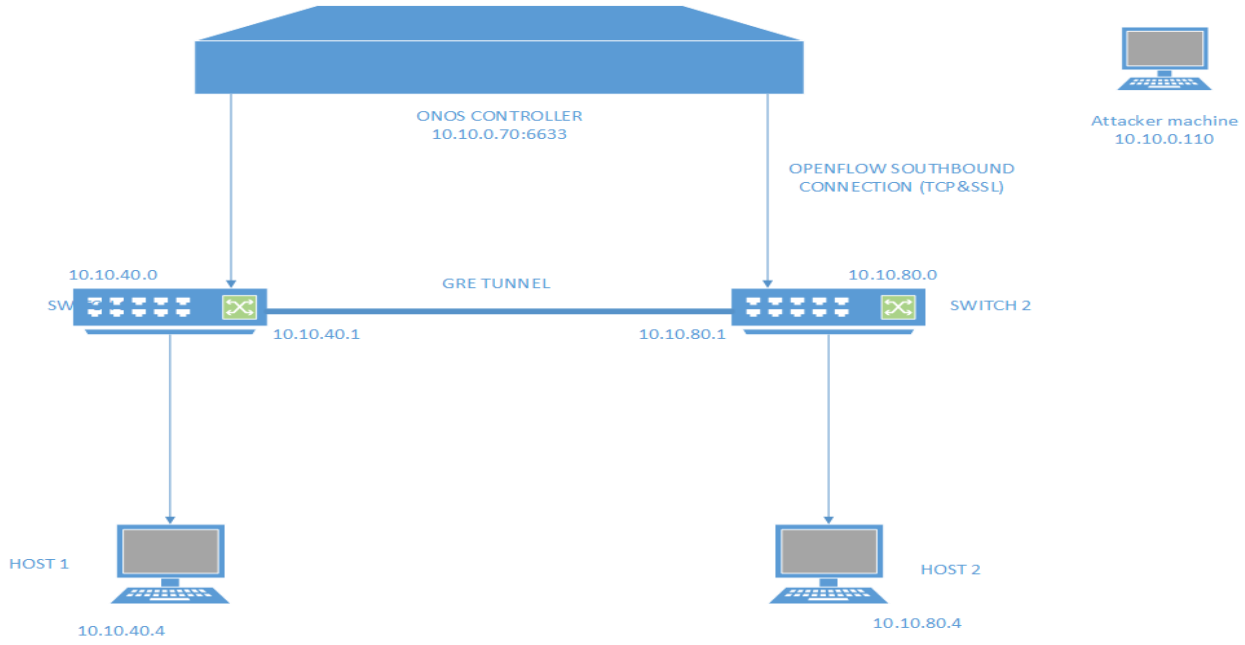

Figure 7. Showing MITM scenario

This attack is taking into consideration that the attacker has prior knowledge of network information such as subnet address and IP address of individual devices, through a reconnaissance (passive) attack or any other means. The ARP sniffer was configured to sniff for traffic between switch 1 and the ONOS controller, switch 1 and 2 and finally, switch 2 and the controller, as seen in the screenshot below

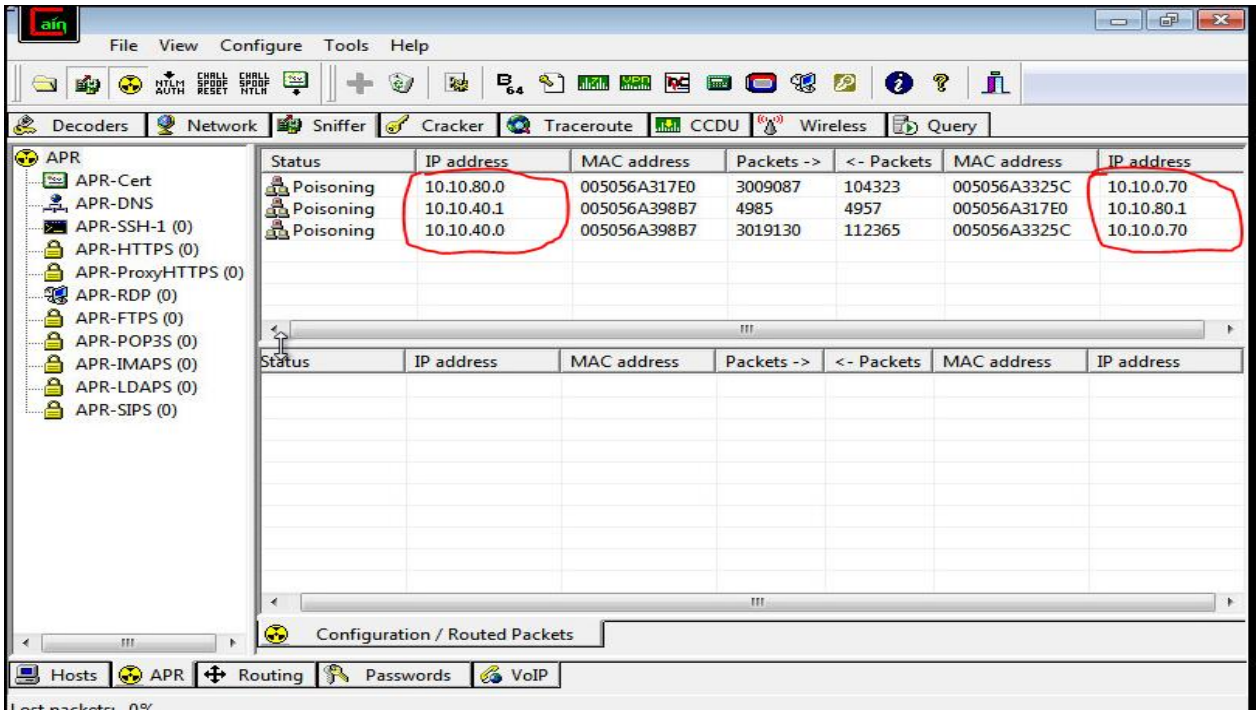

Figure 8. Showing routes to be poisoned in Cain and Abel software

- **Host communicating with host (through FTP)**

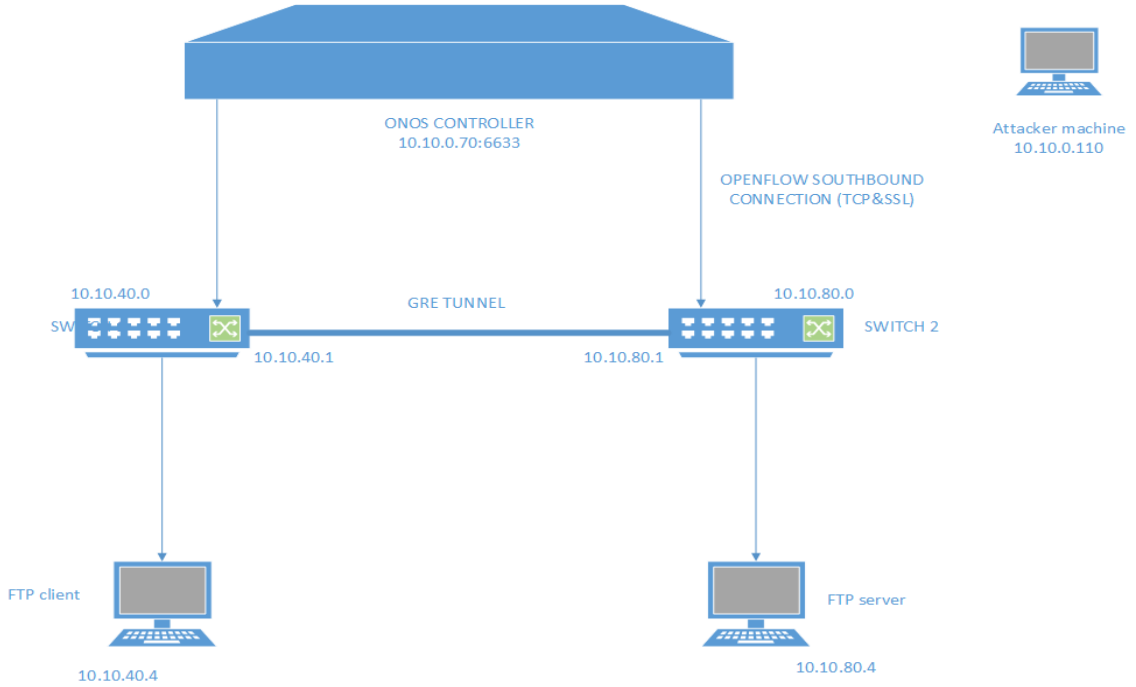

Figure 9. Showing FTP client/server attack scenario

In Figure 9, Host 1 is acting as FTP client and Host 2 as FTP server. An FTP connection was initiated from the client.

This was done with the intention of sending traffic through the network to test if the sniffer running on the attacker machine would retrieve the credentials above, along the poisoned routes.

- **Host communicating with Webserver**

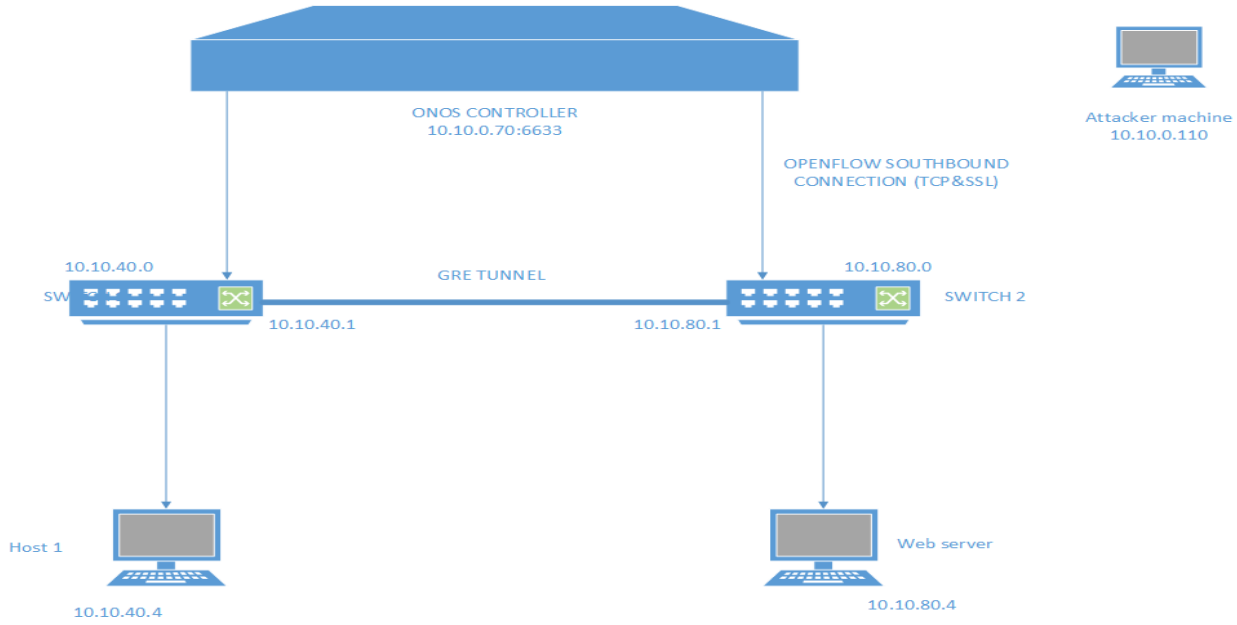

Figure 10. Showing Host/webserver attack scenario

In this scenario, a simple HTML authentication page on the web server was created, which is basically a page that requires a username and password to access a resource or another page.

A tool named Netcat (Netcat, 2013) was used, which is capable of listening in for network connection through specified ports, to listen in on port 80 for the html file.

With the attacker machine still sniffing for connections and Netcat listening on port 80, the web server was accessed through the host server using the proper credentials.

3) DENIAL OF SERVICE ATTACK

The goal of this experiment is to cause the ONOS controller to go offline, which affects other devices on the network layer, because they cannot connect and subsequently receive instructions from the controller on how to handle certain packet types. This can be done in various ways, but in this experiment, we chose the method of generating and sending large quantities of modified (enlarged) packets to see if the ONOS controller has any protective measures for abnormally large quantities of traffic.

In this test, we made use of **hping3** (0DAYsecurity.com, 2010) ping-utility tool in Kali-linux environment. This tool can send specific amount of packets of a certain size (which could be larger than normal) and to a specific host port. This attack makes use of a technique called the half-open SYN flood attack. In this technique, the attack machine makes the SYN requests all appear valid, but because the IP addresses are random and fake, it is impossible for the controller to close down the connection by sending RST (reset) packets back to the attack machine. Instead, the connection stays open. Before time-out can occur, another SYN packet arrives from the attack machine. This keeps the ONOS controller always busy with requests and causes a scenario where it is unable to service legitimate requests.

```
root@kali:~# hping3 -V -c 10000000 -d 1400 -p 6633 --flood --rand-source 10.10.7
0.0
using eth0, addr: 10.10.0.94, MTU: 1500
HPING 10.10.70.0 (eth0 10.10.70.0): NO FLAGS are set, 40 headers + 1400 data byt
es
hping in flood mode, no replies will be shown
```

Figure 11. hping command for DOS attack

The command above consists of the following:
-V: Run in verbose mode (display extended information)
-c: Packet count (10 million)
-d: Data size (1400 data bytes)
-p: Destination port (6633, the ONOS port)
--flood: send packets a fast as possible and don't show replies
--rand-source: randomize source ports

V. ANALYSIS AND RESULTS

A. Northbound interface MITM attack

With the remote system, ONOS controller and the Ettercap software set-up to sniff connections between the ONOS controller and the remote machine, the attack machine proceeded to access the ONOS GUI from the remote system.

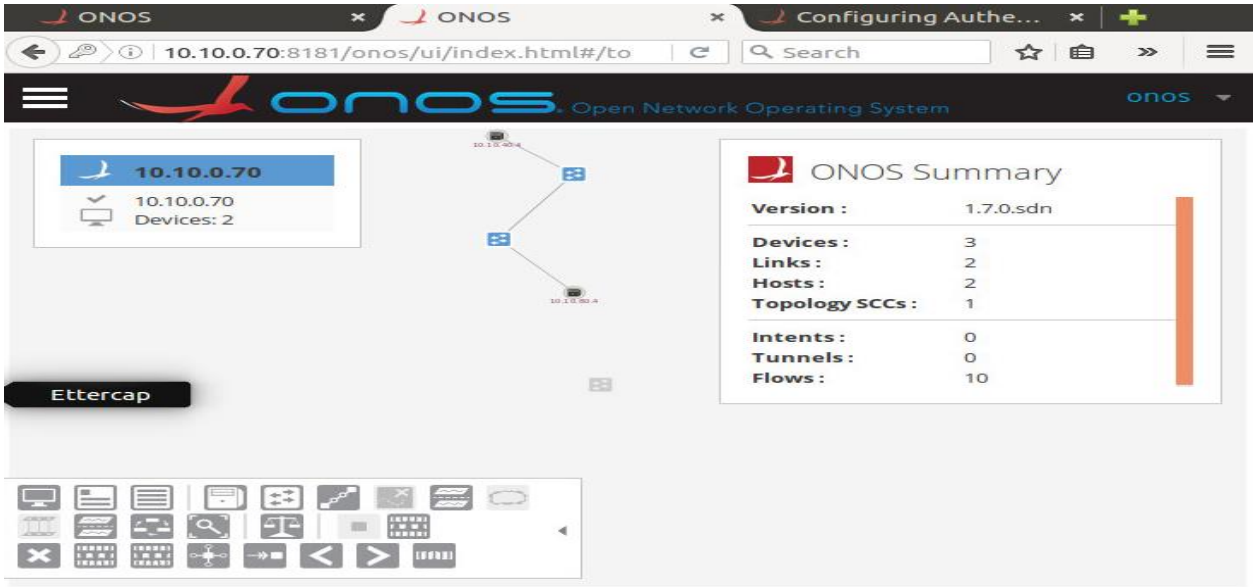

Figure 12. Remote system accessing ONOS controller GUI

Once the authentication was complete, Ettercap was able to capture the user credentials in plain text as seen in Figure 13

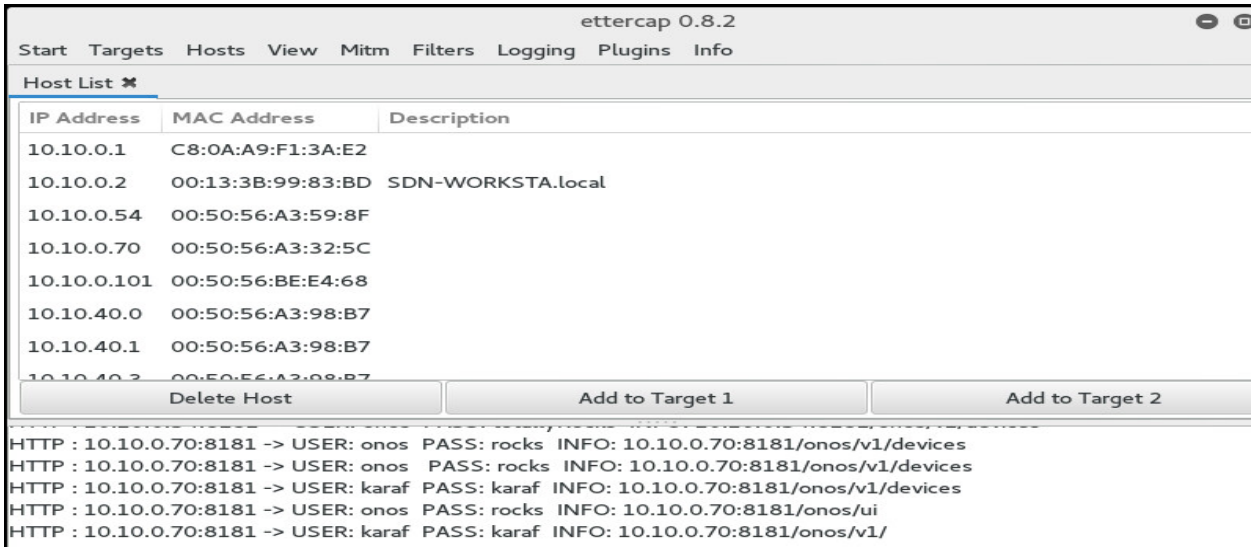

Figure 13. Recovered credentials in Ettercap

This issue is caused by not implementing a secure HTTP (HTTPS) for connections to the northbound interface in ONOS. While this feature exists in ONOS, it is not a default setting and is left to be implemented by the user. But from experience, turning on the feature is hard, and not as easy and straightforward as it could be. In this attack scenario, we were able to achieve the following:

- Perform an ARP-cache poisoning attack (the most common version of a MITM attack) on the controller;
- Perform traffic sniffing
- Capture login credentials to the ONOS web interface

B. Southbound interface MITM attack

Our southbound tests were based on two popular application-layer protocols: HTTP and FTP. Cain and Abel software was able to capture some traffic running through the network, however, was not able to capture any credentials in transit.

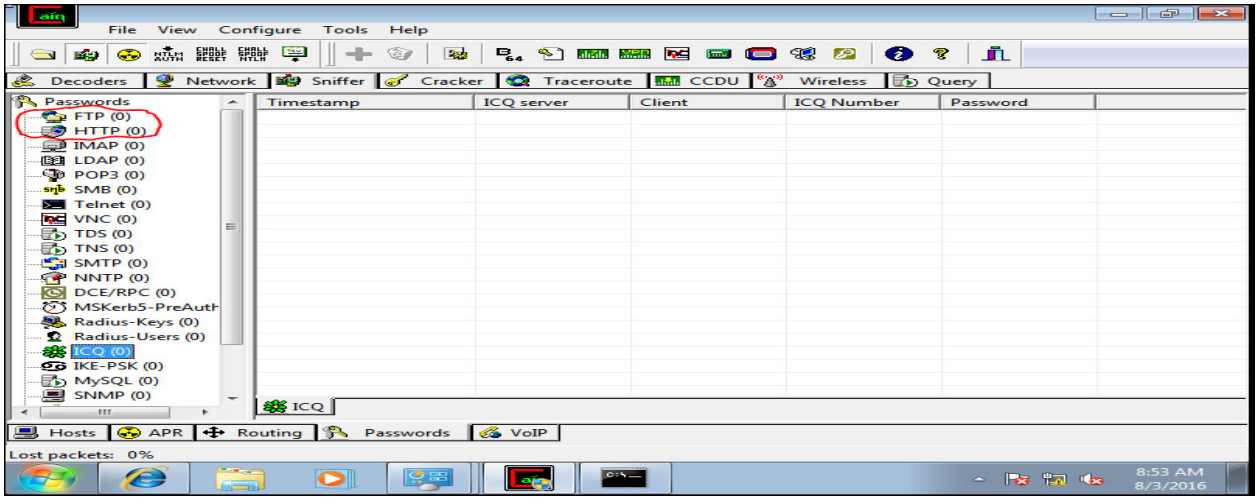

Figure 14. Showing Cain and Abel password section

Ettercap also produced the same results.

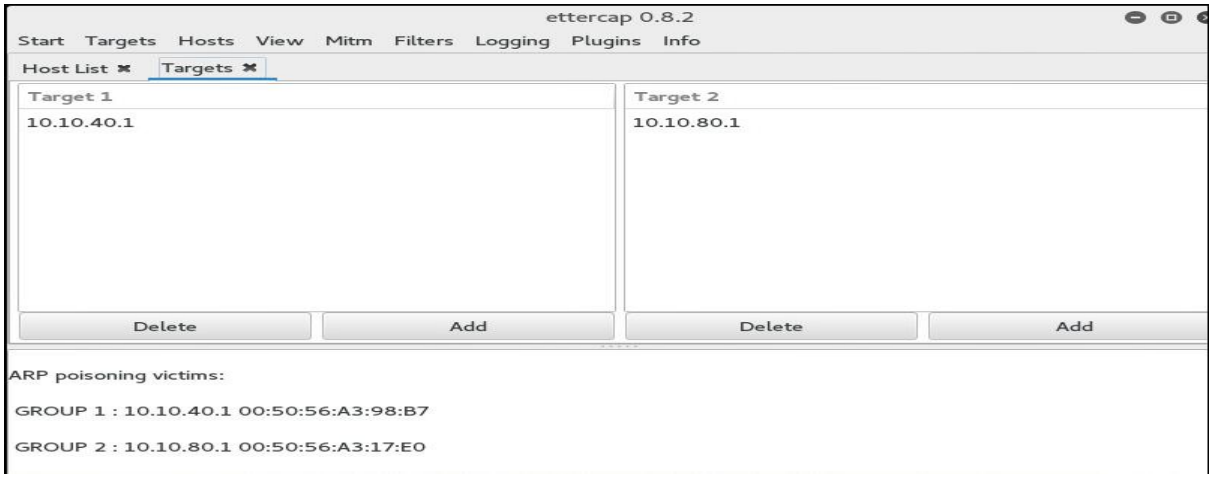

Figure 15. Ettercap recovery attempt

C. Denial of service attack

We ran the following command from three terminals, exponentially increasing its effect and simulating a Distributed denial of attack scenario.

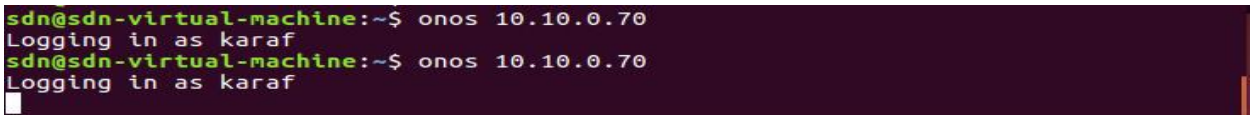

Figure 16. ONOS attempting to initialize

In the screenshot above, we see that the ONOS controller tries to initialize, but it cannot because of heavy utilization.

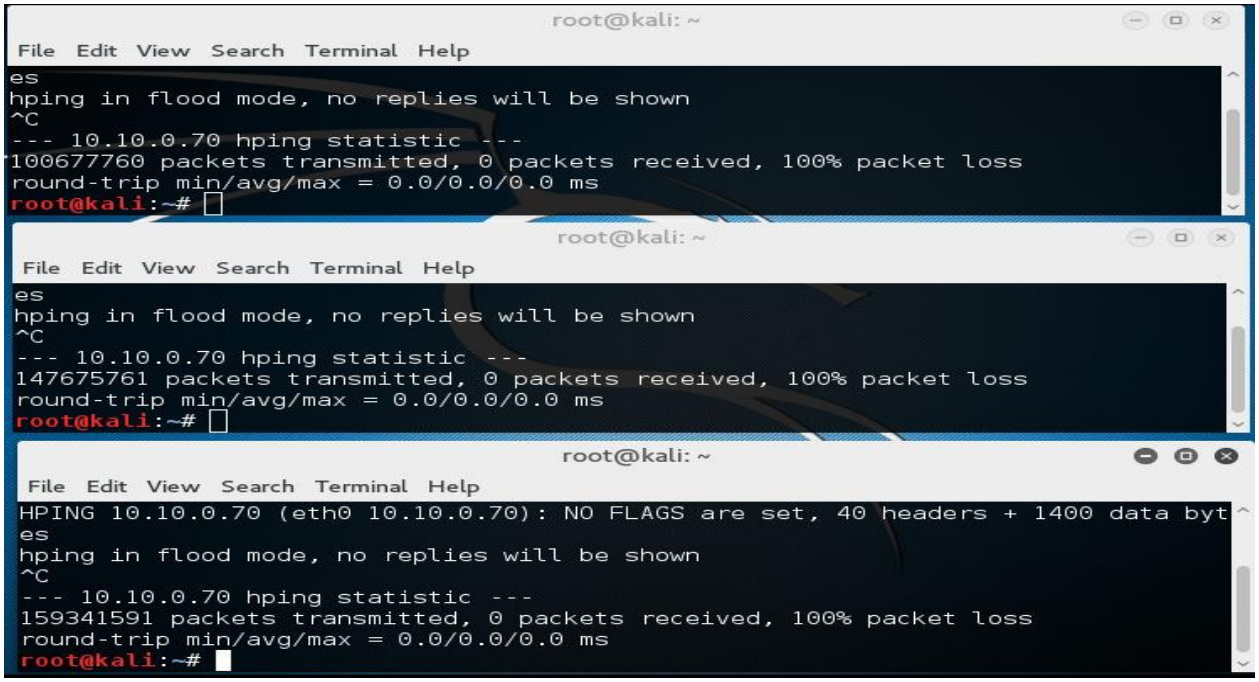

Figure 17. Showing hping3 statistics for the three terminals

The screenshot above shows the terminals with their respective statistics.

Overall it took (100677760 + 147675761 + 159341591) = 407,695,112 packets and approximately 328 seconds to cause a denial of service on the ONOS controller.

```
onos@onos-virtual-machine:~$ netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 10.10.55.0:34992        10.10.55.0:8181         ESTABLISHED
tcp        0      0 localhost:52814         localhost:6633          ESTABLISHED
tcp        0      0 localhost:52816         localhost:6633          ESTABLISHED
tcp        0      0 localhost:52812         localhost:6633          ESTABLISHED
tcp6       0      0 10.10.55.0:6633         10.10.0.50:31641        SYN_RECV
tcp6       0      0 10.10.55.0:6633         10.10.0.50:38830        SYN_RECV
tcp6       0      0 10.10.55.0:6633         10.10.0.50:31644        SYN_RECV
tcp6      60      0 10.10.55.0:6633         10.10.90.1:54535        ESTABLISHED
tcp6       0      0 10.10.55.0:6633         10.10.0.50:31648        SYN_RECV
tcp6       0      0 10.10.55.0:6633         10.10.0.50:31657        SYN_RECV
tcp6       0      0 10.10.55.0:6633         10.10.0.50:31643        SYN_RECV
tcp6       0      0 localhost:6633          localhost:52814         ESTABLISHED
tcp6       0      0 10.10.55.0:6633         10.10.0.50:31649        SYN_RECV
tcp6       0      0 10.10.55.0:6633         10.10.0.50:31653        SYN_RECV
tcp6       0      0 10.10.55.0:6633         10.10.0.50:31658        SYN_RECV
tcp6       0      0 10.10.55.0:6633         10.10.0.50:31654        SYN_RECV
tcp6       0      0 10.10.55.0:6633         10.10.0.50:31655        SYN_RECV
tcp6       0      0 10.10.55.0:6633         10.10.0.50:31642        SYN_RECV
tcp6       0      0 10.10.55.0:6633         10.10.0.50:38831        SYN_RECV
tcp6       0      0 10.10.55.0:6633         10.10.0.50:38835        SYN_RECV
tcp6       0      0 10.10.55.0:6633         10.10.0.50:38837        SYN_RECV
tcp6       0      0 10.10.55.0:6633         10.10.0.50:38829        SYN_RECV
tcp6       0    357 10.10.55.0:8181         10.10.55.0:34992        ESTABLISHED
tcp6       0      0 10.10.55.0:6633         10.10.0.50:38834        SYN_RECV
tcp6       0      0 10.10.55.0:6633         10.10.0.50:31652        SYN_RECV
tcp6       0      0 localhost:6633          localhost:52812         ESTABLISHED
tcp6       0      0 10.10.55.0:6633         10.10.0.50:31651        SYN_RECV
tcp6       0      0 10.10.55.0:6633         10.10.0.50:38833        SYN_RECV
```

Figure 18. Half open connections seen on the ONOS controller port (SYN_RECV)

The screenshot above shows the half open connections being received on the ONOS controller port 6633 from the random foreign ports (the attacker address). Notice the state is still in SYN_RECV, which confirms that there is not complete handshake.

VI. DEFENCE MEASURES

The most worrying fact about dealing with Man in the middle attacks is that they are very hard to detect once they have already occurred. It's better to prevent the attack before it is successfully executed. There are a number of techniques which can defend against this attack, especially the use of the ARP poisoning technique, used in this experiment. Some of these techniques include:

ARP ALERT (Arpalert.org, 2013**)**: This tool examines all the ARP traffic on a given interface and then ensures that the IP address to MAC address mappings matches a pre-configured list. If the result of a MAC to IP address resolution does not match this list, the software alerts the administrator of a possible attack. These tools can be deployed on the Linux host which the controller is installed on.

Packet filtering: Setting up packet filters can be very helpful in mitigating this attack. Packet filters inspect packets whilst in transmission within a network and they can filter out and block packets with conflicting source address information. Conflicting in this case implies packets coming from outside the network that display addresses from inside the network. This could help in filtering out addresses like that of the attacker machine, which in an ideal situation, would not be part of the network.

Encryption: ARP MITM traffic spoofing attacks can be mitigated by encrypting data in the network before transmission and also authentication upon arrival. The use of HTTPS (SSL HTTP) in my experiment would have mitigated against the north-bound MITM attack. On the south-bound network, technologies like IPsec can be implemented over the GRE tunnel can also be used to ensure confidentiality. Also the use of SSL over the open-flow connection between the controller and the switch, which was done in this project through the use of self-signed certificates, is also a good mitigation technique.

Intrusion prevention system: A tools such as Snort has an ARP spoof pre-processor, which has the same operations as ARP alert, but does this one wider scale in that it watched the entire network for ARP attacks. This will involve setting up an IPS and configuring it with the network interface used by the SDN controller.

For the Denial of service attack performed in this experiment, the use of packet filter, filtering packets from a specific IP address, might have sufficed but only temporarily. The attacker can adapt in this situation by launching the attack from another spoofed IP address or worse, from multiple spoofed IP addresses in a situation known as a distributed denial of service attack.

Another technique of protection is decrease the total time keeping half-open connections in the queue. When a server receives a TCP request, its sends a response with SYN and ACK flags set, and then waits for an ACK from the client which in the SYN flood attack, never comes. In this situation, the server retransmits the response packet severally and after a few minutes of no response, removes the half-open connection. This defense technique attempts to reduce this time of removing half open connections by changing the time of first retransmission and also changing the total number of retransmissions.

VII. CHALLENGES & SUGGESTIONS FOR FUTURE WORK

During the course of this project, some of the challenges were as following

- Part of our findings are that there are not too many commercial solutions out in the market handling SDN security, so It is challenging to review a full commercial security implementation.
- There are also not many resources to discuss and aid a user with a security implementation in SDN, whether it is from the northbound or southbound interface. The resources we found were either ambiguous or much more generic, and left much more to infer.
- Also, some ONOS releases are quite unstable and erratic in implementation. We had to re-install or revert back to stable snapshots of the virtual memory because of implementation errors. It may be due to the fact that I was using the most recent version (1.7.0), which may be subject to more stable revisions.

Future work in this area could include the following:

- The man in the middle attack performed on the SDN southbound network could be tried with a wider variety of tools to provide a more comprehensive study.
- It is a valid argument that the ARP attack also may have been unsuccessful because all the network devices already had full connectivity. The ARP attack functions by the attacker poisoning the ARP cache of two devices while attempting to establish a connection. However, in the real world, this type of attack could be successful in situations where the network goes down for an upgrade. The attacker will sit in the network waiting for the network to come back up and poison the routes while the devices are re-establishing connectivity.
- An SSL implementation on the northbound interface of ONOS could improve security since HTTPS has become the standard in internet security.

VIII. CONCLUSION

This project provided an opportunity not only to interface with the ONOS controller, which is a more recent development and addition to the SDN project; but also to discover some of its vulnerabilities and be able to exploit them in a controlled environment through penetration testing. This is of significance to the industry, much because the ONOS project has attracted interest from the private technology sector, which is an indication of major future adoption. This project represents one of the early attempts to test the security of the ONOS controller

Just like other widely used IT infrastructure and also traditional networks which it is modelled after, SDN has own security challenges, especially with the control layer. The controller stands as a central point of failure if it can be compromised. This paper shows that it is possible in more than one aspect. Through a man-in-the middle attack, it is possible to recover adequate information to compromise the controller. The DOS attack tampers with

the availability of the network, which can lead up to a plethora of challenges. With the controller compromised, the network can be manipulated in any way the attacker chooses.

The SDN controller contains policies for configuring and managing the data layer, providing a single point of failure on the network as a whole. This project was focused on exposing a few security vulnerabilities alongside gaining advanced knowledge on the operations of the ONOS controller in a bid to stimulate future work. The results of this project emphasize that if the SDN technology is to cement its place in commercial implementations, it is imperative that security be taken as priority.

IX. REFERENCES

1. Kreutz, D., Ramos, F., & Verissimo, P. (2013, August). Towards secure and dependable software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (pp. 55-60). ACM.

2. Chalyy, D. J., Nikitin, E. S., & Antoshina, E. J. (2015, April). A Simple Information Flow Security Model for Software-Defined Networks. In*Proceedings of the 17th Conference of Open Innovations Association FRUCT. Yaroslavl, Russia* (pp. 276-282).

3. Hu, H., Han, W., Ahn, G. J., & Zhao, Z. (2014, August). FLOWGUARD: building robust firewalls for software-defined networks. In *Proceedings of the third workshop on Hot topics in software defined networking* (pp. 97-102). ACM.

4. Jafarian, J. H., Al-Shaer, E., & Duan, Q. (2012, August). Openflow random host mutation: transparent moving target defense using software defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks* (pp. 127-132). ACM.

5. Benton, K., Camp, L. J., & Small, C. (2013, August). Openflow vulnerability assessment. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (pp. 151-152). ACM.

6. Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., ... & Parulkar, G. (2014, August). ONOS: towards an open, distributed SDN OS. In *Proceedings of the third workshop on Hot topics in software defined networking* (pp. 1-6). ACM.

7. Moshref, M., Bhargava, A., Gupta, A., Yu, M., & Govindan, R. (2014, August). Flow-level state transition as a new switch primitive for SDN. In*Proceedings of the third workshop on Hot topics in software defined networking* (pp. 61-66). ACM.

8. Porras, P. A., Cheung, S., Fong, M. W., Skinner, K., & Yegneswaran, V. (2015, February). Securing the Software Defined Network Control Layer. In*NDSS*.

9. Scott-Hayward, S., Natarajan, S., & Sezer, S. (2015). A survey of security in software defined networks. *IEEE Communications Surveys & Tutorials*,*18*(1), 623-654.
10. Scott-Hayward, S. (2015, April). Design and deployment of secure, robust, and resilient SDN Controllers. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on* (pp. 1-5). IEEE.

11. Shin, S., & Gu, G. (2013, August). Attacking software-defined networks: A first feasibility study. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (pp. 165-166). ACM.

12. Shin, S., Yegneswaran, V., Porras, P., & Gu, G. (2013, November). AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (pp. 413-424). ACM.

13. Namal, S., Ahmad, I., Gurtov, A., & Ylianttila, M. (2013, November). Enabling secure mobility with openflow. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for* (pp. 1-5). IEEE.

14. Dangovas, V., & Kuliesius, F. (2014, January). SDN-driven authentication and access control system. In *The International Conference on Digital Information, Networking, and Wireless Communications (DINWC)* (p. 20). Society of Digital Information and Wireless Communication.

15. Chen, Y. J., Lin, F. Y., & Wang, L. C. (2014). Dynamic security traversal in openflow networks with qos guarantee. *International Journal of Science and Engineering*, *4*(2), 251-256.

16. Brooks, M., & Yang, B. (2015, September). A Man-in-the-Middle attack against OpenDayLight SDN controller. In *Proceedings of the 4th Annual ACM Conference on Research in Information Technology* (pp. 45-49). ACM.

17. Kandoi, R., & Antikainen, M. (2015, May). Denial-of-service attacks in OpenFlow SDN networks. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)* (pp. 1322-1326). IEEE.

18. Kiran Vemuri. (2014). What is the difference between forwarding state and flow entries in SDN? Retrieved from Quora: https://www.quora.com/What-is-the-difference-between-forwarding-state-and-flow-entries-in-SDN

19. Koshibe, A., & Hall, J. (2016, June 3). ONOS from scratch. Retrieved from Onosproject.org: https://wiki.onosproject.org/display/ONOS/ONOS+from+Scratch

20. Koshibe, A., & Parlakisik, M. (2016, June 23). Installing and running ONOS. Retrieved from Onosproject.org: https://wiki.onosproject.org/display/ONOS/Installing+and+Running+ONOS

21. Lee Doyle. (2013). Why do I need SDN technology? Retrieved from TechTarget: http://searchsdn.techtarget.com/answer/Why-do-I-need-SDN-technology

22. MININET TEAM. (2016). Mininet Walkthrough. Retrieved from mininet.org: http://mininet.org/walkthrough/

23. ONF. (2014). Principles and Practices for software defined networks. Retrieved from Open networking: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf

24. Pfaff, B. (2015, November 29). Configuring Open vSwitch for SSL. Retrieved from Github: https://github.com/openvswitch/ovs/blob/master/INSTALL.md

25. zhang, s. (2016, January 20). Configuring OVS connection using SSL/TLS with self-signed certificates. Retrieved from Onosproject.org: https://wiki.onosproject.org/pages/viewpage.action?pageId=6358090

26. DigitalOcean. (2014). How To Use Netcat to Establish and Test TCP and UDP Connections on a VPS | Digitalocean.com. Retrieved 7 July 2016, from https://www.digitalocean.com/community/tutorials/how-to-use-netcat-to-establish-and-test-tcp-and-udp-connections-on-a-vps

27. 0DAYsecurity.com. (2010). Hping3 Examples - Firewall testing | 0daysecurity.com. Retrieved 5 July 2016, from http://0daysecurity.com/articles/hping3_examples.html

28. Ettercap (2015). Ettercap Home Page. (2015). Ettercap.github.io. Retrieved 5 July 2016, from https://ettercap.github.io/ettercap/

29. Netcat (2013). Netcat: the TCP/IP swiss army. Nc110.sourceforge.net. Retrieved 4 July 2016, from http://nc110.sourceforge.net/

30. Cain and Abel (2014). Retrieved 15 July 2016, from http://www.oxid.it/index.html

31. Arpalert.org (2013). Arpalert.org: main page. Retrieved 4 August 2016, from http://www.arpalert.org/arpalert.html