

Software Defined Networking: Network Diagnosis and Security

Damien Piris

John Jay College of Criminal Justice
damien.piris@jjay.cuny.edu

Sovatha Sang

John Jay College of Criminal Justice
sovatha.sang@jjay.cuny.edu

Marc Suda

John Jay College of Criminal Justice
marc.suda@jjay.cuny.edu

Chong Yu

John Jay College of Criminal Justice
chong.yu@jjay.cuny.edu

Sven Dietrich, Mathematics & Computer Science Department
John Jay College of Criminal Justice

ABSTRACT

In one month from now, there will be over 50 billion devices connected to the internet. As a result, networks will need to be scaled at a much larger levels which requires resource extensive network/security devices, therefore increasing the complexity of network management. The use of Software Defined Networks (SDN) provides an alternative approach to networking that can be a solution that provides central control over the network through the use of a programmable “controller”. When scaling the network, defending against network security attacks in traditional networks is difficult. In this paper we set up an SDN controller with the use of other detection tools such as ForenGuard [4] to test the detection of security vulnerabilities against an implemented MAC Spoofing attack. To mitigate this threat, this paper proposes to use the central control of SDN for attack detection and introduces a solution that is effective and lightweight in terms of the resources that it uses. More precisely, this paper aims to show how MAC spoofing attacks can exhaust controller resources and provides a solution to detect such attacks based on the entropy variation of the destination IP address.

KEYWORDS

Software Defined Networking, Security, Forensics, Diagnosis, MAC Spoofing

1 INTRODUCTION

This project is aimed at discovering the best practices for detecting and defending against MAC spoofing attacks on Software Defined Networks (SDN). We hope to determine what kind of tools currently exist to support network security forensics and diagnosis. We seek to further analyze SDN security policy by testing various SDN controllers and forensic tool such as ForenGuard [4], OpenDaylight [19], Floodlight [22], MongoDB [2] and Mininet [20]. In this paper, we test the effectiveness of various intrusion detection tools/applications for software defined networks. To further analyze SDN security policies with the attacks in progress, we will be using ForenGuard [4], which provides flow-level forensics and diagnosis functions in SDN networks. For the purpose of our research, we first have to set up an SDN similar to real life topology. To accomplish this, we will be utilizing a network emulator called Mininet. Mininet is a *network emulator* which creates a network of virtual hosts, switches, controllers, and links [20]; Mininet hosts run standard Linux network software, and its switches support OpenFlow [17] for highly flexible custom routing and Software-Defined Networking. Therefore, we would use Mininet for the creation of virtual switches so that they may be utilized by a controller such as ForenGuard [4]. Once the SDN environment is set up, to perform various attacks in the SDN environment while using ForenGuard [4] to further deeply analyze SDN security policy

and diagnose several real control plane attacks. When we scale our network it can be even more frustrating in networking environments using emerging Software Defined Networks. This is because traditional network troubleshooting tools are not suitable for pinpointing the root cause of problems in the control plane of a software defined networking (SDN) as a result of the control plane is decoupled. SDN are increasingly being standardized in Leading SDN Market Players such as Cisco Systems Inc, IBM Corporation, Hewlett Packard Enterprise, VMware, Juniper Networks, Huawei Technologies Co. Ltd. Since SDN is being used by the leading market.

2 BACKGROUND

Software defined networking is an emerging and developing technology. The main idea behind SDN is that it separates the switch data plane, that forwards packets, from the control plane, which is responsible for configuring the data plane [11]. The control plane is further physically distributed between a switch and a controller running on a general purpose computer. The controller communicates with the switch to instruct it how to configure the data plane by sending flow modification commands that place rules in the switch forwarding table. To put it simply, a Software Defined Network provides a way to virtualize the hardware utilized in network infrastructure; an SDN simplifies and centralizes the network therefore making it easier to configure and manage [6].

Software Defined Network was created in response to demands from large data centers, who found problems coping with very unpredictable traffic patterns. SDN eliminates the need for probabilistic exemption and will scale with technology. A tiered filtering process can be employed to inspect large volumes of data [11]. Properly architected SDNs provide the capability of real-time deep packet analysis with cascading complexity. Initial analysis may be simple,

providing a quick decision to forward packets. Further packet inspection for a subset of the flow may be determined by the algorithm. All the traffic, including elephant (high-bandwidth, long duration) data flows, will be analyzed therefore increasing the possibility of discovering malicious packets.

These traffic patterns would cause very high demands for particular resources that they couldn't meet with the existing network infrastructure. So they had two choices: either scale the network infrastructure to meet the peaks, which is very expensive and means that the majority of the network is under-utilized most of the time, or you build your network in such a way that it can reconfigure itself automatically to cope with those peaks, and channel the resources to meet the appropriate demand. And that is what Software Defined Network does. The system presents an effort to compose and test a network monitoring framework designed to capture and analyze potential network threats. It's a problematic method where the customer can alter the network according to their own business rules, to meet those peaks and demands at very short notice. So Software Defined Network is ideally suited for customers who have rapid changes in their day to day network load, such as social networking sites or maybe internet search engines. Large data centers, which have geographically dispersed resources and workloads in a specific location, may be in sudden high demand, and they have to reconfigure their network to meet the demands for that specific resource. There has been a lot of hype about software defined networking, and there is a lot of interest in it from all sorts of customers, but the reality is that it's really only suited for large data centers, or customers who have to have this rapid reconfiguration of their network.

The problem with Software Defined Network is that it demands you change your entire network infrastructure to implement the SDN protocol and SDN controller. Meaning you

have to completely reconfigure your network, retrain your staff, as well as learn new management tools, new diagnostic tools. This means that it is a huge change to your business, and the cost associated with that, for most customers, would not get the benefit in terms of performance gains. Developing and building such a software environment uses various applications that work in tandem. These software packages included Open vSwitch, OpenDaylight [19], Floodlight [23] and virtual machines.

The solution to this problem is a tool that will allow you to centrally manage your network without you having to reconfigure your network; without having to buy more network devices; without having to retrain your staff and incur all the costs associated with that. The control plane at the switch is realized by an OpenFlow — firmware responsible for the communication with the controller and for applying the updates at the data plane. The controller needs to keep track of what rules the switch has installed in the data plane. Any divergence between the view seen by the controller and the reality may lead to incorrect decisions and, ultimately, wrong network configuration. However, the protocol does not specify any positive acknowledgments that the update was performed [15]. The only way to deduce this information is to rely on the barrier command. As specified, after receiving a barrier request, the switch has to finish processing all previously-received messages before executing any messages after the barrier request. When the processing is complete, the switch must send a barrier reply message [2]. Switch data and control plane performance is essential for successful OpenFlow deployments, therefore it was a subject of measurements in the past.

During their work on the FlowVisor network slicing mechanism, Sherwood et al. [19] report switch CPU-limited performance of about few hundred OpenFlow port status requests per second. Similarly, as part of their work on the Devoflow modifications of the OpenFlow model [4], Curtis et al. identify and explain the

reasons for relatively slow rule installation rate on an HP OpenFlow switch. OFLOPS [18] is perhaps the first framework for OpenFlow switch evaluation. Its authors used it to perform fine-grained measurements of packet modification times, flow table update rate, and flow monitoring capabilities. This work made interesting observations, for example that some OpenFlow agents did not support the barrier command. OFLOPS also reported some delay between the control plane's rule installation and the data plane's ability to forward packets according to the new rule. Huanget al.[5] perform switch measurements while trying to build High-Fidelity Switch models that will be used during emulation with Open vSwitches

2.1 ODL AND OPENFLOW OVERVIEW

OpenFlow deployments are now also becoming a common occurrence both in datacenters and wide-area networks. Therefore, this means that the many SDN developers created many frameworks as well as network administrators that are using a variety of SDN controllers is rapidly increasing.

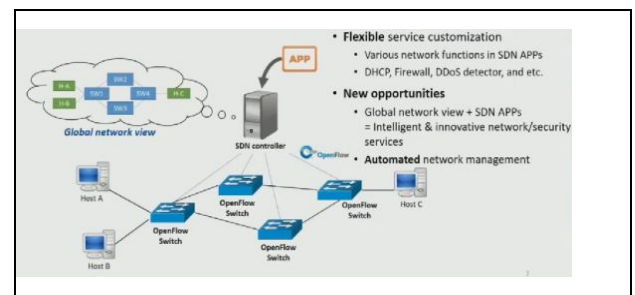


Figure 1: SDN controller overview

An OpenFlow is a protocol that allows access to the forwarding plane of a network switch or router over the network. This protocol can utilize multiple OpenFlow switches. The centralized controller manipulates the network by forwarding rules in the OpenFlow [17] switch flow tables. OpenFlow's frameworks are reliable and

performs accordingly that are necessary in the production environment.

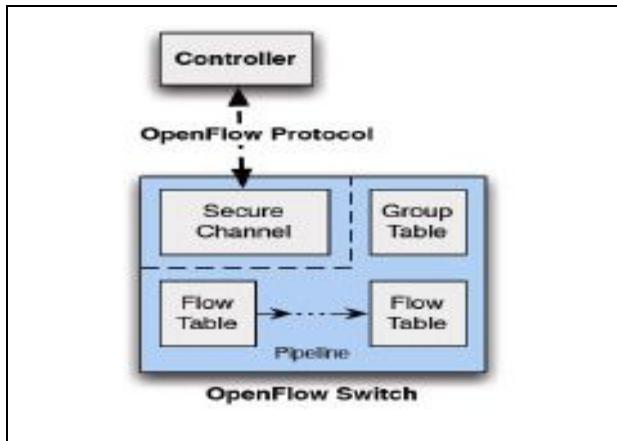


Figure 2: OpenFlow framework

The OpenFlow switch may be programmed to:

- (1) Identify and categorize packets from an ingress port based on various packet header fields.
- (2) Process the packets in various ways, including modifying the header.
- (3) Drop or push the packets to a particular egress port or to the OpenFlow Controller.

The OpenFlow transmits the controller to the switch which are structured as “flows”. Each flow contains a packet that match the fields, flow priority, various counters, packet processing instructions, flow timeouts, cookie and the flows are organized in tables. Any incoming packet will be processed by the flows in multiple pipelined tables before exiting on the port. The OpenFlow protocol standard is evolving quickly with release 1.3.2 as the current revision at the time of this blog being published.

Aside from using OpenFlow as the protocol. We will transition to discuss what the OpenDaylight [21] is now. OpenDaylight (ODL) is an open source project under the Linux Foundation with the innovation of SDN through the industry supported framework. ODL with memberships covering over 40 companies, such

as Cisco, IBM, NEC, etc. Projects focus on adding specific features to the controller. ODL [21] is as an open-source framework designed to “shape the future of SDN” mainly because it provides an open platform for developers to contribute, use, and, in fact, even build commercial products. Because it is open source with a large community it is a great foundation to test SDN and its potentials.

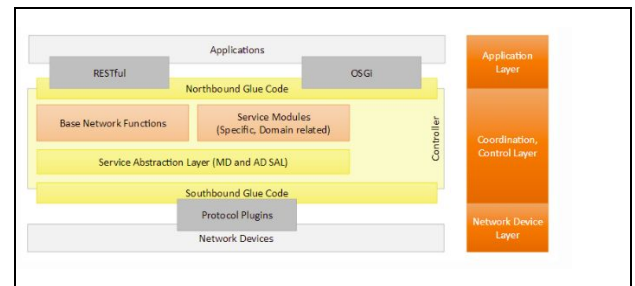


Figure 3: Layer Functions

Considering that OpenDaylight [21] includes multiple components and projects, it may help the reader if we first take a look at the simplified architecture, as shown above the architecture mainly comprises three layers:

1. Southbound plugins and protocols forming the network device layer.
2. Service adaptation and network functions forming the coordination and control layer.
3. Northbound APIs and applications forming the application layer.

The SDN controller acts like a middle man in the ecosystem. The framework glues together the applications requiring the services of the network devices and the protocols that talk to the network devices for extracting services. The controller programs the applications to control the network device’s specifications and allowing the application developers to concentrate on the development of application functionality rather than writing device-specific drivers.

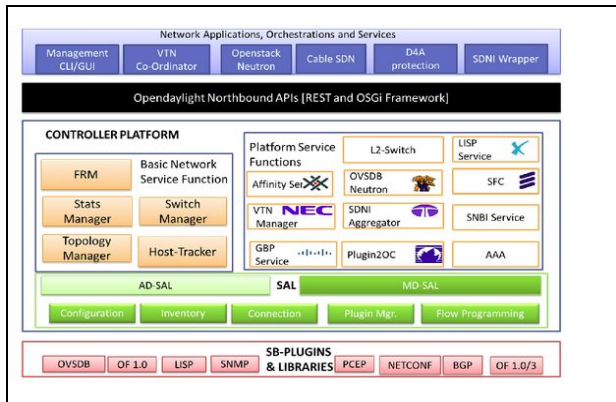


Figure 4: OpenDaylight API

The southbound interface is capable of supporting multiple protocols like OpenFlow 1.0, OpenFlow 1.3, BGP-LS, LISP, SNMP, etc. These modules are dynamically linked to a service abstraction layer (SAL), which determines how to fulfill the service requested by certain applications of the underlying protocol used between the controller and the network devices. In ODL [21], the service abstraction layer (SAL) is the key design that enables the abstraction of services between the services' consumers and producers. SAL acts like a large registry of services advertised by various modules and binds them to the applications that require them. Modules providing services, or producers, can register their APIs with the registry. When an application, or a consumer, requests a service from the API, SAL is responsible for requesting by binding producer and consumer into a contract, brokered and serviced by SAL. SAL has two architecturally different ways of implementing this registry: application-driven SAL and module-driven SAL.

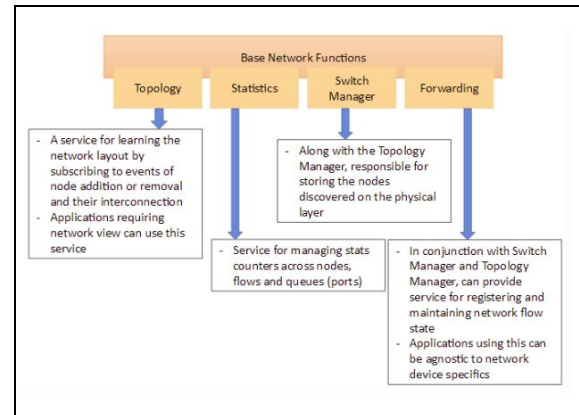


Figure 5: Network functions

The controller has basic network functions included as part of the shipped base. This includes services for topology discovery. The forwarding manager for managing basic forwarding rules, and a switch manager for identifying networking elements in the underlying physical topology. SAL acts as an active registry for brokering contracts between service providers, such as protocol plugins and basic network function plugins, and the service consumers, such as the applications. These contracts are honored by the SAL without any direct dependency on the respective plugins. For example [21], a topology service plugin is responsible for discovering nodes and physical paths between them to generate a graph. This plugin exposes functions that can be used by an application to get a complete view of the physical layer. Thereafter, the application can use the flow programmer service to create flows across all the network devices, without worrying about their make or model, to implement an end-to-end flow logic. The controller platform contains a collection of pluggable modules to perform needed network tasks.

The ODL controller includes application and service modularity, managed by the implementation of a service abstraction layer (SAL). The controller exposes open northbound APIs, which are used by applications. OpenDaylight, interestingly, supports both the OSGi framework and the bidirectional REST

APIs in the northbound layer. The OSGi framework is used by applications that will run in the same address space as the controller, whereas the REST (Web-based) API is used by applications that can run on the same machine as the controller or on a different machine.

3 MOTIVATION

SDN is here to stay, as evidenced by some of its users such as companies like AT&T, Comcast, Alibaba Group, CenturyLink and many more that are deploying or planning to deploy it [11]. With SDN growing in popularity, it's becoming increasingly important to understand what SDN actually is and to identify any potential threats to it. Users and companies alike need to become aware of the potential dangers they're opening themselves up to when they decide to implement an SDN for their services. SDN, like anything else, isn't 100% foolproof. One of its potentially glaring weaknesses is that the SDN controller itself is susceptible to attacks such as applications that seek to attack the controller. This central point of control is one of the key advantages of deploying an SDN but with that, it is also a single point of failure for the network [5]. If the controller is vulnerable, how feasible is it to raise one's permissions and pose as the owner/host of the controller? Are there any safeguards against access control attacks?

The way traditional networks are set up, we already know what security flaws they're predisposed to. When a company implements an SDN for their business, these flaws don't automatically go away. While making the switch makes it easier to monitor the network, existing problems such as spoofing or DDoS attacks still remain. This is, in part, due to the fact that the controller channel between the control and data planes aren't well protected [3], resulting in attacks such as DDoS to still exist in an SDN. We want to be able to implement strategies and defenses in order to safeguard a deployed SDN, which would make the transition of going away

from traditional networks safer. The easier it is for a network administrator to be able to diagnose network connectivity issues, the more likely SDNs will become the new adopted standard for networks both big and small.

4 APPROACH

We seek to setup an SDN controller from OpenDayLight, create the topology using Mininet [20]. Start our instrumented Floodlight [22] controller and prepare for the MAC spoofing attack. We attack its controller through the means of exploiting any potential holes in the access level control schemes that were put in place. We then aim to see if it's possible to be able to detect new permissions that are being granted without administrator approval and stop thwart an attacker before any damage can be done. If we're unable able to detect the attack before it succeeds, the next step is to try and implement new access control schemes that are strong against this kind of attack.

However, an elevated permissions attack isn't the only means to attack an SDN service or controller. In order to determine how secure SDNs are as a whole, we have to find and hopefully prevent as many exploits and we can. We aim to exploit the SDN by various other means such as a MAC Spoofing attack. Through these attacks, we'll be able to identify which are the best means of crippling an SDN and how best to prevent such attacks.

There are tools that exist that are able to scan an SDN and diagnose networking issues that may arise. This is possible with tools such as ForenGuard [4] and Defense4All [18]. We aim to use these tools to accurately diagnose any connectivity problems that should arise from our planned attacks. We will do this in order to not only gauge the effectiveness of these attacks on an SDN but to test the capabilities of these diagnostic tools.

4.1 Set up

For our project we tried to use MongoDB [22] as a database to store the results's activities from our experiment but we did not manage to get into the shell so were not able to store our results. Even though, we successfully manage to obtain the results we wanted [6]. As previously said, we ran Mininet to create a virtual small network for our Software Defined Networking controller. This could have been whether on the same virtual machine but using a different terminal or on a separate virtual machine via a downloaded iso file and we would have to SSH into it [5]. The host machine we used for our experiment is a quad-core Intel and eight threads Core i7 7700K up to 4.2Ghz CPU running Windows 10 and having VirtualBox with Ubuntu 14.04 for the processing of our experiment.

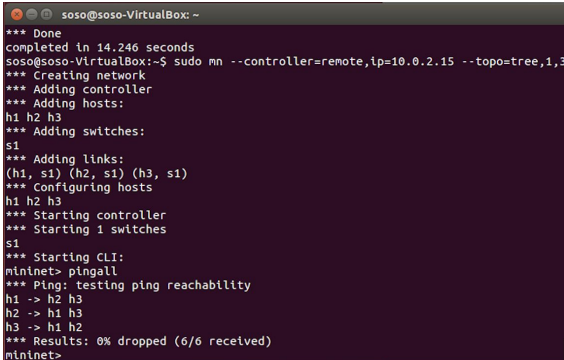
We have many methods to approach our research because SDN is in such a high demand, there's so many companies creating their own SDN with their own policies. We would like to try out various SDN environments such as OpenDayLight, Open Network Operating System, OpenContrail, Open vSwitch [17], Open Platform, and many more on github. We seek to setup an SDN controller from OpenDayLight, create the topology using Mininet. Start our instrumented Floodlight [22] controller and prepare for the MAC spoofing attack.

Once the SDNs are set up, we set up the various hosts as our variables, then set up the ethernet switch that allows networked devices, VMs, containers, and applications to communicate as if they all reside in the same physical data center or cloud region. Once the environment is set up, we will test an attack on the SDN controllers and the SDN applications in various techniques such as spoofing, tampering, repudiation, Information Disclosure and Elevation of Privileges. To further analyze SDN security policies with the attacks in progress, we will be using ForenGuard [4] which provides flow-level forensics and diagnosis functions in

SDN networks. ForenGuard [4] monitors and records the runtime activities and their causal dependencies involving both the SDN control plane and data plane and is able to quickly display causal relationships of activities and help to narrow down the range of suspicious activities that could be the root causes [4]. Because of this tool, we should be able to do various attacks to further diagnose several real control plane attacks.

4.1.1 Mininet

On a separate Ubuntu 14.04 based virtual machine or also doable on the same one, we'll install and run the Mininet application using the command “sudo mn --controller=remote, ip=10.0.2.15 --topo=linear, 1, 3”. Mininet is able to emulate a virtual network topology consisting of switches, controllers and hosts. In our case we decided to go with one controller, three switches, and one host. This testbed supports SDN and allows us the flexibility of creating custom topologies that we may need down the road for further testing. If needed, we could create hundreds of hosts and switches in a given topology. We will run this environment alongside another virtual machine running OpenDaylight.



```
soso@soso-VirtualBox: ~$ sudo mn --controller=remote,ip=10.0.2.15 --topo=tree,1,3
*** Done
completed in 14.246 seconds
soso@soso-VirtualBox:~$ sudo mn --controller=remote,ip=10.0.2.15 --topo=tree,1,3
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
*** Starting 1 switches
s1
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

Figure 6: The creation of three hosts and three switches with Mininet

4.1.2 OpenDaylight

The setup for OpenDayLight (ODL) involved a Ubuntu 14.04 based virtual machines. We first needed to install and set the JAVA_HOME path to the right directory in order for ODL to operate properly. Next we downloaded and installed OpenDayLight Beryllium version 4.2 from 2016 because of some contents that have been removed or modify with the newer version that we needed, in our case we had to install the features called l2switch-switch-ui to enable the OpenDayLight GUI on a browser and the other one is odl-dlux which is the feature to enable the graphic visual for ODL on the GUI. Once everything has been downloaded, we opened the terminal from the ODL directory and executed it via the command ```./karaf -of13``` which means using the OpenFlow protocol version 1.3. Once it starts running and the needed features are installed, we'll be able to access ODL's web GUI which will allow us to view and modify the network topology we've created using Mininet [20].

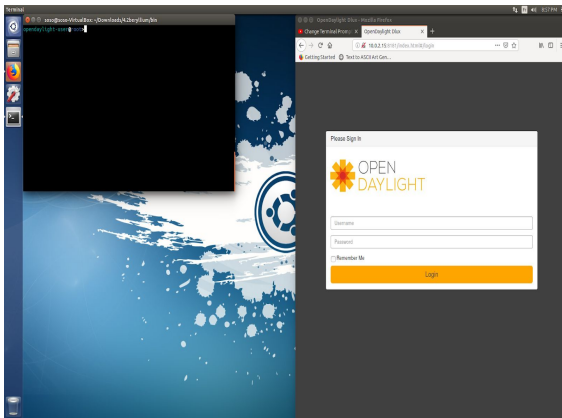


Figure 7: Accessing the ODL dlux web GUI to log into our topology

4.1.3 ForenGuard

ForenGuard [4] is packaged inside a virtual machine image using Ubuntu as its OS,

containing all the related applications necessary to create the network topology. As for ForenGuard [4] itself, it is a troubleshooting tool used for diagnosing network issues. The tool provides flow-level forensics and diagnosis functions in SDN networks [4]. What makes this tool unique is that it allows us to monitor issues that may arise on both the SDN control and data planes, where traditional tools are only able to monitor one or the other [4]. It has the capability to record and monitor activities done on both the control and data planes, and backtrack through them in order to detect the problem. With this tool, we will be able to detect a performed MAC spoofing attack on the network we created with Mininet [20].

```
root@floodlight:~# ifconfig
h1-eth0  Link encap:Ethernet  HWaddr fa:f3:c8:2a:3a:de
          inet addr:10.0.0.1  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::f8f3:c8ff:fe2a:3ade/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:78 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:14395 (14.3 KB)  TX bytes:648 (648.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

```
root@floodlight:~# ifconfig
h2-eth0  Link encap:Ethernet  HWaddr 32:fb:81:9b:61:85
          inet addr:10.0.0.2  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::30fb:81ff:fe9b:6185/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:77 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:14577 (14.5 KB)  TX bytes:648 (648.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@floodlight:~# ifconfig h2-eth0 down
root@floodlight:~# ifconfig h2-eth0 hw ether f6:aa:5c:de:ad:a6
root@floodlight:~# ifconfig h2-eth0 up
root@floodlight:~# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data:
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=10.5 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.122 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.032 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.031 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.026 ms
64 bytes from 10.0.0.1: icmp_seq=6 ttl=64 time=0.028 ms
64 bytes from 10.0.0.1: icmp_seq=7 ttl=64 time=0.038 ms
64 bytes from 10.0.0.1: icmp_seq=8 ttl=64 time=0.042 ms
^C
--- 10.0.0.1 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7007ms
rtt min/avg/max/mdev = 0.026/1.363/10.590/3.487 ms
```



```

root@floodlight:~# ifconfig
h3-eth0  Link encap:Ethernet  HWaddr f6:aa:5c:de:ada:6
          inet addr:10.0.0.3  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::f4aa:5c:ff:fe:de:ada:6/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:87 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:16034 (16.0 KB)  TX bytes:648 (648.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@floodlight:~# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data:
From 10.0.0.3 icmp_seq=1 Destination Host Unreachable
From 10.0.0.3 icmp_seq=2 Destination Host Unreachable
From 10.0.0.3 icmp_seq=3 Destination Host Unreachable
From 10.0.0.3 icmp_seq=4 Destination Host Unreachable
From 10.0.0.3 icmp_seq=5 Destination Host Unreachable
From 10.0.0.3 icmp_seq=6 Destination Host Unreachable
From 10.0.0.3 icmp_seq=7 Destination Host Unreachable
From 10.0.0.3 icmp_seq=8 Destination Host Unreachable
From 10.0.0.3 icmp_seq=9 Destination Host Unreachable
64 bytes from 10.0.0.3: icmp_seq=10 ttl=64 time=1006 ms
64 bytes from 10.0.0.3: icmp_seq=11 ttl=64 time=0.037 ms
^C
--- 10.0.0.1 ping statistics ---
11 packets transmitted, 2 received, +9 errors, 81% packet loss, time 10016ms
rtt min/avg/max/mdev = 0.037/503.163/1006.290/503.127 ms, pipe 3

```

Figures 8, 9 and 10: Post spoof attack

5 INITIAL RESULTS

Our expected outcomes for this paper are to successfully gather important information about the victims with the planned attacks and in doing so, we'll know what precautions to take against attacks on SDN and how to prevent them. Our goal is to detect the threats against the SDN controllers using ForenGuard [4] and how ForenGuard [4] might fail to diagnose certain attacks. We aim to discover what could be managed correctly to prevent future attacks from occurring again. Beyond that, our future expectations would be to successfully perform an attack on the SDN while remaining undetected and to gather relevant information about the victim, such as access to the controllers and applications. Having successfully tested SDN's security through our experiments and findings, we hope to be able to formulate feasible defenses against the planned attacks and show how the current implementation and structure of SDNs are or aren't safe overall.

5.1 Research Challenges

We originally intended on using a tool called ForenGuard [4], which provides flow-level forensics and diagnosis functions in SDN networks. Unfortunately we ran into troubles when attempting to download and utilize said tool. We tried to download it on multiple systems and operating systems, but the download failed near the halfway point. The original goal was to utilize ForenGuard [4] to monitor and record runtime activities and their causal dependencies on the SDN control plane. We wanted to use ForenGuard [4] to diagnose several real control plane attacks. To do this, we would have had to implement ForenGuard [4] on top of the Floodlight [22] controller and use it to diagnose several real control plane attacks. We show that ForenGuard [4] can quickly display causal relationships of activities and help to narrow down the range of suspicious activities that could be the root causes. Our performance evaluation shows that ForenGuard [4] will add minor runtime overhead to the SDN control plane and can scale well in various network workloads.

6 INITIAL CONCLUSIONS

Following our complications with ForenGuard [4], we decided to search for alternative diagnostic or attack detecting tools that we could test. We came across both Defense4All [18] and DELTA [8]; The former is an application used for detecting and mitigating DDoS attacks [18] while the latter is a tool for penetration testing [8]. DELTA is not only capable of testing against known attacks on SDN controllers but it's also able to detect new ones [8]. It is also capable of diagnosing the issues encountered in the network's connectivity.

6.1 Defense4All

As we set up our testing environment we have our client node, server node and our main

SDN controller by OpenDayLight [19] and we also have our Defense4All [18] application. Defense4All is an SDN application for detecting and to prevent DDoS attacks. The application communicates with the OpenDaylight Controller from the OpenDayLight north-bound REST API. Because of this, Defense4All [18] is able to perform two the following main tasks monitoring behavior of protected traffic and diverting attacked traffic to selected network locations to divert traffic to selected AMSs. When an attack is finished, the application removes these flow entries, and therefore returning to normal operation and traffic monitoring.

7 REVISED APPROACH

7.1 ForenGuard

Utilizing VMWare, we decided to set up another environment to test ForenGuard [4]. Using Ubuntu SMP as the operating system, with a 1.6 GHz Intel Core i5 processor and 8 GB of 1867 MHz DDR3 memory, we implemented ForenGuard [4] alongside MongoDB [2] and the ODL [19] controller. MongoDB will be used to store the actions that take place during the experiment. Upon starting the virtual machine, we first create a path for MongoDB to store the data. The creation of the topology is done with Mininet [20], and we use it to create three switches that connect to a single host (H1, H2, H3) each. All three of the hosts have a unique MAC and IP address given to them upon their creation and we use ‘ifconfig’ on each to ensure they’re all different. The attack begins by first taking H2 offline, manually altering H2’s MAC address by entering H3’s MAC address and then bringing H2 back online. With this, we have H2 send network packets over to H1 and make H3 attempt to do the same. H1 will receive H2’s packets but H3 will fail to make the connection. Using the ForenGuard [4] tool, we are able to diagnose why H3 failed to connect to H1. With the tool, we are able to trace the paths H3 and H1

took, resulting in us finding out what caused H3 failed attempts at reaching H1.

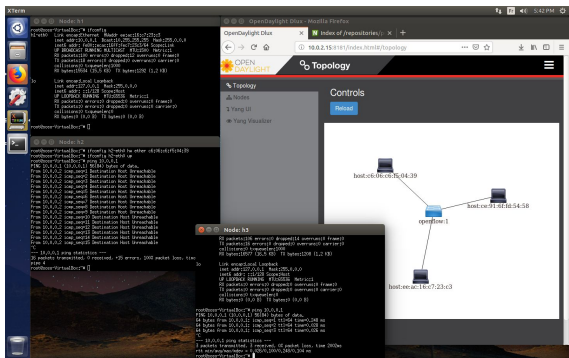


Figure 11: Successful spoof attack

7.2 OpenDaylight

After some initial complications with getting the necessary software to work properly, we succeeded in creating a network topology using Mininet [20] through OpenDaylight [19]. Through its web GUI, we were able to view the topology we wanted to create, which was having three switches and one host. However, ForenGuard [4] is packaged into a customized VM image and we were unable to apply it here. This network could, however, be used with Defense4All since it’s an application made to communicate with an OpenDaylight controller [18]. But despite our efforts, this application proved to be difficult to install and run at this time We decided to save the network we've already setup for future use.

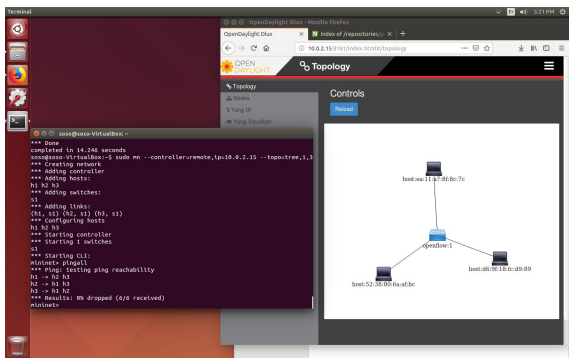


Figure 12: OpenDaylight web GUI topology with a three-way switch using OpenFlow

8 Evaluation

8.1 ForenGuard Revisited

After experiencing initial our difficulties with ForenGuard [4], we reached out to Professor Gu from the Texas A&M University again and managed to finally get more information about how to properly use ForenGuard [4]. It turns out that to successfully used Professor Gu's tool, we needed to run every command as sudo. So with this update in hand, we continued to implement our initial plan of using ForenGuard [4] to detect a MAC spoofing attack as previously presented in this paper. Following the attack, we ran ForenGuard to diagnose why host H3 has failed to connect to host H1.

```
OFSwitchBase DPID[00:00:00:00:00:00:02]:
-> Receive Event OFPacketInVer13
-> Init Function processPacketInMessage
-> Init Function addToPortMap
-> Write Variable macVlanToSwitchPortMap -254649434
-> Return Function addToPortMap
-> Init Function getFromPortMap
-> Read Variable macVlanToSwitchPortMap -517582621
-> Return Function getFromPortMap
-> Init Function pushPacket
-> Return Function pushPacket
-> Init Function writeFlowMod
-> send Message FLOW_MOD
-> Return Function writeFlowMod
-> Init Function writeFlowMod
-> send Message FLOW_MOD
-> Return Function writeFlowMod
-> Return Function processPacketInMessage
-> Return Function receive

OFSwitchBase DPID[00:00:00:00:00:00:02]:
-> Receive Event OFPacketInVer13
-> Init Function processPacketInMessage
-> Init Function addToPortMap
-> Write Variable macVlanToSwitchPortMap -517582621
-> Return Function addToPortMap
-> Init Function getFromPortMap
-> Read Variable macVlanToSwitchPortMap null
-> Return Function getFromPortMap
-> Init Function writePacketOutForPacketIn
-> Return Function writePacketOutForPacketIn
-> Return Function processPacketInMessage
-> Return Function receive
```

```
OFSwitchBase DPID[00:00:00:00:00:00:02]:
-> Receive Event OFPacketInVer13
-> Init Function processPacketInMessage
-> Init Function addToPortMap
-> Write Variable macVlanToSwitchPortMap -254649434
-> Return Function addToPortMap
-> Init Function getFromPortMap
-> Read Variable macVlanToSwitchPortMap -517582621
-> Return Function getFromPortMap
-> Init Function pushPacket
-> Return Function pushPacket
-> Init Function writeFlowMod
-> send Message FLOW_MOD
-> Return Function writeFlowMod
-> Init Function writeFlowMod
-> send Message FLOW_MOD
-> Return Function writeFlowMod
-> Return Function processPacketInMessage
-> Return Function receive

OFSwitchBase DPID[00:00:00:00:00:00:02]:
-> Receive Event OFPacketInVer13
-> Init Function processPacketInMessage
-> Init Function addToPortMap
-> Write Variable macVlanToSwitchPortMap -517582621
-> Return Function addToPortMap
-> Init Function getFromPortMap
-> Read Variable macVlanToSwitchPortMap null
-> Return Function getFromPortMap
-> Init Function writePacketOutForPacketIn
-> Return Function writePacketOutForPacketIn
-> Return Function processPacketInMessage
-> Return Function receive
```

Figures 13 and 14: Diagnosis results

8.2 Future Work

After performing the MAC spoofing attack, we were able to diagnose why node H3 lost connection to H1 using ForenGuard [4]. However, ForenGuard [4] is limited in nature and is unable to diagnose an extensive list of attacks in its current form. Therefore, we aim to find what the limits of ForenGuard [4] are by performing a wider array of attacks against the Mininet network we've created. Once found, we aim to put defenses in place that would prevent these attacks from being successful.

One such attack that we'd like to implement is attacking the SDN controller with a malicious application and gaining control over it as a result. This is one of the biggest threats to an SDN because a malicious party that has access to the controller could wreak havoc on the entire network. Another attack that we would like to implement is a DDOS attack against the current SDN controller. This would allow us to If successful, we'd like to implement the same attack(s) but put them up against another SDN

diagnostic tool such as DELTA [8] or Defense4All [18].

Alternatively, we will try to either improve and modify the existing ForenGuard [4] code or try to create a new but similar diagnostic tool from the ground up. ForenGuard [4] is a tool with plenty of potential and we'd like to push the capabilities of what it can do. If that's not possible, a successful development of our own diagnostic will demonstrate that it's feasible for a company, that has many more resources at its disposal than ourselves, to develop their own. If that happens, it's possible tools like ForenGuard [4] and DELTA [8] can become open source for all to use or could be sold so companies that have deployed an SDN have a means to diagnose their network right away. This wouldn't solve the security issues that are present in the current state SDNs but it'd be another step in securing safer SDN environments.

9 Related Work

Digital Forensics: In the past decade, there has been a shift in focus of research of digital forensics. The well studied research topic is now focused on network-level forensics handling large amount of data (storing, indexing and retrieval) in large-scale, complex networks. TimeMachine [23] records raw network packets and builds the index for the headers of the likely-interesting packets [4]. While monitoring the data plane state, Anteater [24] checks if the state violates specified invariants by using formal analysis to . Teryl et al. proposed a storage system [39] to efficiently build the index of payload information of network packets. VAST [25] is a platform that uses the actor model to capture different levels of network activities and provides a declarative language for query. Another relevant research topic in recent years is Network provenance [26]. The basic foundation of ForenGuard is similar to network provenance, which is to track causality and capture diagnostic data at runtime that can be queried later [4]. While existing tools mostly

target declarative languages, ForenGuard can directly work on the general-purpose programming language [29]. On host-level forensics, Forenscope [29] proposes a framework that can investigate the state of a running operating system without using taint or causing blurriness. BackTracker [30] creates a dependency graph for intrusion detection from recorded files and processes in the operating system. Different from all above work, ForenGuard focuses on a unique context of SDN which decouples control and data planes and also requires both network and host level tracking [4].

SDN Security: SDN security has increasingly become a trending research, where it has been determined that attackers can utilize rootkit techniques to subvert SDN controllers [31]. DELTA [8] presents a fuzzing-based penetration testing framework to find unknown attacks in SDN controllers. TopoGuard [7] proposes mitigation approaches to fortify SDN control plane, as well as also pinpointing two new attack vectors against SDN control plane that can poison network visibility and mislead further network operation. In contrast to existing threats, in this paper we study a new threat to the SDN, i.e., harmful race conditions in the SDN control plane. To fortify SDN networks, AvantGuard [27] and FloodGuard [28] propose schemes to defend against unique Denial-of-Service attacks inside SDN networks. FortNOX [32] and SE-FloodLight [33] provide several security extensions in an attempt to prevent malicious applications from violating security policies enforced in the data plane. SPHINX [34] presents a novel model representation, called flow-graph, to detect several network attacks against SDN networks. Sandbox strategies are proposed by Rosemary [35] to protect SDN control plane from malicious applications. Although some impacts, introduced by the harmful race conditions, such as a system crash, could be isolated, they are not designed to detect those concurrency flaws.

DoS Attacks Against SDN: To mitigate a dedicated DoS attack, AVANT-GUARD [27]

introduces connection migration and actuating triggers to extend the data plane functions. However, it is applicable to TCP protocol only. FloodGuard [28], a protocol-independent defense framework, pre-installs proactive flow rules to reduce table-miss packets and forwards table-miss packets to additional data plane caches. FloodDefender [36] offloads table-miss packets to neighbor switches and filters out attack traffic with two phase filtering, therefore providing the benefit of no hardware modification and addition. Though both attack methods and attack effect, Control Plane Reflection Attacks distinguish themselves from previous works in. The MAC spoofing attack uses a pretty straightforward method that just floods the traffic to trigger the direct data plane events while the reflection resorts to techniques, and a probing-trigger approach is specially developed to exploit both direct and indirect data plane events. Since the attack is not hard to capture, therefore it could have limited attack effects. Because the reflection attacks are more than stealthy and the same attack causes the attacker could be more obvious attack effects for victims.

10 CONCLUSION

We described the design and implementation of Opendaylight, an open source, multi-platform OpenFlow protocol based Software Defined Network and what possible vulnerabilities the controller can manifest. Opendaylight has simple origins but its performance has been gradually optimized to match the requirements of multi-tenant datacenter workloads, which has necessitated a more complex design. Given its operating environment, we anticipate no change of course but expect its design only to become more distinct from traditional network appliances over time.

One of the main advantages of this solution is its flexibility. Any parameter in the solution can be modified to fit the requirements of the controller. Detection field (i.e. destination IP

address), window size and threshold can all be set to correspond to the desired values even in real-time. This is all possible through the controller's interface. To the best of our knowledge, this is the first work that uses entropy for spoofing detection in SDN controller and the first work to, specifically, address MAC spoofing attacks on the controller. To mitigate this threat, this paper proposes to use the central control of SDN for attack detection and introduces a solution that is effective and lightweight in terms of the resources that it uses. More precisely, this paper aims to show how MAC spoofing attacks can exhaust controller resources and provides a solution to detect such attacks based on the entropy variation of the destination IP address.

11 ACKNOWLEDGMENTS

This material is based in part upon work supported by Professor Guofei Gu of Texas A&M University, so we would like to thank him for his support. All opinions, findings and conclusions or recommendations expressed herein are those of the author(s) and do not necessarily reflect the views of the John Jay College or its faculty.

REFERENCES

- [1] M. Dacier, H. Konig, R. Cwalinski, F. Kargl and S. Dietrich, "Security Challenges and Opportunities of Software-Defined Networking", *IEEE Security & Privacy*, vol. 15, no. 2, pp. 96-100, 2017. Available: 10.1109/msp.2017.46.
- [2] "What Is MongoDB?," *MongoDB*. [Online]. Available: <https://www.mongodb.com/what-is-mongodb>. [Accessed: 08-Dec-2019].

- [3] J. Cao, Q. Li, R. Xie, H. Sun, G. Gu, M. Xu, and Y. Yang, "The CrossPath Attack: Disrupting the SDN Control Channel via Shared Links"
- [4] H. Wang, G. Yang, p. Chinprutthiwong, L. Xu, Y. Zhang, and G. Gu, "Towards Fine-grained Network Security Forensics and Diagnosis in the SDN Era", 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), Available: 10.1145/3243734.3243749
- [5] S. M. Mousavi and M. St-Hilaire, "Early detection of DDoS attacks against SDN controllers," 2015 International Conference on Computing, Networking and Communications (ICNC), 2015. Available: 10.1109/iccnc.2015.7069319
- [6] M. Kuzniar, P. Perešini, D. Kostic, "What you need to know about SDN control and data planes", EPFL Technical Report EPFL-REPORT-199497
- [7] S. Hong, L. Xu, H. Wang, G. Gu, "Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures", 2015 Network and Distributed System Security Symposium, 2015. Available: 10.14722/ndss.2015.23283
- [8] S. Lee, et al. "DELTA: A Security Assessment Framework for Software-Defined Networks", 2017 The Network & Distributed System Security Symposium (NDSS).
- [9] O. Abdullaziz, L. Wang, "Mitigating DoS Attacks against SDN Controller Using Information Hiding", 2019 IEEE Wireless Communications and Networking Conference (WCNC), Available: 10.1109/WCNC.2019.8885764
- [10] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, M. Tyson, "FRESCO: Modular Composable Security Services for Software-Defined Networks", ISOC Network and Distributed System Security Symposium, 2019.
- [11] C. Yoon, T. Park, S. Lee, H. Kang, S. Shin, and Z. Zhang, "Enabling security functions with SDN: A feasibility study," *Computer Networks*, vol. 85, pp. 19–35, 2015.
- [12] S. Shin, L. Xu, S. Hong, and G. Gu, "Enhancing Network Security through Software Defined Networking (SDN)," *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, 2016.
- [13] L. Xu, J. Huang, S. Hong, J. Zhang, and G. Gu, "Attacking the Brain: Races in the SDN Control Plane", 26th USENIX Security Symposium, 2019, Available: 978-1-931971-40-9.
- [14] R. Skowrya, L. Xu, G. Gu, V. Dedhia, T. Hobson, H. Okhravi, and J. Landry, "Effective Topology Tampering Attacks and Defenses in Software-Defined Networks," 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2018.
- [15] M. Zhang, G. Li, L. Xu, J. Bi, G. Gu, and J. Bai, "Control Plane Reflection Attacks in SDNs: New Attacks and Countermeasures," *Research in Attacks, Intrusions, and Defenses Lecture Notes in Computer Science*, pp. 161–183, 2018.
- [16] "The basics of SDN and the OpenFlow Network Architecture," NoviFlow, 14-Apr-2019. [Online]. Available: <https://noviflow.com/the-basics-of-sdn-and-the-openflow-network-architecture/>. [Accessed: 08-Dec-2019].
- [17] B. Pfaff, J. Pettit, T. Koponen, et al, "The Design and Implementation of Open vSwitch", 12th USENIX Symposium on Networked Systems Design and Implementation" Available: ISBN 978-1-931971-218.
- [18] "Defense4All: Overview - OpenDaylight Project", Wiki.opendaylight.org, 2019. [Online]. Available: <https://wiki.opendaylight.org/view/Defense4All:Overview>. [Accessed: 07- Dec- 2019].
- [19] "Platform Overview - OpenDaylight", OpenDaylight, 2019. [Online]. Available: <https://www.opendaylight.org/what-we-do/odl-platform-overview>. [Accessed: 07- Dec- 2019].
- [20] Mininet. Rapid Prototyping for Software Defined Networks.

- <http://yuba.stanford.edu/foswiki/bin/view/OpenFlow/Mininet/>.
- [21] S. Rao, "SDN Series Part Six: OpenDaylight, the Most Documented Controller," *The New Stack*, 02-Jul-2019. [Online]. Available: <https://thenewstack.io/sdn-series-part-vi-opendaylight>. [Accessed: 08-Dec-2019].
 - [22] "Floodlight OpenFlow Controller -," *Project Floodlight*. [Online]. Available: <http://www.projectfloodlight.org/floodlight/>. [Accessed: 08-Dec-2019].
 - [23] G. Maier, R. Sommer, H. Dreger, A. Feldmann, V. Paxson, and F. Schneider. 2008. Enriching Network Security Analysis with Time Travel. In Proceedings of the ACM SIGCOMM 2011 Conference (SIGCOMM)
 - [24] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, and S. T. King. 2011. Debugging the Data Plane with Anteater. In Proceedings of the ACM SIGCOMM 2011 Conference (SIGCOMM).
 - [25] M. Vallentin, V. Paxson, and R. Sommer. 2016. VAST: A Unified Platform for Interactive Network Forensics. In Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)
 - [26] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr. 2011. Secure Network Provenance. In Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP).
 - [27] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. 2013. AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-Defined Networks. In Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS).
 - [28] Haopei Wang, Lei Xu, and Guofei Gu. 2015. FloodGuard: A DoS Attack Prevention Extension in Software-Defined Networks. In Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN).
 - [29] E. Chan, S. Venkataraman, F. David, A. Chaugule, and R. Campbell. 2010. Forenscope: a framework for live forensics. In Proceedings of the 2010 Annual Computer Security Applications Conference (ACSAC).
 - [30] S. T. King and P. M. chen. 2003. Backtracking intrusions. In Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP).
 - [31] RPKE, C., AND HOLZ, T. SDN Rootkits: Subverting Network Operating Systems of Software-Defined Networks. In RAID'15 (2015).
 - [32] PORRAS, P., SHIN, S., YEGNESWARAN, V., FONG, M., TYSON, M., AND GU, G. A Security Enforcement Kernel for OpenFlow Networks. In HotSDN'12 (2012).
 - [33] PORRAS, P., CHEUNG, S., FONG, M., SKINNER, K., AND YEGNESWARAN, V. Securing the Software-Defined Network Control Layer. In NDSS'15 (2015).
 - [34] DHAWAN, M., PODDAR, R., MAHAJAN, K., AND MANN, V. SPHINX: Detecting security attacks in software-defined networks. In NDSS'15 (2015).
 - [35] SHIN, S., SONG, Y., LEE, T., LEE, S., CHUNG, J., PORRAS, P., YEGNESWARAN, V., NOH, J., AND KANG, B. Rosemary: A Robust, Secure, and High-Performance Network Operating System. In CCS'14 (2014).
 - [36] G. Shang, P. Zhe, X. Bin, H. Aiqun, and R. Kui, "FloodDefender: Protecting data and control plane resources under SDN-aimed DoS attacks," *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017.