INVESTIGATING SDN AS A SECURITY TOOL

Karan Desai
Rutgers University
desai.karan@cs.rutgers.edu

Abstract

Software Defining Networking (SDN) is a new approach of building networking equipment and software that separates elements of these systems. It allows abstraction of lower level functionality and all can be achieved by programing and not hardware interference is needed for configuration purposes. This project deals with how security professionals and engineers can use SDN to their advantage and create a more secure environment. We will cover some potential use cases for SDN and security by showing two proof-of-concepts. The first one is the Rule Based Forwarding and the second is an added feature of anti-ARP poisoning in the switch. Note that this project is not about security in SDN but rather how SDN and specifically Openflow can be used as a security advantage.

I. Introduction

Software Defined Networking (SDN) is becoming increasingly popular and we are certain that it has a great future ahead. A major portion of the security community is not aware of this upcoming trend as it will not affect be affecting that major sector soon. Discussion about SDN's security has already started and the security community is doing work at a good scale. This work is inspired by various such projects. The important ones are FortNOX, SE-Floodlight and Cisco's upcoming OnePK. All of them revolve around similar strategies. We want a more efficient way to handle and configure our network and network equipment. The division of the controller and Datapath planes have led to great new opporunities. Openflow was a huge success during this time and now the SDN community is trying to make it the de facto standard of implementation, which is why most of the vendor implementations are based on it. So how does Openflow come in the picture? Its inventors consider OpenFlow an enabler of software-defined networking. OpenFlow is mainly used as the protocol between the switch and controller on a secure channel. OpenFlow was originally introduced as a way for researchers to run experimental protocols in the networks they use every day. OpenFlow was based on an Ethernet switch, with an internal flow-table, and a standardized interface to add and remove flow entries. OpenFlow provides an open protocol to program the flow table in different switches and routers. It now being continously updated with features.The datapath of an OpenFlow Switch consists of a Flow Table, and an action associated with each flow entry. OpenFlow lets a network administrator define rules on a switch that allows us to do what we are trying to achieve in this project. But we are still not using the inbuilt functions (provided for it in Openflow 1.3) as we want to show how easy it can be to make your own custom model for such tasks. Also the version 1.3 though being more advanced lacks some needed documentation that limited the scope of this project. The things that we need for successful implementation include the Openflow framework(1.0), the OpenVSwitch and the

POX controller(POX is a Python-based SDN controller platform geared towards research and education).Our virtual environment is Mininet, which allows us to create the virtual environment with switches and hosts. The controller is where all the code logic goes. Without the controller its just static hosts and switches that cannot even communicate. We will use wireshark dissector designed for Openflow. They appear as OF in wireshark. The following diagram shows how Openflow works

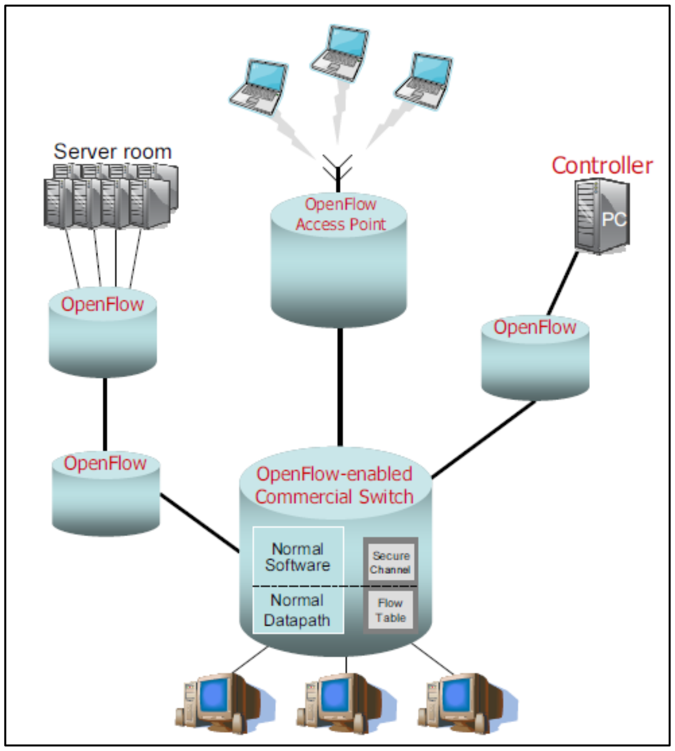


Figure 1

The important benefits would include:

- Your networking equipment can take care of the security for you. Our example of this: the anti ARP poisoning logic
- Greater control (and more work) for the administrator
- Logic/control easy to program and change. Developing standards like Openflow.
- Switch is now working on upper layers too
- Early detection
- It can not only detect but trigger appropriate actions
- Can be made generic to applications so that applications do don't have to worry about such security issues.

II. Main Concepts

The following are the two main ideas implemented in this project.

A. Rule Based Forwarding

This is a very familiar technique where we try to implement forwarding rules. Imagine it as a firewall that can be extended and edited as and when needed. The big difference is that this processing is now being done at the networking equipment, the switch in this case. Request that you don't want your switches to forward to the servers can be stopped there itself. You do not need a processing engine at your server for this task (or even a dedicated server in most cases) to this task. The most common are: Forward the packet out a given port or ports, encapsulate the packet and forward to the controller so a decision can be made about it, drop the packet, or just send the packet to the normal switch processing pipeline. Commonly, if the packet matches no rules in the flow table, the packet can be sent to the controller and the controller can decide what to do with it. This concept is similar to what Cisco demonstrated during the recent RSA 2013 conference. Not only will you be able to implement this logic at these lower levels but will trigger the appropriate response and the respective action will be executed. By default all packets from all destinations, to all sources are blocked, on all ports. We will still allow all the network layer protocols like RARP, ARP, and ICMP so the switch obviously still does the work it is supposed to do. The task for the switch increases because now as it has to parse layers above the ones it normally does. This rule based technology applies to filtering on these layers. A sample rule would like this[Source: 00:00:00:00:00:01 ; Destination: 00:00:00:00:00:02 ; Port: 443] . This means that this specific source cannot send any type of packets to this destination except for packets on port 443 HTTPS. Programmatically this logic is implemented by first filtering the TCP packets. Because most of these protocols work on TCP this is the first thing we do. Once a packet is captured, it is parsed to see if it is has a port 443 from the source and if so then the packet is forwarded to the appropriate destination based on the IP-Mac mapping in the forwarding table. The performance can be poor based on the feature desired and type of implementation. But that is not in the scope of this project. The newer version Openflow 1.3 has the following fields to assist is this kind of a system.

B. Anti ARP Poisoning

ARP spoofing is a technique whereby an attacker sends fake ("spoofed") Address Resolution Protocol (ARP) messages onto a Local Area Network. Generally, the aim is to associate the attacker's MAC address with the IP address of another host (such as the default gateway), causing any traffic meant for that IP address to be sent to the attacker instead. Openflow by itself does not implement a technique wherein this is taken care of. We are not trying to provide a solution to it, but rather show that how any security holes like this can easily be take care of by simple change to the switch code in openflow. Adrian Crenshaw has nicely demonstrated this during his work on this topic. For mitigating this security concern we do two things. First we set a timeout for the forwarding table, i.e. every 10 seconds the entries in the table should start disappearing (the time is chosen randomly and can be based on specific requirements). The second thing that we implement is that we filter ARP packets. If an attempt is seen to map a MAC address to an IP that is already taken according to the IP to MAC address table, the switch can detect this and take counter measures.

III. Implementation

Our implementation uses Mininet as the virtual environment based upon Openflow and Openswitch. Our controller is the POX controller so our code is Python. We SSH to the openflow environment using Terminals in Mac OSX. The virtualization software is vmware fusion. Wireshark's dedicated version for openflow is used for traffic monitoring. Our network topology is based of the openflow tutorial for simplicity. The figure below shows out implemented topology for the switch.

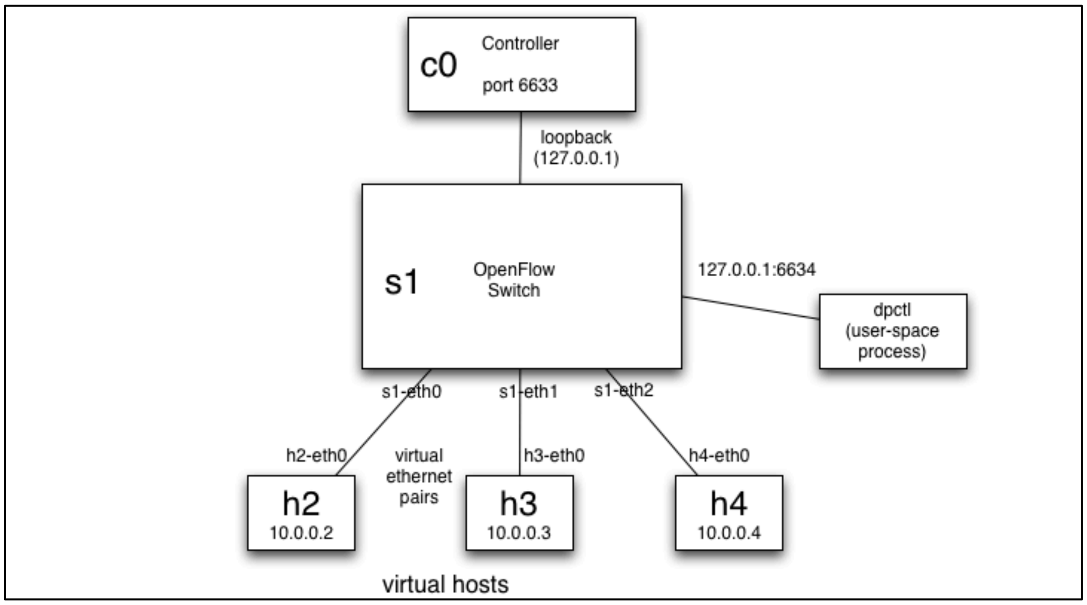


Figure 2

The following commands are required to execute it -

SSH to Mininet's Environment
ssh -Y openflow@openflow (Password : openflow)

Create Topology
Openflow> sudo mn --topo single,3 --mac --switch ovsk --controller remote

Start Controller

./pox.py log.level --DEBUG misc.myswitch

Check topology

Mininet> net

Make a host act as a server

h2 python -m SimpleHTTPServer 80 &

Send a client request to server

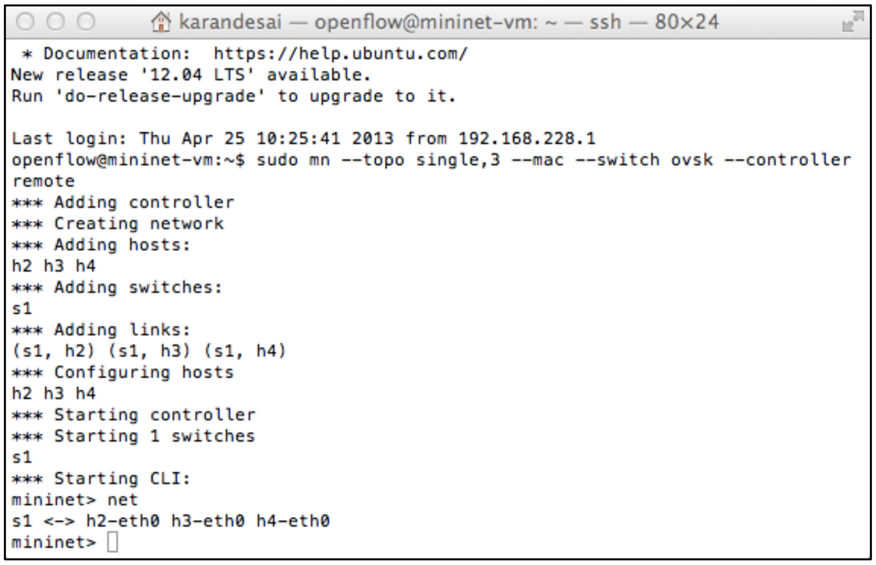
h3 wget -O - h2

Host reachability

Ping all

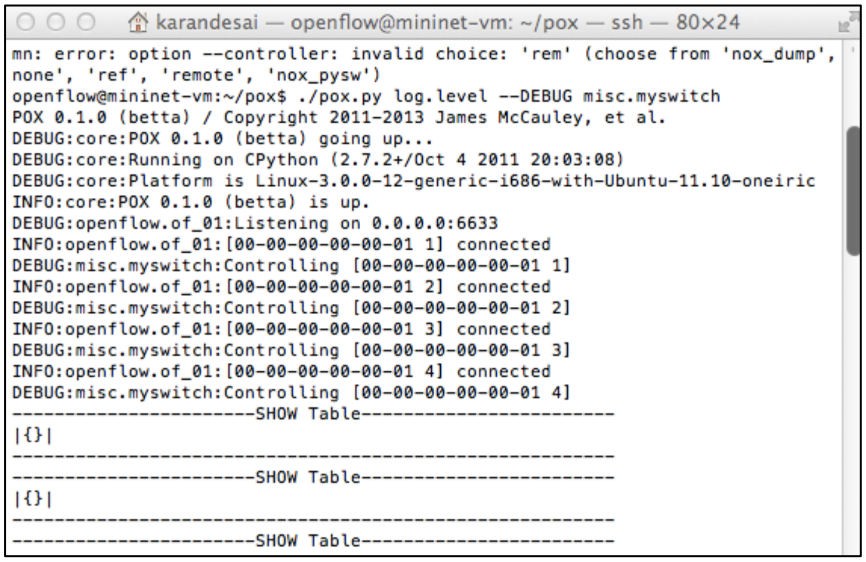
Start the controller using this set of commands. Once the controller starts we will be able to actually confirm the simulated environment. We are trying to setup a client server environment, so first we make h2 as the server and the client will try to send a request to h2. The following is the scenario where h3 tries to download some files from h2. Now as per the implementation we notice two important factors here. First, we had defined a rule that h3 cannot communicate with h2 on port 80, based on which we get the message saying that the action is blocked on execution. Also one would notice that for the timeout of 10 seconds that we coded, the tables get updated. All these messages would appear in the ongoing log.

Refer to the following diagrams for the screenshots of the implementation.



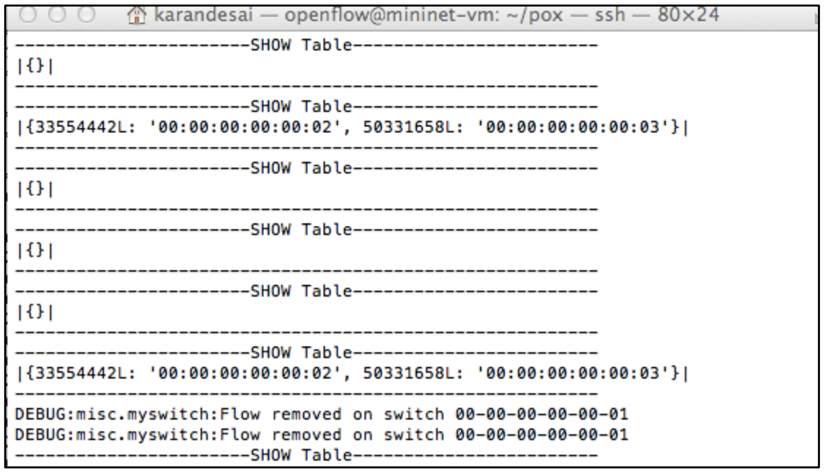
```
 * Documentation:  https://help.ubuntu.com/
New release '12.04 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Thu Apr 25 10:25:41 2013 from 192.168.228.1
openflow@mininet-vm:~$ sudo mn --topo single,3 --mac --switch ovsk --controller
remote
*** Adding controller
*** Creating network
*** Adding hosts:
h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h2 h3 h4
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> net
s1 <-> h2-eth0 h3-eth0 h4-eth0
mininet> 
```

Figure 3. Starting the Topology



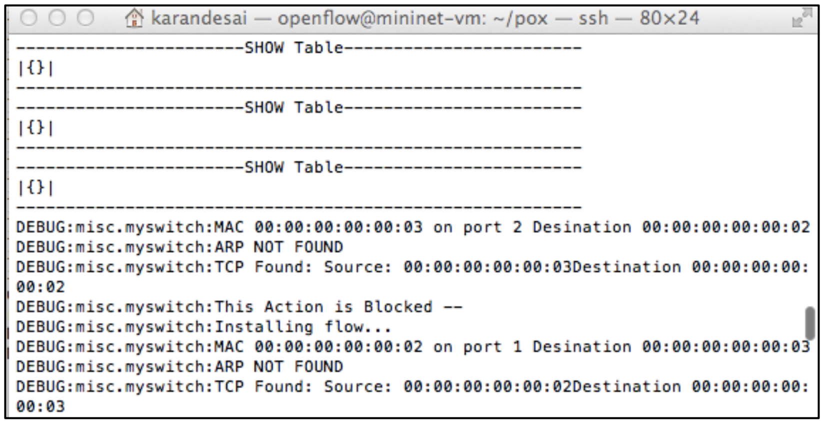
```
mn: error: option --controller: invalid choice: 'rem' (choose from 'nox_dump', 
none', 'ref', 'remote', 'nox_pysw')
openflow@mininet-vm:~/pox$ ./pox.py log.level --DEBUG misc.myswitch
POX 0.1.0 (betta) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.1.0 (betta) going up...
DEBUG:core:Running on CPython (2.7.2+/Oct 4 2011 20:03:08)
DEBUG:core:Platform is Linux-3.0.0-12-generic-i686-with-Ubuntu-11.10-oneiric
INFO:core:POX 0.1.0 (betta) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
DEBUG:misc.myswitch:Controlling [00-00-00-00-00-01 1]
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:misc.myswitch:Controlling [00-00-00-00-00-01 2]
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
DEBUG:misc.myswitch:Controlling [00-00-00-00-00-01 3]
INFO:openflow.of_01:[00-00-00-00-00-01 4] connected
DEBUG:misc.myswitch:Controlling [00-00-00-00-00-01 4]
-----------------------SHOW Table-----------------------
|{}|
--------------------------------------------------------
-----------------------SHOW Table-----------------------
|{}|
--------------------------------------------------------
-----------------------SHOW Table-----------------------
```

Figure 4. Starting the controller



```
-----------------------SHOW Table-----------------------
|{}|
--------------------------------------------------------
-----------------------SHOW Table-----------------------
|{33554442L: '00:00:00:00:00:02', 50331658L: '00:00:00:00:00:03'}|
--------------------------------------------------------
-----------------------SHOW Table-----------------------
|{}|
--------------------------------------------------------
-----------------------SHOW Table-----------------------
|{}|
--------------------------------------------------------
-----------------------SHOW Table-----------------------
|{}|
--------------------------------------------------------
-----------------------SHOW Table-----------------------
|{33554442L: '00:00:00:00:00:02', 50331658L: '00:00:00:00:00:03'}|
--------------------------------------------------------
DEBUG:misc.myswitch:Flow removed on switch 00-00-00-00-00-01
DEBUG:misc.myswitch:Flow removed on switch 00-00-00-00-00-01
-----------------------SHOW Table-----------------------
```

Figure 5. Table Refresh



```
-----------------------SHOW Table-----------------------
|{}|
--------------------------------------------------------
-----------------------SHOW Table-----------------------
|{}|
--------------------------------------------------------
-----------------------SHOW Table-----------------------
|{}|
--------------------------------------------------------
DEBUG:misc.myswitch:MAC 00:00:00:00:00:03 on port 2 Desination 00:00:00:00:00:02
DEBUG:misc.myswitch:ARP NOT FOUND
DEBUG:misc.myswitch:TCP Found: Source: 00:00:00:00:00:03Destination 00:00:00:00:
00:02
DEBUG:misc.myswitch:This Action is Blocked --
DEBUG:misc.myswitch:Installing flow...
DEBUG:misc.myswitch:MAC 00:00:00:00:00:02 on port 1 Desination 00:00:00:00:00:03
DEBUG:misc.myswitch:ARP NOT FOUND
DEBUG:misc.myswitch:TCP Found: Source: 00:00:00:00:00:02Destination 00:00:00:00:
00:03
```

Figure 6. TCP Port 80 Block

IV. Future Implementation

This section of the paper is inspired from the 2013 RSA keynote that we mentioned earlier. We will give steps on how the small proof of concept that demonstrated earlier can be used for this wider systems. Let us take the example of any of the big companies that have employees logging in from various devices and they have an IT department that maintains the servers and the applications hosted on them. Now employees use different type of devices like smartphones tablets and laptops. All of them may be accessing different things and different versions on different servers. Now all this network traffic goes through some of the bottleneck switches in the data center where we try to implement our systems. We know that not all the employees are not allowed to access every thing and that the default policy is generally just block all traffic from everybody. Then access control is defined which defines who has access to what. As noted by our policy mechanism lets say that an employee gets access to files on an FTP server because of a bug in the implementation, so now the employee is able to get through and grab files as he likes. Or the scenario can be that the employee is repeatedly trying to login to areas he does not have access to. The employee is though logged into various devices by his unique ID. The switch implementation detects this access by the code logic and tries to block it at the switch level itself and hence the exploit stops there. It not only does that but the switch can trigger code that takes care of the situation the desired way. Say we had coded the switch to block the employee for sometime until it is figured out what went wrong and log him off all his sessions throughout all his devices. This actually is not very tough to implement and one can see that how it helps. Not only that but changing this logic is equally easy

V. REFERENCES

[1] Open Networking Foundation, "OpenFlow Switch Specification 1.3.0," 25 June 2012. [Online].

[2]"POX Wiki," Stanford University, [Online]. Available: https://openflow.stanford.edu/display/ONL/POX+Wiki . [Accessed 12 12 2012].

[3] I. Aggarwal, Implementation and Evaluation of ELK, an ARP scalability enhancement, 2011.

[4] S. Shin and G. Gu, CloudWatcher: Network Security Monitoring Using OpenFlow in Dynamic Cloud Networks (or: How to Provide Security Monitoring as a Service in Clouds?), 2012.

[5]E. Kissel, G. Fernandes, M. Jaffee and M. Swany, Driving Software Defined Networks with XSP, 2012.

[6] G. Yao, J. Bi and P. Xiao, Source Address Validation Solution with OpenFlow/NOX Architecture, 2011.

[7] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, OpenFlow: Enabling Innovation in Campus Networks, 2008.

[8] Adrian Crenshaw , Security and Software Defined Networking: Practical Possibilities and Potential Pitfalls (2012) [online – irongeek.com]

[9] RSA 2013, Keynote – SDN Security , Cisco , John Chambers