

Santa Clara University

Scholar Commons

Computer Science and Engineering Senior
Theses

Engineering Senior Theses

6-15-2020

SDHome: Securing Fast Home Networks

Christopher Batula

Holden Gordon

Tianyi Zhao

Follow this and additional works at: https://scholarcommons.scu.edu/cseng_senior



Part of the [Computer Engineering Commons](#)

SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Date: June 15, 2020

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

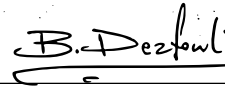
Christopher Batula, Holden Gordon, Tianyi Zhao

ENTITLED

SDHome: Securing Fast Home Networks

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING



Thesis Advisor

DocuSigned by:



96CE94A1A83A48A...

Department Chair

SDHome: Securing Fast Home Networks

by

Christopher Batula, Holden Gordon, Tianyi Zhao

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 15, 2020

SDHome: Securing Fast Home Networks

Christopher Batula, Holden Gordon, Tianyi Zhao

Department of Computer Science and Engineering
Santa Clara University
June 15, 2020

ABSTRACT

Distributed denial of service (DDoS) is a highly discussed network attack in Software Defined Networks. Attacks such as the Mirai Botnet threaten to compromise portion of large networks, including home users. Today, corporations secure their network using enterprise level software to protect their network from DDoS attacks. But there solutions are meant for large networks and depend on expensive hardware. There are few security solutions for home users and most are expensive or require a subscription for full protection. In this paper, we propose a new solution in the form of a plug and play device that will allow home users to easily take control of their network. We will be using the SDN controller Faucet and the protocol OpenFlow 1.3 to enable software defined functionalities. In addition to more basic network features such as blocking websites, the device will allow users to receive notifications about possible malicious activities on their network, generate device profiles for all devices on the network, and automatically detect and mitigate flooding attacks using a random forest classifier. We implement our network virtually using Graphic Network Simulator 3.

Table of Contents

1	Introduction	1
1.1	Motivation	1
1.2	Problem	1
1.3	Solution	2
2	Requirements	3
2.1	Functional Requirements	3
2.1.1	Critical	3
2.1.2	Recommended	3
2.1.3	Suggested	4
2.2	Non-functional Requirements	4
2.3	Design Constraints	4
3	Use Cases	5
3.1	Use Case Diagram	5
3.2	Actions	6
3.2.1	Manage Blocked Sites	6
3.2.2	View Malicious Activity	6
4	Activity Diagram	7
5	Technologies Used	9
5.1	Programming Languages and Frameworks	9
6	Design Rationale	12
6.1	Technology	12
6.2	User Interface	12
7	Architectural Diagram	13
7.1	Virtual Home Network	13
7.2	Faucet and OpenFlow	15
8	System Description	16
9	Testing	17
9.1	UI Testing	17
9.2	Machine Learning Testing	17
9.3	Network Testing	18
9.4	System Testing	19
10	Difficulties Encountered	20
10.1	Group Meetings	20
10.2	COVID-19	20
10.3	Centralized Virtualization	21

11 Ethics	22
11.1 User Information	22
12 Conclusion	23
12.1 Suggested Changes	23
12.2 Lessons Learned	24
A Installation Guide	25
A.1 Installing the system	25
B User Manual	27
B.1 Manage Websites	27
B.2 View Malicious Activity	27

List of Figures

3.1	Use Case Diagram	5
4.1	Activity Diagram	7
7.1	Virtual Network Simulated in GNS3	13
7.2	VLAN Network	14
7.3	Faucet/Gauge Interactions	15

Chapter 1

Introduction

1.1 Motivation

The IoT Marketplace has seen exponential increases in growth the past years [Doffman et al., 2019]. The industry has grown from 8.7 billion to 42.1 billion IoT devices connected from 2012 to 2019. Revenue from IoT Devices has also followed an exponential growth and is forecasted to reach 1.5 trillion dollars in 2025[19]. Attacks are also on the rise. The company FSecure found a 300% increase in attacks on their honeypots in 2018 alone [Siv]. Attacks went from 813 Million to 2.9 billion in a during this year and 99.9% were also automated. Within these attacks there have been several crippling DDoS attacks composed of compromised IoT devices that have been detrimental to modern infrastructure.

DDoS, distributed denial of service, is a prevalent network security attack in Software Defined Networks. The attack can be described as a congregation of compromised devices sending mass amounts of data to a single system. This causes the network to be overwhelmed with incoming packets and will eventually crash. With the rise of the Internet of Things, attacks such as the Mirai botnet threaten to compromise portions of large networks. The Mirai botnet was estimated to be comprised of 20,000 infected devices with a max bandwidth of about 901 GBPS. This poses a large problem for not only large companies, but the everyday user as well. According to Kaspersky Lab, a single DDoS attack costs enterprises and smaller businesses an average of 2 million dollars and 120,000 dollars, respectively. Depending on the severity of the attack, the amount of time that networks stay offline from a DDoS can vary from 24 hours to a week long. Even though there is no direct cost to an everyday user, they won't have internet access until the attack is addressed.

1.2 Problem

Today, enterprises secure their network using software such as Verizon's DDoS Shield but there isn't a clear security solution for home internet users. Some tech companies offer routers for home users with security solutions such as Norton Core, but their products are expensive and do not allow the user to collect network analytics. Most off the

shelf routers have a pre-installed firewall. Unfortunately, a firewall will not protect a network from a DDoS attack as malicious users will use a firewall as an entry point. Additionally, admin consoles on most routers are extremely basic and most users don't have the experience to use it. Some solutions pose a universal policy to mitigate attacks on a home network and treats all devices equally. They implement these policies using packet level inspection. However, there are numerous problems to this approach. These solutions naively assume that all devices have the same behavior. For example, a smart microwave may have less internet activity than a ring camera. If we treat these devices equally, we assume that these devices will fall within the same behavior baselines. In addition, packet inspection yields a high overhead which is counterproductive for a security system.

1.3 Solution

Home users should be able to remotely monitor and manage their network in real time. Currently, there is no way for a user to manage their network without replacing their router with a security-integrated router. A simpler solution for the everyday user would consist of a plug-and-play device that allows remote network management through a mobile app. This should be accomplished through a software defined network architecture due to the increased switching speed and real-time forwarding programmability. This can give users greater control over their network while providing APIs to obfuscate lower level details. We access information flow features rather than packet by packet features to reduce overhead of our detection system. We also provide better data analytics to users for their home networks. Our detection algorithms are real time, highly accurate, and utilize purely stateless flow information that is novel and implementable in a real home network, separating our work from other state of the art implementations. The processing for our machine learning is executed at the edge for both training and inference. We partition the network into verified and unverified VLANs to help mitigate false positives. This would allow users to easily install the device and receive notifications about possible malicious activity. The device would also allow the user to switch between configurations as they see fit.

Chapter 2

Requirements

These are the requirements for our system.

2.1 Functional Requirements

2.1.1 Critical

- DDoS Mitigation
- Machine learning based DDoS detection

We prioritize the functionality of the system than the UI. The main goal of this system is to create a device that mitigates DDoS attacks that target your network. We use a random forest classifier to classify device types and to detect flooding attacks on the network. This machine learning approach makes this version

2.1.2 Recommended

- Link device names to IP addresses
- Black box
- Display Network Notifications
- Automatic connection between app and "black box"

Our functional requirements are as follows. We want to link devices names to their IP addresses to enable the user to determine which device moves from VLANs. We also want our system to connect as a "black box". In other words, we want our system to be easily connected to the user's home network. This would abstract all the complicated details away from the user which appeals to a much wider range of users.

2.1.3 Suggested

- Monthly analytical report

If we have time, we plan to add a monthly analytical report. This report would consist of all the detected DDoS attacks from the last month and which device they originated from. The report would help the user keep a record of these attack if they so please.

2.2 Non-functional Requirements

- System must work efficiently enough to quickly mitigate DDoS attacks
- System should be easy and intuitive to use

Our system will perpetuate simplicity from the users perspective. Most of the work will be done in the background. This will allow the system to be more reactive than detection via user interaction. This will make the system very intuitive to use as it deals with

2.3 Design Constraints

- Must be deployable in home networks
- Must be affordable for an everyday internet user

Our target audience are home users who don't have programming experience but want additional home security. We initially aimed to make this device deployable on raspberry pi's for affordability. However, since we virtualized our project, we were not able to test the performance on a raspberry pi.

Chapter 3

Use Cases

3.1 Use Case Diagram

All actors are home internet users who are represented in the diagram by Home User. We want our users to be able to use SDHome without any previous internet or computer knowledge. Therefore we want to hide low-level, abstract information from our users. The system mostly works independently of the user. All the classification and VLAN activity is completely hidden from the user. However, they receive a notification when a device is moved between VLANs and when a flooding attack is detected. This ensures that the user only sees important information and doesn't overwhelm them. The user can view a report of all these activities through the app's main menu. In addition, the user can manage blocked websites.

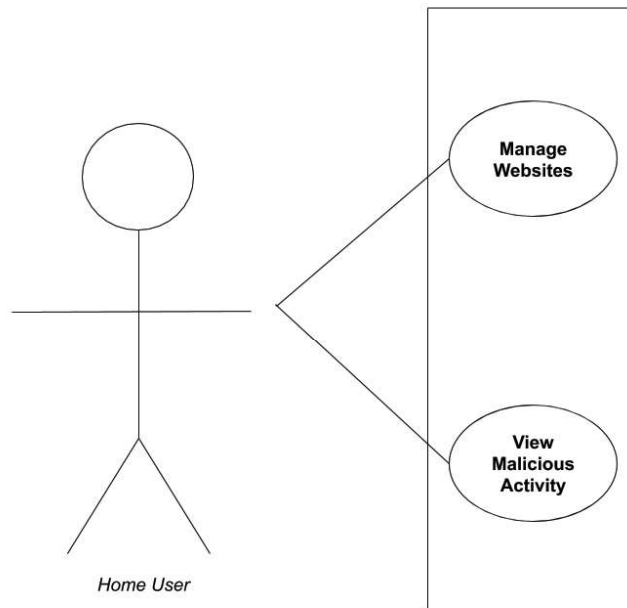


Figure 3.1: Use Case Diagram

3.2 Actions

3.2.1 Manage Blocked Sites

- Description: Block/unblock websites
- Actors: Home User
- Pre-Condition: (1) SDHome must be connected to your router
- Post-Condition: (1) Website will be blocked/unblocked for network users
- Exception: Website is already blocked/not blocked

The first page in our UI is managing websites. On this page, the user can block or unblock websites. However, you cannot block a website that is already blocked. Similarly, you cannot unblock a website that is not blocked.

3.2.2 View Malicious Activity

- Description: View malicious activity report
- Actors: Home User
- Pre-Condition: (1) Malicious activity occurred on the network
- Post-Condition: (1) App displays malicious activity report
- Exception: None

The second page in our UI is viewing the malicious activity report. This page displays devices recently moved between VLANs. In addition, the user can see recently detected flooding attacks from which ip.

Chapter 4

Activity Diagram

This diagram shows how the user will navigate our app and use its functionalities. Our system is designed to work with little or no user interaction. In this way, the system can react to attacks much quicker than a user can. This will also diversity our target audience by making our device intuitive for users of all skill levels. However, the user will be notified of activities on their network.

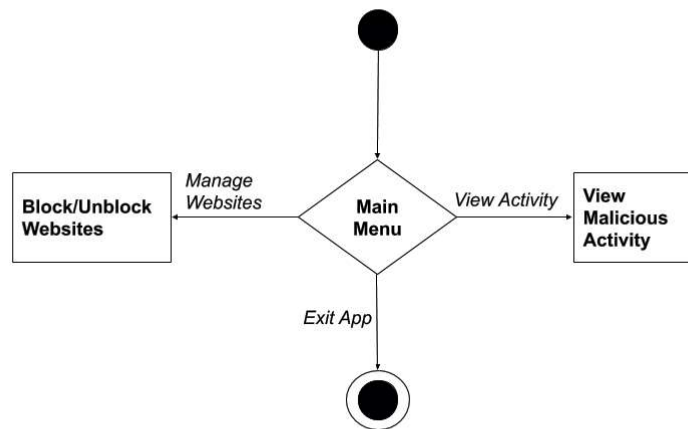


Figure 4.1: Activity Diagram

This diagram shows two activities that the actor can do; view activity or manage websites. The malicious activity report will show a running list of:

- Devices moved between VLANs

- Detected flooding attacks

The information in this section will be the activity in the last month. Secondly, we have another page to block or unblock websites on your network. This is accomplished through remote flow rule installation.

Chapter 5

Technologies Used

These are the technologies we used for our system. All of the components are open source to enhance innovation in the industry. We use Open vSwitch and OpenFlow 1.3 many SDN devices support these open source software. Graphic Network Simulator 3, GNS3, is used to connect the system together. It provides an easy way to connect devices and supports a wide range of devices. GNS3 is integrated with VMWare Fusion as well. This made connecting devices easier than using Virtual Box. We use the machine learning libraries Numpy, Pandas, Sklearn, and matplotlib to classify device types. We write all of our code using the programming languages Python 3, Swift, and C#.

5.1 Programming Languages and Frameworks

- Open vSwitch

Open vSwitch is an open source virtual multilayer switch that allows us to configure forwarding functions while maintaining its switching capabilities. OVS uses the protocol OpenFlow 1.3 to receive control messages from the controller. This switch is easily integrated with virtual machines, making it an excellent open source switch for local virtualization using GNS3.

- OpenFlow 1.3

OpenFlow is a protocol that allows for custom configuration of the forwarding plane of a switch. This allows a SDN controller to manage switches and configure the flow of packets. We chose this specific version of OpenFlow because it is widely used and supported in the IoT industry. Additionally, it supports IPv6, tunneling, and quality of service.

- Faucet SDN Controller

The Faucet Controller is a software defined networks controller based on RYU and Valve. This explains Faucet's high scalability. This versatile controller supports configuration of access control lists, VLANs, stacking of

switches, and many more features. The controller also has components Gauge, Grafana, and Prometheus. Gauge collects and sends flow and port statistics to Prometheus and Grafana for local visualization.

- Graphic Network Simulator 3

Graphic network simulator 3, allows you to virtualize a network on your host computer. The software supports a vast number of devices such as numerous Cisco routers, Juniper routers, and other hosts through VMWare, VirtualBox, Qemu, and Docker. GNS3 allows for easy connection between devices and is even integrated with Wireshark and VMWare Fusion.

- VMWare Fusion

VMWare Fusion is a hypervisor that is made for macOS. It allows for emulation of different operating systems such as Windows, Solaris, and Linux. This is powerful as you can run multiple operating systems concurrently on the same host system. However, doing so will decrease performance.

- Numpy

Numpy is a python library which offers multidimensional arrays and utilities to manipulate and restructure arrays. This library has numerous computational tools including linear algebra routines and mathematical functions. We use this library to preprocess collected metrics before using our machine learning algorithm.

- Pandas

Pandas is a Python library that is designed for data manipulation and analysis. It is extremely powerful due to its ability to manipulate and restructure data. We use this library to manipulate our dataset.

- Sklearn

SKlearn is a free but powerful machine learning library for python. It supports many different machine learning models such as KNN, naive bayes, and random forest. It is widely used by many machine learning programmers. We use SKlearn to run Random Forest to classify devices on the user's network.

- Matplotlib

Matplotlib is a powerful plotting library for python and is widely used by many python programmers. It provides an objected-oriented API for embedding plots into applications. We use matplotlib to graph our data for machine learning during development. We do not use matplotlib besides for machine learning testing.

- Python 3

Python is a powerful, interpreted, high level programming language. Python supports a wide variety of machine learning and deep learning libraries. This being said, Python proves to be a very good choice to implement

our learning algorithms. In addition, the Faucet controller is based on the RYU controller which is written in Python. This made it very easy for us to implement our learning algorithms inside of our SDN controller.

- Swift

Swift is a programming language that is designed for IOS mobile application development. We use swift to program our user interface because swift offers fast performance and we want to achieve a fast data transfer between our mobile application and our controller.

Chapter 6

Design Rationale

6.1 Technology

We chose to use Open vSwitch and OpenFlow 1.3 due to their widespread use across the SDN industry. The Faucet SDN controller posed to be a perfect fit for our system due to its dynamic configuration and metric collection capabilities. Faucet also supports custom VLAN creation and allows you to specify rules on those VLANs. Faucet is based on the RYU controller, which is based in Python. The Gauge section of the Faucet controller periodically sends metrics to Prometheus, a monitoring and database system. However, this provides a streamlined way of collecting metrics for other functions. We use the python libraries sklearn, pandas, and numpy to implement our random forest algorithm for device classification. Using these libraries simplified our machine learning implementation. The random forest algorithm runs with very little overhead which is essential for a software defined network controller. GNS3 provides an easy way of virtualizing our network. The software connects interfaces together and provides support for a wide range of devices via docker and vmware. The mobile application was implemented using Swift to create a user friendly user-interface. We decided to use C# as the main programming language here because it's a type-safe programming language and has rich features.

6.2 User Interface

In order to allow users with no previous computer or internet background to use our IOS application, we decided to hide low-level information from our users. We also want our user interface be user friendly. We also ensured that we have fast and secure data transfer between the mobile application and the controller. Using our mobile application users will be able to use specific features such as view malicious activities on their network via notifications and modify blocked websites. In addition, the UI allows the user to feel more engaged by sending them notifications about their network.

Chapter 7

Architectural Diagram

In this section, we describe the different architectural aspects of our software.

7.1 Virtual Home Network

The virtualized network consists of a two hosts, a switch, and a controller. The hosts are running Ubuntu 19.10 on VMWare Fusion. The switch runs Open vSwitch and uses the OpenFlow protocol to communicate with the controller on datapath 0x1. We use the Faucet controller as our software defined network controller. We chose this architecture due to the high overhead of running VM's on a local machine.

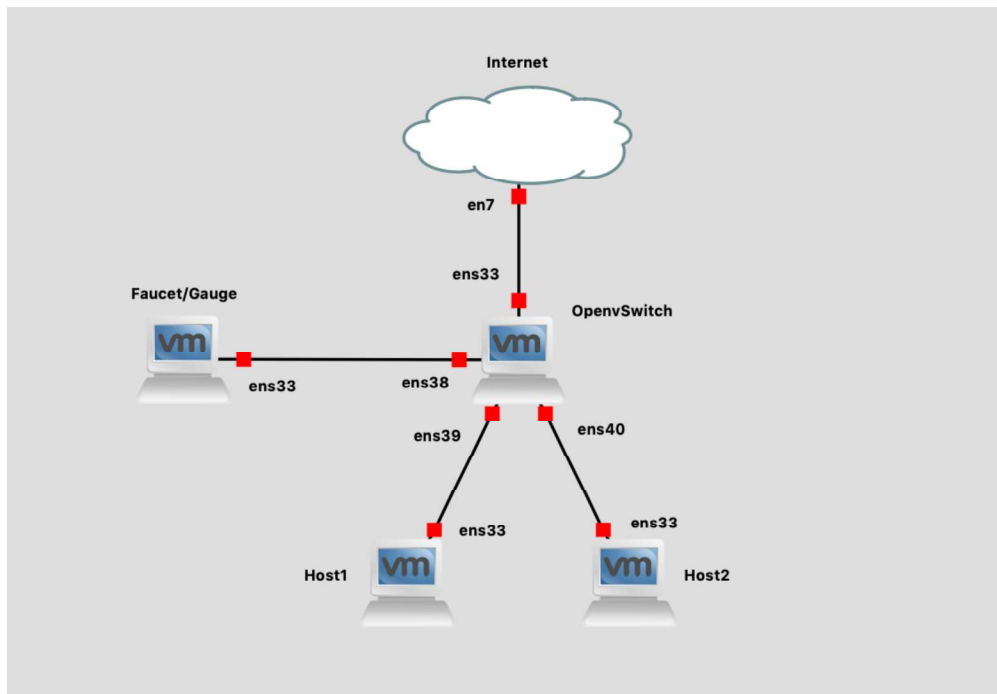


Figure 7.1: Virtual Network Simulated in GNS3

This architecture is simulating a home network with two devices. These two hosts initially start in the unverified

VLAN and are migrated based on our random forest classifier. Each VM runs with 1024 MB of ram, except for the Faucet VM that has 1536 MB to account for the overhead of device classification.

This is our virtual network. We have two VLANs over the home network. For our experiments, we used a two hosts that simulated IoT devices residing in both networks and we observed the behavior depending on the traffic they are sending. We use captured IoT data from UNSW Sydney to simulate real IoT traffic in the network. This allows for a repeatable way to test the system. We assign an IP to each device in the network 192.168.128.0/24. We connected our switch to a router via an ethernet connection. The connections from host to switch are created using VMWare's network adapters. They are configured inside of GNS3 and are connected using vmnets.

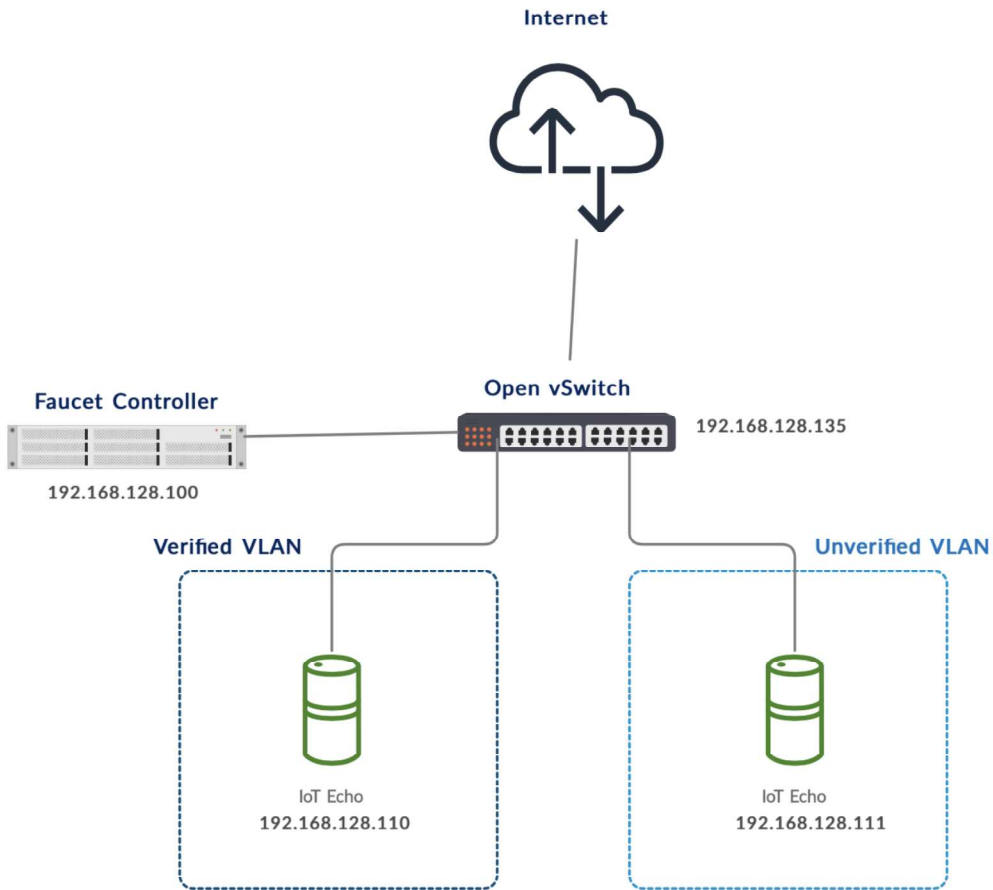


Figure 7.2: VLAN Network

7.2 Faucet and OpenFlow

The first half of our SDN controller is Faucet. Faucet is actually based on the SDN controller RYU, allowing us to access some of the controller's core features. The Faucet controller is responsible for managing flow, switch states, and VLANs. Faucet reads its configuration from the faucet.yaml file. The power behind faucet resides in its ability to partition VLANs, configure datapaths, and implement access control list rules over those VLANs. The VLANs, datapaths, and ACLs can be configured inside the faucet.yaml file. Lets take a look at the second half of our SDN controller. The Gauge controller is responsible for collecting flow and port statistics from the switch. Gauge polls this information every 2 minutes by default. The Faucet/Gauge compound controller also has three more components called Prometheus, Influx database, and Grafana that can be used to graph collected statistics. However, we collected metrics using our own scripts so we never utilized these components. Gauge reads its configuration from the gauge.yaml file. In this file, you can configure watchers that poll on specific datapaths. For our system, we used the default gauge configuration. This compound controller supports OpenFlow switches as well as Open vSwitch.

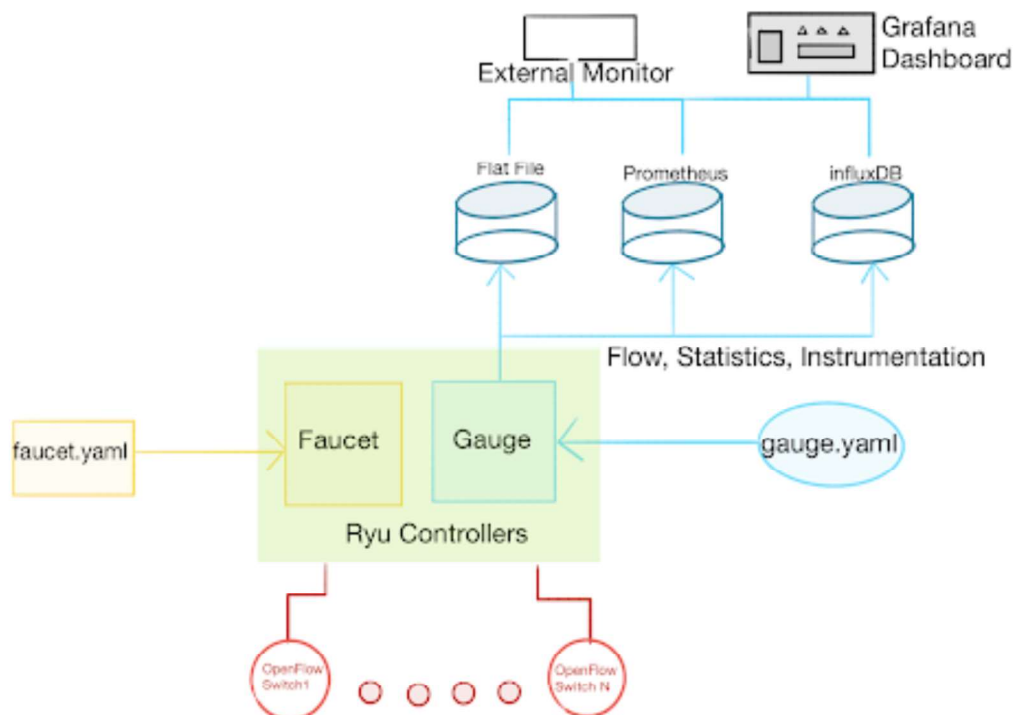


Figure 7.3: Faucet/Gauge Interactions

Chapter 8

System Description

Our system protects the user's home network by partitioning their home network and creating per-device profiles. In order to mitigate DDoS attacks, we use a random forest machine learning algorithm to classify device types and detect when a flood attack is occurring. We segregate verified and unverified devices in two separate VLANs on the network. Our verified VLAN has full connectivity while the unverified VLAN has very strict quality of service. More specifically, number of packets sent every 120 seconds and packet size are regulated. A device's profile is based on their usage of a particular protocol. We use a running average for each protocol percentage per unique ip address. If a protocol is used more than .1%, we install a tolerance of 10% deviation over that protocol. If any of the device's metrics deviate by 10% or more, the device is moved to the unverified VLAN. There, the device's flow metrics are run through another random forest algorithm to detect whether or not a flooding attack is occurring. If a flood attack is detected, the device is kicked off the network. After 15 minutes, the device will be able to reconnect to the network. Once the device passes the flood attack 10 times (about 20 minutes), it is moved back to the verified VLAN. When a device is moved between VLANs or a flooding attack is detected, a notification is sent to the UI. Gauge periodically collects these statistics from devices in both VLANs. We abstract all these lower level details away from the user to create an intuitive system.

Chapter 9

Testing

9.1 UI Testing

- Test if application displays appropriate design
- Test if app pages have all appropriate links
- Test if app receives information quickly
- Test if workflow executes successfully

We checked if our user interface is user friendly and easy to use. We invited 5 people with no internet or computer background to use our mobile application and test if they could use our application without any questions. We tested each swift story board and tested every one of them is connected correctly with each other. By using penetration testing, we tested each button's connection to the next story board that it is connected to. We ensured that every package is successfully delivered to our mobile application. We tested if our application receives information quickly. We tested our TLS connection independently of our system to ensure a secure channel which the app and controller communicate. We tested the connection speed to the app and ensure the information is received in the timely manner. Lastly, we performed testing no workflow to ensure every features, functionality can be executed successfully without any bugs.

We want to hide low-level, abstract details from our users. During testing stage for user interface, we make sure our design is user friendly and appropriate. We make sure every swift story board is linked correctly with each other. We needed to ensure that data transfer between controller and mobile application to be fast and secure. Lastly, we tested every functionality of the mobile application to ensure that every workflow executes successfully.

9.2 Machine Learning Testing

- Test error from random forest using required features
- K-fold cross validation

The machine learning algorithm used was random forest. This algorithm was used to classify a device into one of five different device types and was used to detect if basic flooding types were occurring. The flood types were UDP flooding, ICMP flooding, and SYN flooding. The data from these flooding attacks was gathered from the SIoT (Santa Clara Internet of Things) Lab. The five device profiles were switches and triggers, cameras, hubs, light bulbs, and new smart thing devices, and this data was from UNSW Sydney's Open Source data from their IoT testbed.

Flow information was extracted from the software-defined switch and all of the features were purely stateless. Furthermore, these features were all collected at defined 120 second time intervals. These features were the percent of UDP, TCP, and ICMP packets, the total packets sent, the average packet length, and the ratio of communication. These features were collected every 120 seconds from each IP address on the lower privilege VLAN. The ratio of communication is the number of unique destination IP addresses divided by the total number of packets sent. This is used as a measure of how much traffic is split over unique partners between a device and its endpoints.

Random forest was decided as the optimal algorithm because the principle component analysis demonstrated that about 99 percent of the explained variation was within the first few components. This highlighted that a model that could balance both linear models and classification would be optimal. Therefore, we chose random forest and had a very high accuracy in our experimentation

Using data from our two sources, the first experiment was a simple splitting of the data into training and testing sets. 75 percent of the data was training while the remaining 25 percent was testing data that the model had not seen before. This split was done randomly over ten different iterations and averaged to 99.7 percent accuracy. After this a K-Fold Cross Validation was done. K was set to 10 and the average accuracy was 99.8 percent.

These were very strong results on data that the model had not seen.

9.3 Network Testing

- Test if switch and controller can connect
- Test if switch and hosts can connect
- Test if hosts can reach internet through switch

We initially tested the basic setup on GNS3 using ping to ensure the switch was operational. After confirming connectivity, we gathered the statistics via Gauge and Valve functions and parsed them to a log file to be read by our machine learning algorithm. Tcpreplay enabled us to take packet capture files and replay them. We replayed normal and attack IoT data from the University of New South Wales. For our normal traffic, we added options to tcpreplay to loop the pcap file 50 times. For our attack traffic, we similarly added options to send packets as quickly as possible and loop 50 times. We tested the ACL rules to drop any and all communication between the two VLANs.

9.4 System Testing

- Test connectivity between controller and switch
- Test if system gets correct device types
- Test if system can detect an occurring flooding attack
- Test if system can move devices between VLANs

First, we tested the controller switch connection to ensure that they are connected on the same datapath and could send OpenFlow messages. We ensured that the IP's on both devices corresponded to each other. The connectivity between controller and switch was a very useful indicator during debugging. If we restarted the controller and it wouldn't connect, we knew that we had an error in our code. Only when there were no errors in our code would both Faucet and Gauge connect to the switch. When a device is sending data, we tested that the correct device type was assigned to that device. Most of this testing regarding device type was done in the machine learning unit testing; independent of our virtualized network. Testing device types in GNS3 was not difficult due to the high accuracy of our machine learning model. Then, we tested whether or not the model could detect a flooding attack in the unverified VLAN. Lastly, we tested if we can move devices between VLANs.

Chapter 10

Difficulties Encountered

We had numerous difficulties during this project. Our issues started after COVID-19 rose early this year. We had a good amount of progress on our physical set-up. We planned to pass around the raspberry pi's so we could each individually work on it. However, we were forced to take our project virtual and that caused us to lose almost all of our progress on the setup. We were not able to get funding for the project so we couldn't buy all the necessary hardware, i.e. linux box, multiple raspberry pi, ethernet cables, ethernet adapters, and monitors. Furthermore, we were not able to use the design center nor graduate building for their resources. This forced us to individually work on the project rather than together. Initially, the new technology made it difficult to progress fast and we couldn't meet up to figure it out together. In the future, communication will help mitigate these problems.

10.1 Group Meetings

- Meeting time agreement
- Meeting frequency
- Timely responses to team members

Our group was very busy and comprised of juniors and seniors. This caused us to have conflicting meeting availability. Unfortunately, this caused sparse meetings and untimely responses from team members. After COVID-19 began, our time conflicts due to moving and virtual meetings posed to be less efficient than those on campus.

10.2 COVID-19

- Caused us to lose all our progress on physical set up
- Forced us to migrate to a virtual set up
- New set up required new technologies to virtualize the network

- Needed to independently troubleshoot new components on the fly
- Isolated team members

COVID-19 was definitely the largest difficulty we faced during our project. Given that our meeting times were very limited due to our schedules, this caused even more segregation of team members. We could no longer hold physical meetings nor work on our physical set up. This led to miscommunication because we would mostly communicate through text. In addition, we now could not physically work together on the project which forced us to visualize our network. Since COVID-19 began midway through Winter quarter, we lost a lot of progress that we had made the last quarter and a half.

10.3 Centralized Virtualization

- High overhead on host laptop from VM and GNS3
- Too much overhead to run app simultaneously with GNS3 setup on host laptop

Visualizing our network on a single Macbook pro posed to be troublesome. Since we ran 4 Ubuntu virtual machines, this caused the laptop to become unresponsive at times. This also caused an issue when running the app alongside GNS3, as the laptop ran at peak CPU usage. We originally planned to set this up with real, physical devices rather than in a virtual environment.

Chapter 11

Ethics

11.1 User Information

Our system is ethical and does not invade on a user's privacy. We take into account the user's privacy by refraining from collecting and holding non-pertinent information for our classification and detection algorithms.

- Collecting only flow metrics
- Metrics are used only for our algorithms
- Metrics deleted every 120 seconds
- Metrics held locally
- Metrics are not sent to anyone
- Malicious Activities held for 30 days

Our system collect flow information about from the users network. We are not collecting personal data like devices such as the Amazon Echo. We also do not have an account system, so we do never know who is using the device. This information is solely used for the algorithm and all metrics except for device type are deleted every 120 seconds. In addition, all of the metrics the system collects is held locally and are not sent to anyone. Malicious activities are recorded on the users phone will be held for one month. This ensures that the user will have an active record of the activities on their network. Home users have a right to information security and our device is compliant with the user's rights.

Chapter 12

Conclusion

This project allowed us to apply numerous software defined networks theories and devise a method to mitigating DDoS attacks. VLAN partitioning provides a way of securing a section of your network. Another great technique we utilized was stateless flow information, which ultimately reduces the overhead of our system. By being able to classify device types and providing VLAN partitioning to split non-verified IoT devices from verified IoT devies, we are able to help separate malicious actors from the privileged section of the network. Furthermore, our same features are used to enforce policy on the privileged sector of the VLAN by allowing for 10 percent deviation from the average of all of the features collected in the lower privileged sector of the LAN. This allows for a per device enforcement policy that is local to that device in the specific environment it is in within the home. This novel enforcement strategy is coupled with a custom user interface and is able to be implemented on commodity such as a raspberry pi driving the price point down for home users. This system is all open source so any developer can improve or use our code. Despite the setbacks that COVID-19 has caused for our group, we learned a vast amount about networking, important features with respect to DDoS attacks, and analysis of stateless information in a software defined network environment.

12.1 Suggested Changes

- More elaborate UI

We would like to create a more user friendly UI. Currently, our UI is functional but not aesthetic. We want to invite someone who specializes in front-end design to help us design and decorate our UI.

- Implement more user interaction

We would like to develop more features for our user interface in the future. Currently a user is limited to two features, adding more features can allow users to gain more control over their home internet.

- Monthly activity report

We would like to give our users a monthly activity report on their home internet every month. Doing so can give users a clear picture of their own internet status.

12.2 Lessons Learned

- Necessity of planning for the unexpected

Covid-19 pandemic happened quickly and surprisingly. Our team did not plan for such dramatic change and had to adjust the project so we could provide a functional product while working remotely and independently. This caused some challenges along the way. We should plan for any unexpected situation that might come up during the development period.

- Need for a strict schedule

During the development period, our team fell behind on our development timeline. In the future, we would like to establish a strict schedule so we can get ahead of our schedule.

- Splitting up work when teammates are working remotely

Due to the unexpected Covid-19 pandemic, our team had to work remotely on the project. This resulted in one person doing more work than others, in the future we would like to plan ahead and split up the work evenly.

Appendix A

Installation Guide

Due to COVID-19, we did not create a plug n play physical box to implement our system. However, this installation guide outlines how to setup our setup in both virtual and physical settings.

A.1 Installing the system

Make sure you are running with root priveledges on all devices.

On an appropriate devices (e.g. a Linux boxes or VM's), install the Faucet SDN and Open vSwitch packages by running the following scripts:

<https://github.com/cbatula/Senior-Design/tree/master/installScripts/faucet.sh>

<https://github.com/cbatula/Senior-Design/blob/master/installScripts/ovs.sh>

In Open vSwitch, copy the netplan to /etc/netplan. Make sure you are running Ubuntu, or you will need to translate the netplan file into the correct format.

<https://github.com/cbatula/Senior-Design/tree/master/OpenFlowSwitch/netplan>

Afterwards, run the following scripts:

<https://github.com/cbatula/Senior-Design/tree/master/OpenFlowSwitch/>

<https://github.com/cbatula/Senior-Design/tree/master/OpenFlowSwitch>

In Faucet, copy the netplan to /etc/netplan. Make sure you are running Ubuntu, or you will need to translate the netplan file into the correct format.

<https://github.com/cbatula/Senior-Design/blob/master/faucet/netplan>

Afterwards, copy the following files to /lib/python3/dist-packages/faucet:

https://github.com/cbatula/Senior-Design/blob/master/faucet/gauge_prom.py

<https://github.com/cbatula/Senior-Design/blob/master/faucet/TRAINDATA.csv>

Then, copy the following files to /etc/faucet:

<https://github.com/cbatula/Senior-Design/blob/master/faucet/faucet.yaml>

<https://github.com/cbatula/Senior-Design/blob/master/faucet/acls.yaml>

After adding all the files, run these commands to restart Faucet and Gauge:

```
systemctl reload faucet
```

```
systemctl restart gauge
```

Connect your switch to your router. Then connect the hosts to the switch via wireless or GNS3.

Follow these steps to ensure the set up is connected:

- On OVS, run `ovs-vsctl show`; Should show controller: connected for both Faucet and Gauge
- If you are virtualizing the network, attempt to ping google to ensure connectivity through the switch

After these steps, the system will be monitoring on your network.

Appendix B

User Manual

B.1 Manage Websites

1. Block a website
2. Unblock a website

B.2 View Malicious Activity

- View malicious activities on network

References

[19] Jan. 2019. URL: <https://priceconomics.com/the-iot-data-explosion-how-big-is-the-iot-data/>.

[Siv] Arunan Sivanathan. *Classifying IoT Devices in Smart Environments Using ...* URL: <https://ieeexplore.ieee.org/document/8440758>.