Causal Analysis for Software-Defined Networking Attacks

Ankita Singh
Clemson University

Mayank Patel
Clemson University

Samuel Murtaugh
Clemson University

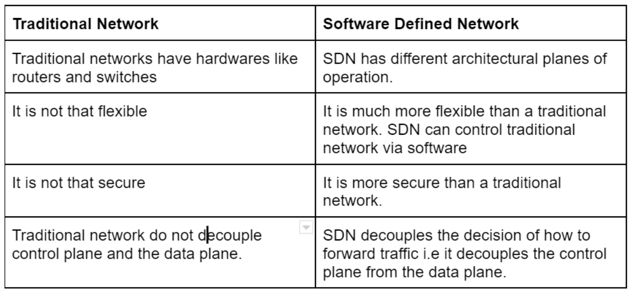
Abstract

Software Defined Networking (SDN) has been emerging as a flexible network architecture. The flexible nature of SDN helps in doing all required setup automatically instead of manually. Although SDN is flexible and improves network security oversight and policy enforcement, ensuring the security of SDN from sophisticated attacks is an ongoing challenege. The existing tools try to identify these attacks,but the holistic analysis of the attacks still remains a challenge.
The challenges may arise due to several factors like dependency explosion, not able to make sense of the end to end attack. We decided to look into this problem and hence We picked the paper "Causal Analysis of Software Defined Network" . In this paper Authors present a tool called PICOSDN which tries to make sense of the attack on SDN's and addresses the challenegs that existing tools have.

1 Introduction

Over the past decade, the software-defined networking (SDN) architecture has snowballed as a result of its flexibility and programmability. "Software defined network is an approach to networking that uses software based controllers or Application programming interfaces(APIs) to communicate with underlying hardware infrastructure and direct traffic on a network." If a website is expecting a flood of traffic to its servers . In order to accommodate all of the traffic in its datacenter some virtual servers need to be configured. Some load balancers need to be configured in order to balance the load. All this setup needs to be done. The software defined network helps to configure all of these setup through a programme. The programme helps to spin up the virtual machines,reconfigure the quality server,etc.

Difference between traditional network and Software Defined network.



Traditional Network	Software Defined Network
Traditional networks have hardwares like routers and switches	SDN has different architectural planes of operation.
It is not that flexible	It is much more flexible than a traditional network. SDN can control traditional network via software
It is not that secure	It is more secure than a traditional network.
Traditional network do not decouple control plane and the data plane.	SDN decouples the decision of how to forward traffic i.e it decouples the control plane from the data plane.

Figure 1: Difference Between Traditional and Software Defined Network

1.1 How Does SDN(Software Defined Networking) work?

SDN consists of various architectural planes

1. Data plane - Data plane is responsible for forwarding packets. A packet comes in on a router or a frame comes in on a switch. That router is going to contact the ip routing table or that frame is going to contact the MAC address table.

2. Control plane- Control plane is the place where protocols run on the router.Things that populate the ip routing table leave at control plane.

3. Management plane- Using the management plane the network engineers connect into the device. This is how the network engineers telnet or secure shell into the device.

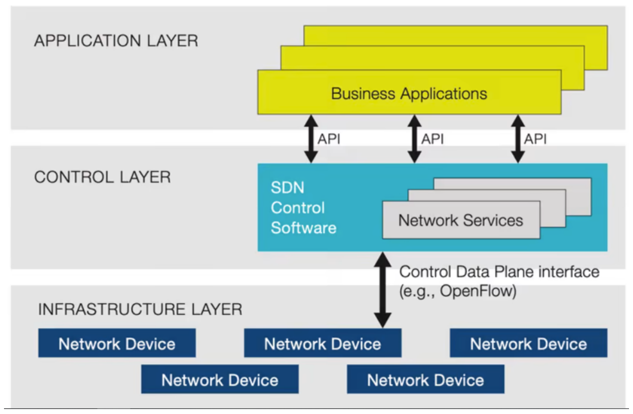


Figure 2: SDNArchitecture

How does SDN change the existing Landscape. SDN is used in combination with a network controller. This takes us from now what we have as a distributed control plane and moves it into the network controller. So it does not remain distributed anymore. This network controller propogates appropriate information to individual devices.When we are communicating from the network controller down to the devices. APIs are used in order to communicate the network controller down to the devices.

An API is a piece of software that talks to another piece of software. In case of SDN we have API's in the controller that talks to the API's in the devices. In order to interact they both speak the same SDN protocol. In the SDN controller there is are northbound and southbound interfaces. The southbound API's are used to connect to the devices. The control plane in SDN is centralized. The north of the SDN controller are the applications. The kind of APIs used in SDN are called restful APIs.

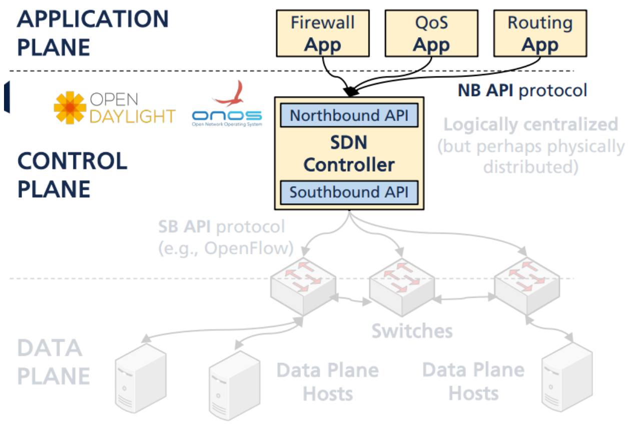


Figure 3: North and southbound API

Though SDN is flexible it is also vulnerable to certain security attacks

1. Cross app poisoning - Malicious applications can poison the view of other applications.

2. Cross plane vulnerabilities - Malicious hosts can influence decision making by taking advantage of cross plane information.

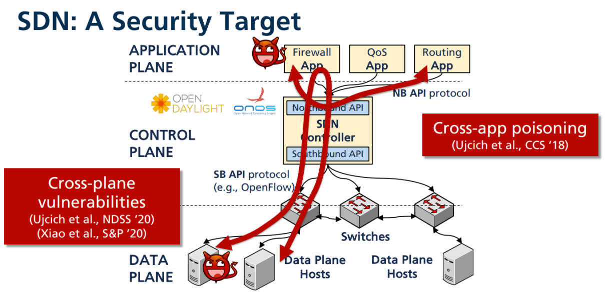


Figure 4: Attacks

2 Background and Motivation

SDN controllers are susceptible to attacks from data plane hosts as well as malicious applications. If a SDN network is attacked then the network administrator would like to know certain information about the attack such as follows

1. What were the significant control plane ,app plane and data plane actions that were taken.

2. Adminstrartors would also like to see the attack from end to end and if they are able to see the attack from end to end still they would need to search through hundreds and thousands of system log files.

3. What happened in the past that led to the current decisions?

4. Where else in the network did the attack affect?

Most of these questions can be answered through data provenance. Data provenance is basically all formatted data that we care about that is generated or used. That includes systems principlas or agents in other words who or what was responsible processes or activities and data objects. The relationship among all these is modeled using DAG(Directed acyclic graph) that shows the history. The key benefit is we can search past history quite efficiently.

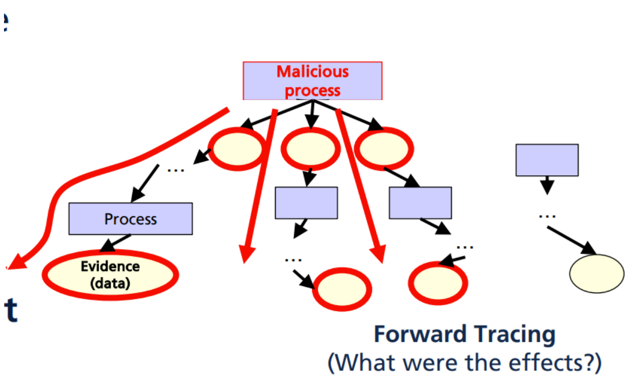


Figure 5: Data Provenance

As we can see in the above figure we have a piece of evidence that an attack has occurred. This piece of evidence was generated by a process and back in its history there is a root cause of a malicious process. We can find out the root cause by backward tracing. If we want to find out the effects we can do it by forward tracing. This way we can get all the history. We can also ignore all the irrelevant history. This sounds useful but it is quite hard to do in SDN context.

Following are the limitations of the above proposed method

1. Dependency explosion – If we are not careful in modeling the provenance graph the every network activity becomes dependent on every other network activity. The API centric model that SDN follows would create many false dependencies because and API call would be falsely dependent on all previous API calls. FORENGUARD's event centric model suffers from this limitation.

2. Incomplete dependencies - If we don't capture the causal relations among the different planes. We might end up missing some dependencies that might end up being crucial. In case of SDN attacks this occurs when data plane's effects on the control plane are not captured by an implicit data plane model.Most of the popular SDN controller's like Floodlight ,openDayLight suffer from this limitation.

3. Attribution and Responsibility - The network identifiers like MAC are easily spoofable assigning responsibility to the host would not solve this problem either as malicious host would simply induce false dependencies in the provenance graph.

4. Interpretation - If all the dependency related challenges are mitigated it is still quite difficult to interpret the provenance graph. The tools like ProvSDN and FORENGAURD use backward tracing to find the root causes. However, if the network administrator would require some other information which requires forward tracing that would be difficult to achieve in these two tools.

2.1 How does PicoSdn addresses these limitations

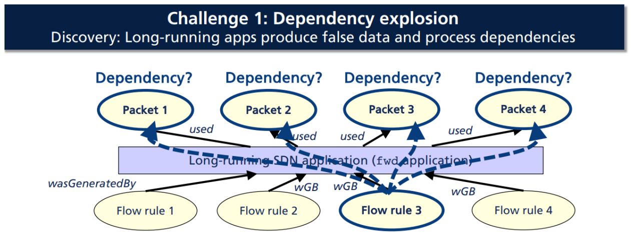


Figure 6: Dependency Explosion

1. Dependency Explosion - Suppose the SDN app is modeled as a long running process. In the figure above we have a fwd application that is represented by a rectangle. It uses a packet and generates a flow rule. So to interpret this we say that the flow rule 1 was generated by the forwarding application.Which used packet 1. This process keeps on repeating for the incoming packets and generates more flow rules. If a network administrator spots a flow rule and wants to trace its history. Suppose , the administrator spots flow rule 3 and wants to trace its history. He sees that every packet input is involved as a result dependency explosion takes place.

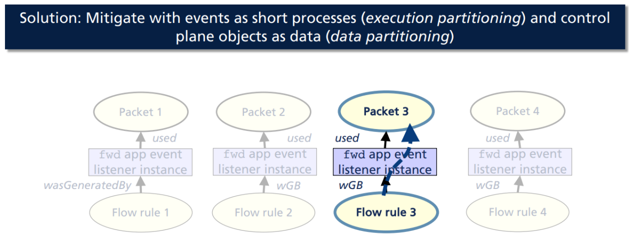

Solution: Mitigate with events as short processes (execution partitioning) and control plane objects as data (data partitioning)
Packet 1
Packet 2
Packet 3
Packet 4
used
used
used
used
fwd app event listener instance
fwd app event listener instance
fwd app event listener instance
fwd app event listener instance
wasGeneratedBy
wGB
wGB
wGB
Flow rule 1
Flow rule 2
Flow rule 3
Flow rule 4

Figure 7: Dependency Explosion solution

In order to fix the problem PicoSdn partitions the objects and processes. This mans that the execution partitioning and data partitioning is done. As the event listeners and applications are loops they can be partitioned very easily. Now when the administrator traces back on the flow rule 3 . He sees that only packet 3 is responsible.

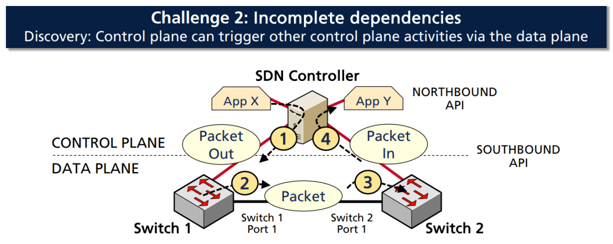


Figure 8: Incomplete dependency

2. Incomplete dependencies - If we look at the above figure we can see that we have two applications X and Y. App X sends a packet out into the data plane. The packet is sent with an instruction for all the other ports in switch 1. One of those packets goes towards switch 2. Switch 2 sees that packet as an incoming packet that doesn't match any incoming flow rule that it has. It sends that packet out to the controller for processing where App Y sees it . It is clear that App X had something to do with the packet ,but we cannot see that from the control plane alone. So this problem is mitigated by combining the control and data plane information

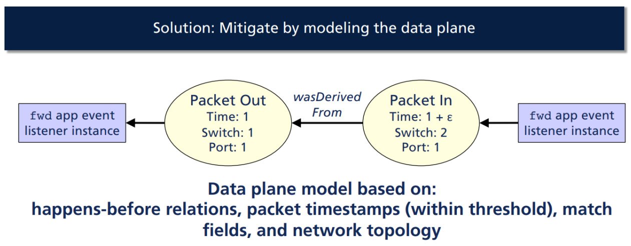

Solution: Mitigate by modeling the data plane
fwd app event listener instance
Packet Out
Time: 1
Switch: 1
Port: 1
wasDerived From
Packet In
Time: 1 + ε
Switch: 2
Port: 1
fwd app event listener instance
Data plane model based on:
happens-before relations, packet timestamps (within threshold), match fields, and network topology

Figure 9: Incomplete dependency solution

3. Attribution and Responsibility - When a switch is added into the network. The controller adds a default flow rule that matches the packet that otherwise haven't been matched. This action is taken so that the unmatched packets do not get dropped. We can see in the image below that we have incoming packets that match the default flow rule and is then sent to the controller.

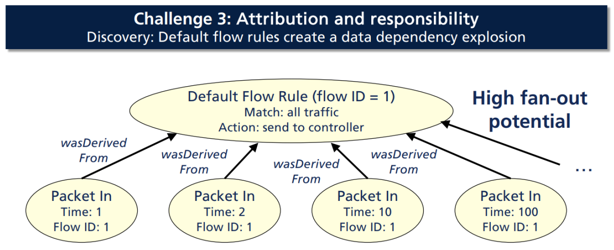


Figure 10: Attribution and Responsibility

If there are running switches it creates a problem for them as everything depends on the default flow rule which causes high fan out as a result. In order to mitigate this a responsibility is assigned to the switch port. This groups dependency as each switch has the responsibility for the packets that are sent to them

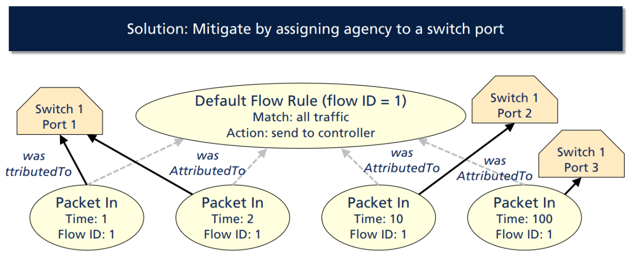


Figure 11: Attribution and Responsibility SOlution

As we can see in the above figure that starting from the control plane. We see that a packet goes out from the app at time 1 and some short time later to time 1. We see that an incoming packet is being processed by some other app or another instance of the same app. In data plane model identifies where this causal link should be added. This way we can capture indirect control plane causality.

4. Interpretation - If we have collected all the information. We need to make sense of all the collected information. Though it is better than going through verbose sys logs files. Still the information can be overwhelming.

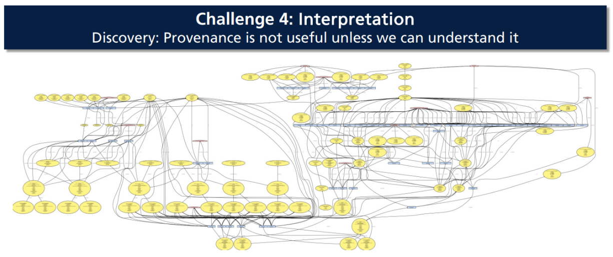

Challenge 4: Interpretation
Discovery: Provenance is not useful unless we can understand it

Figure 12: Interpretation

PicoSdn overcomes this challenge by providing a set of tools that help network administrators Make sense of the attack

Some of the algorithms used in PicoSdn are

(a) Common ancestry - It helps in figuring out what nodes are common ancestors.

(b) Backward - forward - If a path is given between an evidence and root. It helps in figuring out how does the ancestry look at each stage.

(c) Activity summary - This helps in figuring out how does the data plane impacts the flow rule.

(d) Indentifier evolution - This helps in figuring out how do hosts change identity over time.

3 PicoSdn Architecture

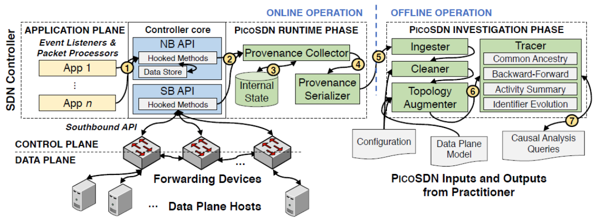


Figure 13: PicoSdn Architecture

PicoSdn has been implemented on the ONOS SDN controller. The network applications and the data plane both use APIs in order to interact with the controller's control plane. Events have been wrapped to identify how dispatchers are being used by listeners.API calls are hooked so that objects can be tagged throughout the call . The internal state is kept as unique identifier for each object. The provenance is then serialized for a concise representation. Some time later offline the provenance is deserialized. This involves converting the serialized representation to a graph that can be opearted on. The graph is then cleaned up and augmented in order to account for the information that is needed in the attack analysis. At last the tracing techniques are used to analyse the graph.

4 Related Work

SDN was introduced out in the world a decade ago [3] and attacks like cross-plane attacks have been bypassing network security restrictions. Several research studies have been done on understanding SDN vulnerabilities and developed tools to analyze these vulnerabilities. Several studies introduced vulnerability detection and analysis tools to understand and mitigate those attacks from happening again.

4.1 ForenGuard

ForenGuard is a tool developed by Wang et al. [6] for providing flow-level forensic and diagnosis functions in SDN networks. The reason behind developing ForenGuard is to investigate the activities of the SDN framework and record those runtime activities and use those recorded activities to mitigate network security problems for the SDN control plane and data plane.

ForenGuard does not disrupt any of the normal operations of other controller applications because it works and runs on the top of the SDN control plane. As shown in the architecture **figure 14**, Forenguard has three parts:

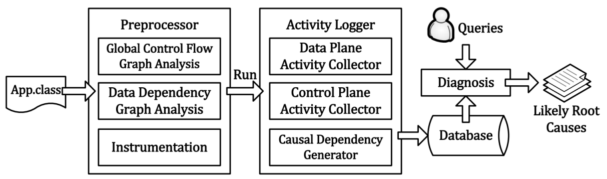

Preprocessor
Global Control Flow Graph Analysis
Data Dependency Graph Analysis
Instrumentation
App.class
Run
Activity Logger
Data Plane Activity Collector
Control Plane Activity Collector
Causal Dependency Generator
Queries
Diagnosis
Database
Likely Root Causes

Figure 14: *ForenGuard Architecture/Design*

- **Preprocessor:** The main goal of the Preprocessor is divided into three parts:

  1. Using static analysis to extract activities - which means that ForenGuard statically identifies the state variables, analyzes the data flows, and instruments the read/write operations of the activities [6],

  2. Generating data dependency graphs - to generate data dependency graph, ForenGuard performs context-sensitive and field-sensitive on controllers, and

  3. Instrumenting the controller - at runtime, this identify and profile essential operations of the control plane.

- **Activity Logger:** Once the Preprocessor is done with its goal, the Activity logger act as a controller component and collects activities from the control plane and data plane for causal dependency relationships. To do that, it divides into three submodules:

    1. Data Plane Activity Collector - collects runtime data plane activities,
    2. Control Plane Activity Collector - collects runtime control plane activities, and
    3. Causal Dependency Generator - develops causal dependency relationship of collected data and add it to the database.

- **Diagnosis:** Diagnosis is done on a command line generated by Wang et al. [6], and it includes problems like reachability, isolation, routine loop, and way-point routing [6].

ForenGuard was developed as the first SDN forensic and diagnosis tool that integrates the control plane and data plane. Furthermore, it can backtrack all the previous activities in the control plane and data plane and do causal analysis to find the root cause of the attack or problem.

4.2 EVENTSCOPE:

EVENTSCOPE is one of the vulnerability detection tools that was developed using a systematic approach by Ujcich et al. [5] to identify and analyze cross-plane vulnerabilities systematically. EVENTSCOPE automatically analyzes SDN control plane event usage, discovers candidate vulnerabilities based on missing event-handling routines, and validates vulnerabilities based on data plane effects [5]. This tool was applied to the ONOS SDN controller for vulnerabilities check and found fourteen new vulnerabilities.

As from the EVENTSCOPE architecture **figure 15**, there are three phases:

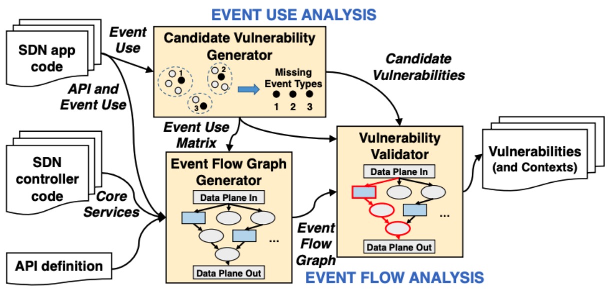


Figure 15: *EVENTSCOPE Architecture*

- **Candidate Vulnerability Generator:** First Phase, the Candidate Vulnerability Generator produces a list of unhandled event types for each SDN application that it takes as input. EVENTSCOPE uses this as a clustering approach that reports event types common in the cluster.

- **Event Flow Graph Generator:** Second Phase, the Event Flow Graph Generator constructs an event flow graph that records how events propagate and influences the system. This Phase takes the application's code, controller's code, and definition of controller API calls as input.

- **Vulnerability Validator:** Third Phase, the Vulnerability Validator takes candidate vulnerabilities in unhandled event types from the first Phase and the event flow graph from the second Phase, and as a result, it generates a list of vulnerabilities.

EVENTSCOPE is a tool that helps practitioners and developers identify cross-plane event-based vulnerabilities. In the simple term, EVENTSCOPE finds potential logic bugs among different applications whose event types are not handled and identify those bugs and vulnerability on the event flow graph.

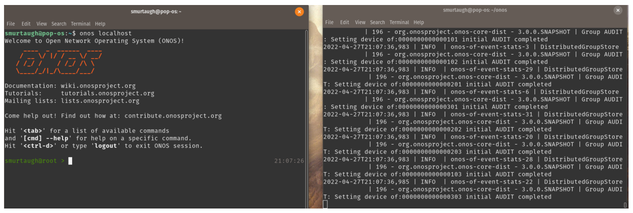
5 Methodology

5.1 Proposed Experiment

The idea for this project was relatively simple. Download and install the ONOS SDN controller. Install the PicoSDN application inside of it. Then use a SDN simulation tool, such as mininet, to simulate one or all of the 4 attacks the authors discuss in the paper. Namely, create networks that demonstrate either EVENTSCOPE CVE-2018-12691, EVENTSCOPE CVE-2019-11189, ProvSDN cross app poisoning attack, or TopoGuard Host Migration attack. We would then use PicoSDN to analyze the simulated network, and compare the results to what the authors got in their paper. As time allowed, we could investigate other existing SDN security and analysis tools, such as EventScope and Forenguard, to see how the results from those tools compare to PicoSDN's in terms of accuracy and utility.

5.2 PicoSDN Issues

Unfortunately, the project did not go as planned. We had a number of issues Getting PicoSDN to work and run over modern installations of ONOS. To begin with, the authors provide scant documentation. Neither the project nor its documentation have been updated to provide details for how PicoSDN should be installed over modern versions of ONOS. ONOS recently switched its build tool from Buck to Bazel, and before it used buck it used a combination of Maven and Karaf. The documentation provided by the authors still assumes ONOS Buck can be used, and even still tries to use Maven and Karaf at some points. The authors also do not provide accurate instructions for configuring the installation environment, telling you to edit files that do not exist in modern installations of ONOS. We searched for a way to download older versions of ONOS to try and make it work, but could find nowhere to do so. We could build ONOS using Maven and Karaf and the instructions made available by the Pica8 Open Networking Project[4], but Ultimately the simplest way we could get ONOS to work was to download and install the latest version and use Bazel, following the instructions provided on the ONOS wiki[2]. No method was ever found for allowing Buck to build and install ONOS, but a picture of onos running using Bazel is below.

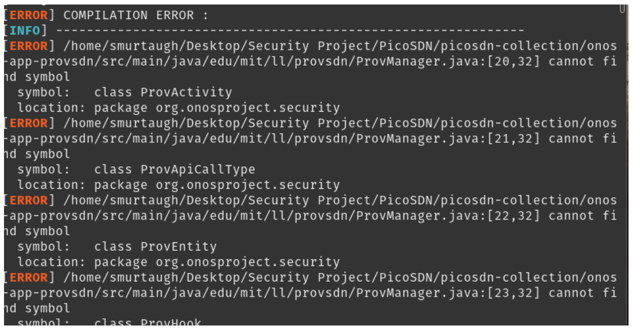

smurtaugh@pop-os: ~
File Edit View Search Terminal Help
smurtaugh@pop-os: ~/onos
File Edit View Search Terminal Help

```
smurtaugh@pop-os:~$ onos localhost
Welcome to Open Network Operating System (ONOS)!
     ____  _  ______  ____
    / __ \/ |/ / __ \/ __/
   / /_/ /    / /_/ /\ \
   \____/_/|_/\____/___/

Documentation: wiki.onosproject.org
Tutorials:     tutorials.onosproject.org
Mailing lists: lists.onosproject.org

Come help out! Find out how at: contribute.onosproject.org

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'logout' to exit ONOS session.

smurtaugh@root >                                              21:07:26
```

```
                | 196 - org.onosproject.onos-core-dist - 3.0.0.SNAPSHOT | Group AUDIT
: Setting device of:0000000000000101 initial AUDIT completed
2022-04-27T21:07:36,983 | INFO  | onos-of-event-stats-3 | DistributedGroupStore
                | 196 - org.onosproject.onos-core-dist - 3.0.0.SNAPSHOT | Group AUDIT
: Setting device of:0000000000000102 initial AUDIT completed
2022-04-27T21:07:36,983 | INFO  | onos-of-event-stats-29 | DistributedGroupStore
                 | 196 - org.onosproject.onos-core-dist - 3.0.0.SNAPSHOT | Group AUDI
T: Setting device of:0000000000000201 initial AUDIT completed
2022-04-27T21:07:36,983 | INFO  | onos-of-event-stats-6 | DistributedGroupStore
                | 196 - org.onosproject.onos-core-dist - 3.0.0.SNAPSHOT | Group AUDIT
: Setting device of:0000000000000301 initial AUDIT completed
2022-04-27T21:07:36,983 | INFO  | onos-of-event-stats-31 | DistributedGroupStore
                 | 196 - org.onosproject.onos-core-dist - 3.0.0.SNAPSHOT | Group AUDI
T: Setting device of:0000000000000202 initial AUDIT completed
2022-04-27T21:07:36,983 | INFO  | onos-of-event-stats-20 | DistributedGroupStore
                 | 196 - org.onosproject.onos-core-dist - 3.0.0.SNAPSHOT | Group AUDI
T: Setting device of:0000000000000203 initial AUDIT completed
2022-04-27T21:07:36,983 | INFO  | onos-of-event-stats-28 | DistributedGroupStore
                 | 196 - org.onosproject.onos-core-dist - 3.0.0.SNAPSHOT | Group AUDI
T: Setting device of:0000000000000103 initial AUDIT completed
2022-04-27T21:07:36,985 | INFO  | onos-of-event-stats-22 | DistributedGroupStore
                 | 196 - org.onosproject.onos-core-dist - 3.0.0.SNAPSHOT | Group AUDI
T: Setting device of:0000000000000303 initial AUDIT completed
```

Figure 16: ONOS Terminal and Running Log

Even once we had overcome the misconfigured environments, and successfully installed ONOS using Bazel, PicoSDN refused to compile and install correctly. During the build process, some 48 errors are returned related to missing functions and variables that the compiler cannot find in the author's code. A screenshot showcasing a few of these errors is below.



```
[ERROR] COMPILATION ERROR :
[INFO] -------------------------------------------------------------
[ERROR] /home/smurtaugh/Desktop/Security Project/PicoSDN/picosdn-collection/onos
-app-provsdn/src/main/java/edu/mit/ll/provsdn/ProvManager.java:[20,32] cannot fi
nd symbol
  symbol:   class ProvActivity
  location: package org.onosproject.security
[ERROR] /home/smurtaugh/Desktop/Security Project/PicoSDN/picosdn-collection/onos
-app-provsdn/src/main/java/edu/mit/ll/provsdn/ProvManager.java:[21,32] cannot fi
nd symbol
  symbol:   class ProvApiCallType
  location: package org.onosproject.security
[ERROR] /home/smurtaugh/Desktop/Security Project/PicoSDN/picosdn-collection/onos
-app-provsdn/src/main/java/edu/mit/ll/provsdn/ProvManager.java:[22,32] cannot fi
nd symbol
  symbol:   class ProvEntity
  location: package org.onosproject.security
[ERROR] /home/smurtaugh/Desktop/Security Project/PicoSDN/picosdn-collection/onos
-app-provsdn/src/main/java/edu/mit/ll/provsdn/ProvManager.java:[23,32] cannot fi
nd symbol
  symbol:   class ProvHook
```

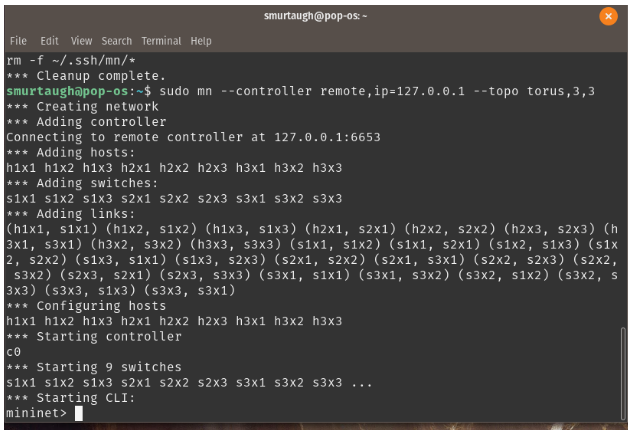
Figure 17: Maven Compilation Errors

Going by the names, it appears that the functions are related to the authors provenance model, and they simply failed to include the functions in their code. However, we did attempt to account for the possibility that it was simply a missing third party package or application that needed to be installed in ONOS that the authors neglected to mention. We searched for any evidence of general "security" packages that could be installed into ONOS, and found only reference to a deprecated "security mode" implemented over felix. We followed the instructions written by ONOS project contributors[1] to install and enable this mode, but it had no effect on the code's ability to compile.

We did make an attempt to reach out to the authors to see if they had any advice. We raised an issue on github, and sent an email to the first author, but they never responded to us. Attempts were made to move files around and reorganize the ONOS and PicoSDN build environments to see if that would make it compile, to no avail. We even looked at trying to add the functions and variables ourselves at one point, but the code was too dense, the concepts too complicated, and the purpose of the missing functions too unexplained for us to accomplish this task successfully. It seems that either the authors neglected to mention a significant step in the installation process in their documentation, or recent updates to ONOS broke this code.

5.3 Mininet

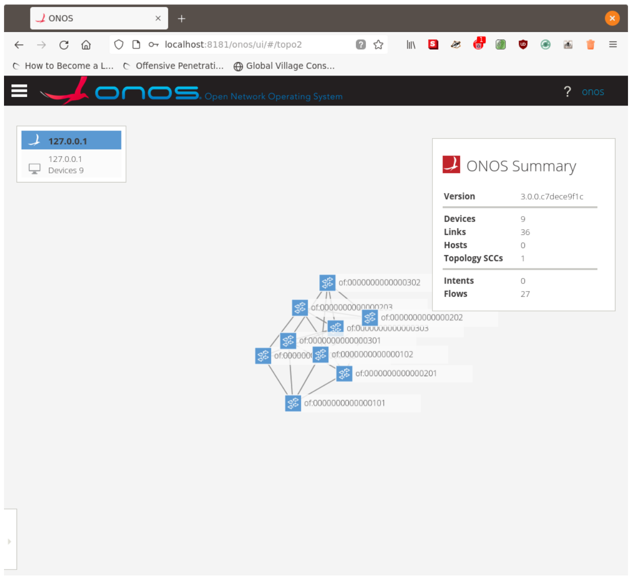
One part of the project that encountered more success was the networking simulation portion of the project. Mininet was downloaded and installed, and used successfully to create a simple networking simulation. A simple 3x3 toroid network was created, containing 9 switches and 9 hosts. The topology of this network is shown below, described by mininet's terminal output:



```
rm -f ~/.ssh/mn/*
*** Cleanup complete.
smurtaugh@pop-os:~$ sudo mn --controller remote,ip=127.0.0.1 --topo torus,3,3
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1x1 h1x2 h1x3 h2x1 h2x2 h2x3 h3x1 h3x2 h3x3
*** Adding switches:
s1x1 s1x2 s1x3 s2x1 s2x2 s2x3 s3x1 s3x2 s3x3
*** Adding links:
(h1x1, s1x1) (h1x2, s1x2) (h1x3, s1x3) (h2x1, s2x1) (h2x2, s2x2) (h2x3, s2x3) (h
3x1, s3x1) (h3x2, s3x2) (h3x3, s3x3) (s1x1, s1x2) (s1x1, s2x1) (s1x2, s1x3) (s1x
2, s2x2) (s1x3, s1x1) (s1x3, s2x3) (s2x1, s2x2) (s2x1, s3x1) (s2x2, s2x3) (s2x2,
 s3x2) (s2x3, s2x1) (s2x3, s3x3) (s3x1, s1x1) (s3x1, s3x2) (s3x2, s1x2) (s3x2, s
3x3) (s3x3, s1x3) (s3x3, s3x1)
*** Configuring hosts
h1x1 h1x2 h1x3 h2x1 h2x2 h2x3 h3x1 h3x2 h3x3
*** Starting controller
c0
*** Starting 9 switches
s1x1 s1x2 s1x3 s2x1 s2x2 s2x3 s3x1 s3x2 s3x3 ...
*** Starting CLI:
mininet>
```

Figure 18: Mininet Terminal

This data from mininet is then sent into ONOS, which interprets it as SDN components. The ONOS gui can then be used to view the network topology graphically, shown below:

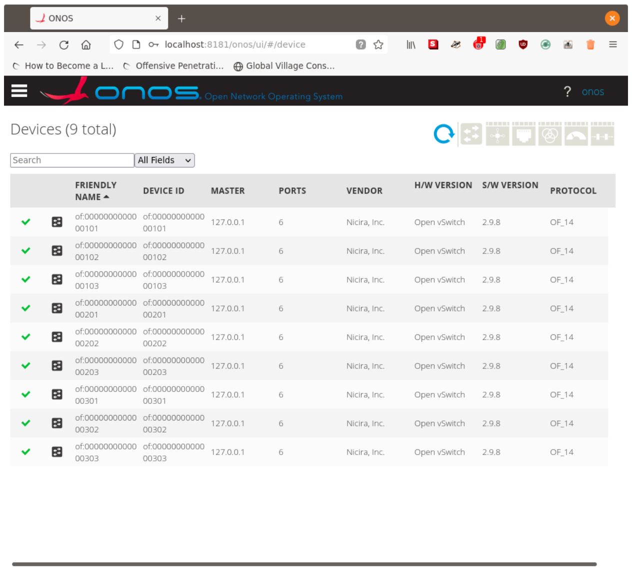


ONOS Summary

Version	3.0.0.c7dece9f1c
Devices	9
Links	36
Hosts	0
Topology SCCs	1
Intents	0
Flows	27

Figure 19: ONOS GUI Displaying Mininet Topology

ONOS also shows other useful networking information such as lists of hosts:

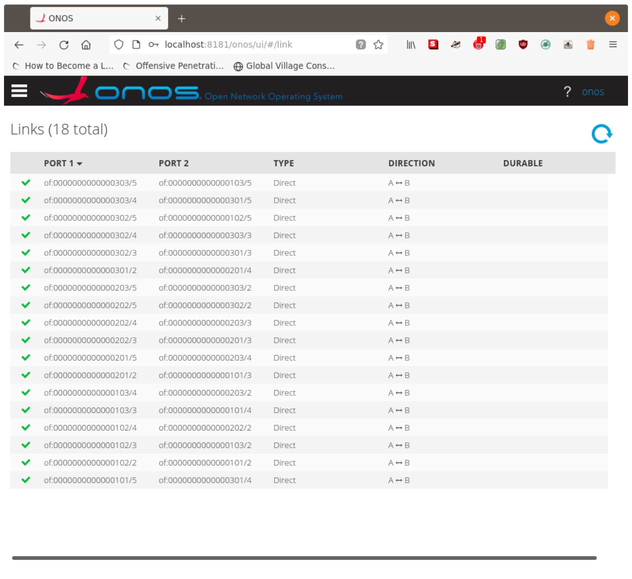


Devices (9 total)

		FRIENDLY NAME ▲	DEVICE ID	MASTER	PORTS	VENDOR	H/W VERSION	S/W VERSION	PROTOCOL
✓		of:0000000000000101	of:0000000000000101	127.0.0.1	6	Nicira, Inc.	Open vSwitch	2.9.8	OF_14
✓		of:0000000000000102	of:0000000000000102	127.0.0.1	6	Nicira, Inc.	Open vSwitch	2.9.8	OF_14
✓		of:0000000000000103	of:0000000000000103	127.0.0.1	6	Nicira, Inc.	Open vSwitch	2.9.8	OF_14
✓		of:0000000000000201	of:0000000000000201	127.0.0.1	6	Nicira, Inc.	Open vSwitch	2.9.8	OF_14
✓		of:0000000000000202	of:0000000000000202	127.0.0.1	6	Nicira, Inc.	Open vSwitch	2.9.8	OF_14
✓		of:0000000000000203	of:0000000000000203	127.0.0.1	6	Nicira, Inc.	Open vSwitch	2.9.8	OF_14
✓		of:0000000000000301	of:0000000000000301	127.0.0.1	6	Nicira, Inc.	Open vSwitch	2.9.8	OF_14
✓		of:0000000000000302	of:0000000000000302	127.0.0.1	6	Nicira, Inc.	Open vSwitch	2.9.8	OF_14
✓		of:0000000000000303	of:0000000000000303	127.0.0.1	6	Nicira, Inc.	Open vSwitch	2.9.8	OF_14

Figure 20: ONOS Device List

And links:

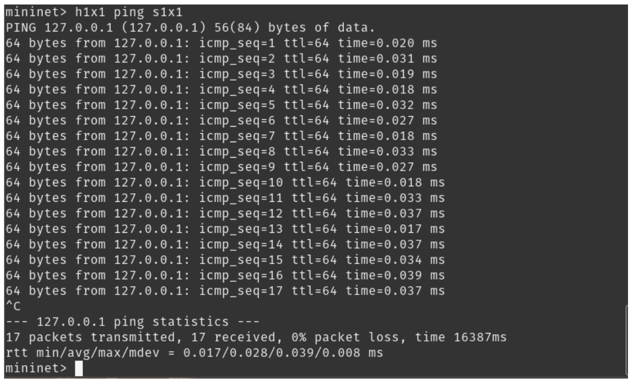


Links (18 total)

	PORT 1 ▼	PORT 2	TYPE	DIRECTION	DURABLE
✓	of:0000000000000303/5	of:0000000000000103/5	Direct	A ↔ B	
✓	of:0000000000000303/4	of:0000000000000301/5	Direct	A ↔ B	
✓	of:0000000000000302/5	of:0000000000000102/5	Direct	A ↔ B	
✓	of:0000000000000302/4	of:0000000000000303/3	Direct	A ↔ B	
✓	of:0000000000000302/3	of:0000000000000301/3	Direct	A ↔ B	
✓	of:0000000000000301/2	of:0000000000000201/4	Direct	A ↔ B	
✓	of:0000000000000203/5	of:0000000000000303/2	Direct	A ↔ B	
✓	of:0000000000000202/5	of:0000000000000302/2	Direct	A ↔ B	
✓	of:0000000000000202/4	of:0000000000000203/3	Direct	A ↔ B	
✓	of:0000000000000202/3	of:0000000000000201/3	Direct	A ↔ B	
✓	of:0000000000000201/5	of:0000000000000203/4	Direct	A ↔ B	
✓	of:0000000000000201/2	of:0000000000000101/3	Direct	A ↔ B	
✓	of:0000000000000103/4	of:0000000000000203/2	Direct	A ↔ B	
✓	of:0000000000000103/3	of:0000000000000101/4	Direct	A ↔ B	
✓	of:0000000000000102/4	of:0000000000000202/2	Direct	A ↔ B	
✓	of:0000000000000102/3	of:0000000000000103/2	Direct	A ↔ B	
✓	of:0000000000000102/2	of:0000000000000101/2	Direct	A ↔ B	
✓	of:0000000000000101/5	of:0000000000000301/4	Direct	A ↔ B	

Figure 21: ONOS List of Mininet Links

The mininet terminal can be used to run applications and commands, the results of which will be reflected in ONOS and the gui. For example, you can run ping between host 1 and switch 1:



```
mininet> h1x1 ping s1x1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.020 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.031 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.019 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.018 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.032 ms
64 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=0.027 ms
64 bytes from 127.0.0.1: icmp_seq=7 ttl=64 time=0.018 ms
64 bytes from 127.0.0.1: icmp_seq=8 ttl=64 time=0.033 ms
64 bytes from 127.0.0.1: icmp_seq=9 ttl=64 time=0.027 ms
64 bytes from 127.0.0.1: icmp_seq=10 ttl=64 time=0.018 ms
64 bytes from 127.0.0.1: icmp_seq=11 ttl=64 time=0.033 ms
64 bytes from 127.0.0.1: icmp_seq=12 ttl=64 time=0.037 ms
64 bytes from 127.0.0.1: icmp_seq=13 ttl=64 time=0.017 ms
64 bytes from 127.0.0.1: icmp_seq=14 ttl=64 time=0.037 ms
64 bytes from 127.0.0.1: icmp_seq=15 ttl=64 time=0.034 ms
64 bytes from 127.0.0.1: icmp_seq=16 ttl=64 time=0.039 ms
64 bytes from 127.0.0.1: icmp_seq=17 ttl=64 time=0.037 ms
^C
--- 127.0.0.1 ping statistics ---
17 packets transmitted, 17 received, 0% packet loss, time 16387ms
rtt min/avg/max/mdev = 0.017/0.028/0.039/0.008 ms
mininet> 
```

Figure 22: Mininet Running Ping

And watch the packets travel across the network shown above. They will also be recorded as data in ONOS that can be viewed later. This simple network topology would have been a great way to get an initial function test of PicoSDN, as well as being a good platform for testing the ProvSDN cross-app poisoning attacks and the TopoGuard Host Migration attacks, since it had plenty of hosts to run vulnerable applications and plenty of hosts that the migration attack could mimic.

6 Other similar analysis tools:

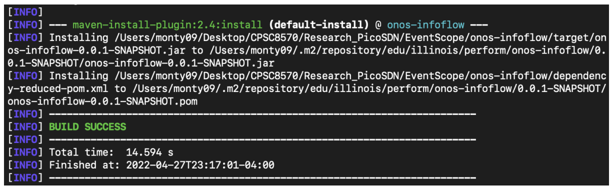
Since We had many issues with the installation of PicoSDN, we decided to duplicate the result of other analysis tools like EVENTSCOPE and FORENGUARD.

- FORENGUARD is very similar to PicoSDN because it also does causal analysis to find the root cause of the attacks as described in the Related work subsection 4.1. Since the source code for ForenGuard was not available for public use, we could not be able to simulate the same result.

- EVENTSCOPE identifies cross-plane vulnerabilities like possible bugs among applications as described in the Related Work subsection 4.2. While working dublicating the result of EVENTSCOPE, we ran into some errors that was unable to fix due to the time complexity. The errors are described on the subsection 6.1.

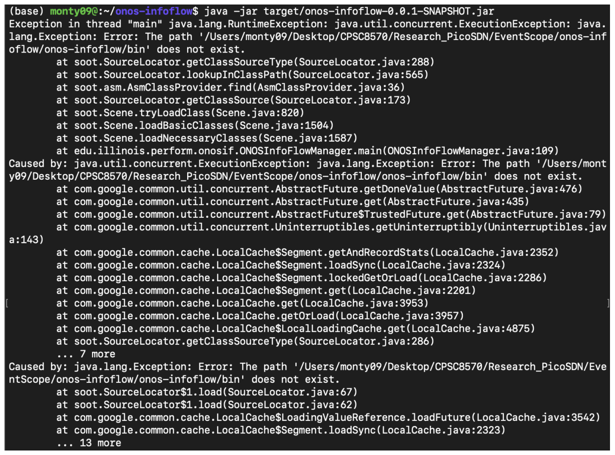
6.1 EVENTSCOPE Issues:

While working on the EVENTSCOPE, we found several errors that we ran into. On the source code of the EVENTSCOPE documentation of installation, there were only three lines of code build, run, and Performance benchmarks. We successfully built the EVENTSCOPE using maven see **figure 23**. While running the EVENTSCOPE using *java -jar*, existing path issue with other dependencies problems see **figure 24**. Without running the EVENTSCOPE, we cannot perform performance benchmarks see **figure 25**.



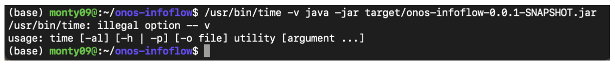
```
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ onos-infoflow ---
[INFO] Installing /Users/monty09/Desktop/CPSC8570/Research_PicoSDN/EventScope/onos-infoflow/target/on
os-infoflow-0.0.1-SNAPSHOT.jar to /Users/monty09/.m2/repository/edu/illinois/perform/onos-infoflow/0.
0.1-SNAPSHOT/onos-infoflow-0.0.1-SNAPSHOT.jar
[INFO] Installing /Users/monty09/Desktop/CPSC8570/Research_PicoSDN/EventScope/onos-infoflow/dependenc
y-reduced-pom.xml to /Users/monty09/.m2/repository/edu/illinois/perform/onos-infoflow/0.0.1-SNAPSHOT/
onos-infoflow-0.0.1-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  14.594 s
[INFO] Finished at: 2022-04-27T23:17:01-04:00
[INFO] ------------------------------------------------------------------------
```

Figure 23: *EVENTSCOPE Build*



```
(base) monty09@:~/onos-infoflow$ java -jar target/onos-infoflow-0.0.1-SNAPSHOT.jar
Exception in thread "main" java.lang.RuntimeException: java.util.concurrent.ExecutionException: java.
lang.Exception: Error: The path '/Users/monty09/Desktop/CPSC8570/Research_PicoSDN/EventScope/onos-inf
oflow/onos-infoflow/bin' does not exist.
        at soot.SourceLocator.getClassSourceType(SourceLocator.java:288)
        at soot.SourceLocator.lookupInClassPath(SourceLocator.java:565)
        at soot.asm.AsmClassProvider.find(AsmClassProvider.java:36)
        at soot.SourceLocator.getClassSource(SourceLocator.java:173)
        at soot.Scene.tryLoadClass(Scene.java:820)
        at soot.Scene.loadBasicClasses(Scene.java:1504)
        at soot.Scene.loadNecessaryClasses(Scene.java:1587)
        at edu.illinois.perform.onosif.ONOSInfoFlowManager.main(ONOSInfoFlowManager.java:109)
Caused by: java.util.concurrent.ExecutionException: java.lang.Exception: Error: The path '/Users/mont
y09/Desktop/CPSC8570/Research_PicoSDN/EventScope/onos-infoflow/onos-infoflow/bin' does not exist.
        at com.google.common.util.concurrent.AbstractFuture.getDoneValue(AbstractFuture.java:476)
        at com.google.common.util.concurrent.AbstractFuture.get(AbstractFuture.java:435)
        at com.google.common.util.concurrent.AbstractFuture$TrustedFuture.get(AbstractFuture.java:79)
        at com.google.common.util.concurrent.Uninterruptibles.getUninterruptibly(Uninterruptibles.jav
a:143)
        at com.google.common.cache.LocalCache$Segment.getAndRecordStats(LocalCache.java:2352)
        at com.google.common.cache.LocalCache$Segment.loadSync(LocalCache.java:2324)
        at com.google.common.cache.LocalCache$Segment.lockedGetOrLoad(LocalCache.java:2286)
        at com.google.common.cache.LocalCache$Segment.get(LocalCache.java:2201)
        at com.google.common.cache.LocalCache.get(LocalCache.java:3953)
        at com.google.common.cache.LocalCache.getOrLoad(LocalCache.java:3957)
        at com.google.common.cache.LocalCache$LocalLoadingCache.get(LocalCache.java:4875)
        at soot.SourceLocator.getClassSourceType(SourceLocator.java:286)
        ... 7 more
Caused by: java.lang.Exception: Error: The path '/Users/monty09/Desktop/CPSC8570/Research_PicoSDN/Eve
ntScope/onos-infoflow/onos-infoflow/bin' does not exist.
        at soot.SourceLocator$1.load(SourceLocator.java:67)
        at soot.SourceLocator$1.load(SourceLocator.java:62)
        at com.google.common.cache.LocalCache$LoadingValueReference.loadFuture(LocalCache.java:3542)
        at com.google.common.cache.LocalCache$Segment.loadSync(LocalCache.java:2323)
        ... 13 more
```

Figure 24: *EVENTSCOPE Run error*



```
(base) monty09@:~/onos-infoflow$ /usr/bin/time -v java -jar target/onos-infoflow-0.0.1-SNAPSHOT.jar
/usr/bin/time: illegal option -- v
usage: time [-al] [-h | -p] [-o file] utility [argument ...]
(base) monty09@:~/onos-infoflow$ 
```

Figure 25: *EVENTSCOPE Performance benchmarks error*

7 Evaluation

Unfortunately, we never got PicoSDN working, so we never actually got to run it on our network and obtain any experimental results. Neither PicoSDN, nor any of the other tools we looked at, were up to date with modern versions of ONOS. Without a code update from the authors, or a significant time investment in updating the code, The results of this paper cannot be reproduced or analyzed. In an ideal world, we would have been able to run some routine network work loads on the mininet simulation, as well as all 4 attacks described in the paper. Data could then be analyzed and collected with PicoSDN as well as 2 or 3 of the other security tools they mentioned in the paper, and the results compared. This would allow us to determine just what improvements, if any were made by PicoSDN, as well as compare performance impacts and assess the actual utility provided by the tool.

8 Future Work

Due to the failure of our attempts to get the code up to date and running, opportunities for future work still include virtually the entirety of our proposed experiments, following work performing a full update of PicoSDN for modern versions of ONOS. An investigation into what has changed in ONOS recently, and where the project is attempting to take its SDN controller should also be performed. This would allow us to see how the tool can best be augmented with those changes in mind, or if the controller is on a path where the tool cannot work, or would not provide increased utility in the face of new security tools or features that would be or have been released alongside these updates.

9 Contributions

SAM's CONTRIBUTION: Downloaded and installed ONOS, helped others get it running. Downloaded PicoSDN, performed troubleshooting relating to fixing the installation environment configuration, enabling security mode, searching for older versions of ONOS, and investigating the feasibility of updating the code ourselves. Downloaded and installed mininet, created the networking simulation, ran basic networking tests, and integrated the mininet simulation with ONOS.

Ankita's contribution: DID Research about ONOS and came up with few issues that ONOS had and why it was not working in its documented form. Also downloaded PicoSDN ,tried to perform some troubleshooting, to make PicoSDN run . Tried to run PicoSDN by building with various tools. Also did extensive research on SDN In order to understand how to make the PICOSdn work.

Mayank's Contribution: I have successfully downloaded and installed new ONOS and helped other members with the process. I have tried installing PicoSDN with the new ONOS but since PicoSDN uses buck, I was unable to install/run successfully. I have reached out to PicoSDN members through GitHub in March but I did not get any help from them. I have also tried installing the ONOS version given by the PicoSDN community, but it failed because of lack of instruction. I have tried several other ways to install ONOS with buck. I have also tried other analysis tools like EVENTSCOPE and FORENGUARD.

References

[1] JOSHI, P., AND VACHUSKA, T. Security-mode onos, Jan 2020.

[2] LANTZ, B. Developer quick start, Apr 2020.

[3] ROSENCRANCE, L., ENGLISH, J., AND BURKE, J. What is software-defined networking (sdn)? definition from searchnetworking, Sep 2021.

[4] TEAM, D. How to install onos, Nov 2018.

[5] UJCICH, B. E., JERO, S., SKOWYRA, R., GOMEZ, S. R., BATES, A., SANDERS, W. H., AND OKHRAVI, H. Automated discovery of cross-plane event-based vulnerabilities in software-defined networking. *Network and Distributed System Security Symposium* (2020).

[6] WANG, H., YANG, G., CHINPRUTTHIWONG, P., XU, L., ZHANG, Y., AND GU, G. Towards fine-grained network security forensics and diagnosis in the sdn era. pp. 3–16.

Notes