

Intelligent Secure Network Switch

Abdullah Naser, Mario Hany, Mostafa Hassan, Mostafa Sedik,

Seif Taher, Youssef El-Khawaga

Supervised By:

Prof. Dr. Ayman Bahaa El-din

Computer and Systems Engineering Department

Faculty of Engineering, Ain Shams University

Cairo, Egypt

July 24, 2021



Computer and Systems Engineering Department

Faculty of Engineering, Ain Shams University

Cairo, Egypt

June 12, 2021

Intelligent Secure Network Switch

A Report Submitted in Partial Fulfillment of the Requirements of the Degree of
Bachelor of Science in Computer and Systems Engineering

Declaration

We hereby certify that this project submitted as part of our partial fulfillment of BSc in Computer and Systems Engineering is entirely our own work, that we have exercised reasonable care to ensure its originality and does not to the best of our knowledge breach any copyrighted materials and have not been taken from the work of others and to the extent that such work has been cited and acknowledged within the text of our work.

Name	Code
Abdallah Naser	1600813
Mario Hany	1501103
Mostafa Hassan	1601408
Mostafa Sedik	1601427
Seif Taher	1600678
Youssef El-Khawaga	1601737

AKNOWLEDGMENT

We would like to express our great appreciation to *Prof. Dr. Ayman Bahaa El-din* the supervisor of our graduation project for his valuable and constructive suggestions during the planning and development of this project. We deeply appreciate that he gave us his time, knowledge, and superlative experience too generously, and also his support to achieve the required output with the required quality.

We are also taking this opportunity to express our gratitude to all members of the Department of Computer and Systems Engineering for their assistance and support.

We also would like to extend our thanks to Dr. Issa Traore, UVic Electrical & Computer Engineering for providing the dataset used in our thesis.

Abstract

The security of computer networks is continuing to become one of the major challenges facing industry enterprises. The need to offer high protection levels against malicious attacks especially that with the rapid development of digitalization, the number of high value targets is increasing. As a result, the use of machine learning trained models for detection of different attacks is being used increasingly. Our project focuses on developing a trained network switch that can flag and detect malicious traffic going through it.

Table of Contents

1. Introduction	11
1.1 Problem Statement and Proposed Solution	12
1.1.1 Ransomware	13
1.2 Project Requirements	14
1.3 Thesis Organization.....	14
2. Theoretical Background	15
2.1 Software Defined Networking (SDN)	15
2.1.1 SDN architecture.....	16
2.1.2 OpenFlow Switch	19
2.2 Machine Learning.....	20
2.2.1 Data collection and preprocessing	22
2.2.2 Feature Selection	22
2.2.3 Choosing a Training Mode	22
2.2.4 Performance Metrics	29
3. Proposed Design and Implementation	32
3.1 Machine Learning Model	32
3.1.1 Data Collection and Pre-Processing.....	32
3.1.2 Feature Extraction Module	35
3.1.3 Training Model	41
3.2 Switch Controller	44
4. Testing and results	46
4.1 Testing on Mininet simulator.....	46

4.1.1 Testing Scenario 1	46
4.1.2 Testing Scenario 2	47
4.1.3 Testing Scenario 3	49
4.2 Testing on TI-mr4320 switch.....	51
4.2.1 Testing Scenario 1	51
4.2.2 Testing Scenario 2	52
4.2.3 Testing Scenario 3	53
5. Conclusion	54
5.1 Future Work	56
Appendix i	58
References	60

Table of Figures

Figure 1: SDN Architecture	16
Figure 2: Machine Learning Cycle	21
Figure 3: Machine Learning Algorithms.....	23
Figure 4: SVM	24
Figure 5: SVM Linear and Non-Linear Planes.....	25
Figure 6: Decision Tree.....	26
Figure 7: Random Forest Classifier	28
Figure 8: Confusion Matrix.....	29
Figure 9: Machine Learning Model Design	32
Figure 10: pcap file output.....	33
Figure 11: Code Snippet 1	36
Figure 12: Code Snippet 2	37
Figure 13: Code Snippet 3	38
Figure 14: Code Snippet 4	40
Figure 15: Feature Importance	42
Figure 16: Model Evaluation	43
Figure 17: Ryu	44
Figure 18: System Block Diagram.....	45
Figure 19: Ping Output	46
Figure 20: Benign Output.....	47
Figure 21: Full Benign Dump.....	48
Figure 22: Ransomware Dump	49

Figure 23: Ransomware Classification	50
Figure 24: Internet Ping	51
Figure 25: Benign traffic result	52
Figure 26: Ransomware traffic result	53

List of Abbreviations and Symbols

IoT	Internet of Things
QoS	Quality of Service
SDN	Software Defined Networking
SVM	Support Vector Machine
ML	Machine Learning
API	Application Programming Interface
NN	Neural Network
ANN	Artificial Neural Network
DNN	Deep Neural Network

1. INTRODUCTION

The world is becoming more interconnected due to Internet and new networking technology. There is a large amount of personal, commercial, military, and government information on networking infrastructures worldwide. Network security is becoming of utmost importance because of intellectual property that can be easily acquired through the internet. As it relates directly to an organization's business continuity, network security breaches can disrupt e-commerce, cause the loss of business data, threaten people's privacy, and compromise the integrity of information. These breaches can result in lost revenue for corporations, theft of intellectual property, lawsuits, and can even threaten public safety.

While there are many ways computers and networks are attacked daily, malwares are considered one of the most dangerous attacks. The proliferation of internet use in recent years has increased the threat of malware. Malware refers to software that has been designed for some nefarious purpose. Such software can be designed to cause damage to a system, such as by deleting all files, or it can be designed to create a backdoor into the system to grant access to unauthorized individuals. Generally, the installation of malware is done so that it is not obvious to the authorized users.

As a result, many detection and prevention systems are implemented to control the increasing threats of malwares. This document provides a system built using SDN architecture.

1.1 Problem Statement and Proposed Solution

Given the gradual intensification of the current network security situation, malicious attack traffic is flooding the entire network environment, and the current malicious traffic detection model is insufficient in detection efficiency and detection performance.

The aim of the project is to design and implement a secure and smart data packet switch. The switch itself is based on the latest Software Defined Networks (SDN) Architecture where the switching and routing algorithm is implemented on a controller and the switch itself is focused only on data forwarding based on the rules dictated by the controller through the OpenFlow Protocol.

The main new feature of the proposed switching system is the ability to intelligently classify the traffic into safe or in-safe based on an intelligent classifier that learns the legitimate traffic from the malicious one.

The detection method used gains inspiration from machine learning technologies. Recently, machine learning algorithms have emerged as a promising solution to identify anomalous packets. The model extracts header features and trains a binary classifier to perform packet inspection.

Through the chapters of this document, we will go through an approach to detect and classify malicious packets.

1.1.2 Ransomware

Ransomware is an ever-evolving form of malware designed to encrypt files on a device, rendering any files and the systems that rely on them unusable. Malicious actors then demand ransom in exchange for decryption. Ransomware actors often target and threaten to sell or leak exfiltrated data or authentication information if the ransom is not paid. In recent years, ransomware incidents have become increasingly prevalent among the world's government entities and critical infrastructure organizations.

Malicious actors continue to adjust their ransomware tactics over time, to include pressuring victims for payment by threatening to release stolen data if they refuse to pay, and publicly naming and shaming victims as secondary forms of extortion. Malicious actors engage in lateral movement to target critical data and propagate ransomware across entire networks. These actors also increasingly use tactics, such as deleting system backups, that make restoration and recovery more difficult or infeasible for impacted organizations.

Anyone with a computer connected to the internet and anyone with important data stored on their computer or network is at risk, including government or law enforcement agencies and healthcare systems or other critical infrastructure entities.

Ransomware can be devastating to an individual or an organization. Some victims pay to recover their files, but there is no guarantee that they will recover their files if they do. Recovery can be a difficult process that may require the services of a reputable data recovery specialist. Ransomware incidents can severely impact business processes and leave organizations without the data they need to operate and deliver mission-critical services. The monetary value of ransom demands has increased, with some demands exceeding \$1 million.[11]

Ransomware incidents have become more destructive and impactful in nature and scope. The economic and reputational impacts of ransomware incidents, throughout the initial disruption and, at times, extended recovery, have also proven challenging for organizations large and small.

Notable ransomware software packages include:

- Reveton
- CryptoLocker
- CryptoLocker.F and TorrentLocker
- CryptoWall
- WannaCry
- Petya
- Cerber

1.2 Project Requirements

- Developing a switch controller that uses OpenFlow protocol to:
 - Handle incoming packets.
 - Build the Flow Table and add flows to it.
 - Be able to monitor network health
 - Exploit multithreading techniques to separate switch basic operations from machine learning data preprocessing and prediction.
- Develop and Build a Machine Learning Model that:
 - Classifies traffic to malicious and safe based on the presence of ransomware.
 - Extracts ransomware unique features from the traffic to feed it into the model.
 - Guarantees minimum prediction latency.
- Implement the controller on a real switch that can:
 - Request IP address and connect to the internet.

1.3 Thesis Organization

- **Chapter 2:** Describes our theoretical background prior to this project deep diving especially into SDN, OpenFlow protocol and Machine Learning.
- **Chapter 3:** Describes in detail our proposed system requirements, design and our switch controller implementation using RYU Controller.
- **Chapter 4:** Goes through the testing and results of our smart switch implementation.

2. THEORETICAL BACKGROUND

2.1 Software Defined Networking (SDN)

It is defined an architecture that abstracts different, distinguishable layers of a network in order to make networks agile and flexible. The goal of SDN is to improve network control by enabling enterprises and service providers to respond quickly to changing business requirements. SDN technology focuses on the separation of the network control plane from the data plane. While the control plane makes decisions about how packets should flow through the network, the data plane moves packets from place to place.[2]

In a software-defined network, a network engineer or administrator can shape traffic from a centralized control console without having to touch individual switches in the network. A centralized SDN controller will direct the switches to deliver network services wherever they are needed, regardless of the specific connections between a server and devices.[1]

This process is a move away from traditional network architecture, in which individual network devices make traffic decisions based on their configured routing tables. SDN has had a role in networking for a decade now, and its evolution and roles have continued to evolve.[1]

2.1.1 SDN architecture

A typical representation of SDN architecture comprises three layers: the application layer (Control Plane), the control layer (Control Plane) and the infrastructure layer (Data Plane).

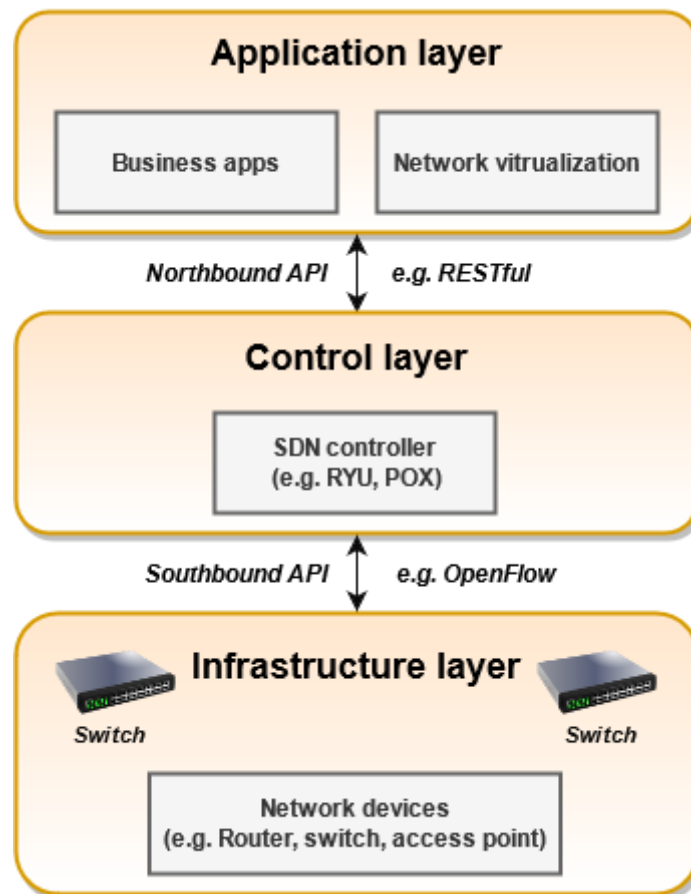


Figure 1: SDN Architecture

1. The application layer, not surprisingly, contains the typical network applications or functions organizations use. This can include intrusion detection systems, load balancing or firewalls. Where a traditional network would use a specialized appliance, such as a firewall or load balancer, a software-defined network replaces the appliance with an application that uses a controller to manage data plane behavior.[1]
2. The control layer represents the centralized SDN controller software that acts as the brain of the software-defined network. This controller resides on a server and manages policies and the flow of traffic throughout the network.
3. The infrastructure layer is made up of the physical switches in the network.

A controller's functionality can be broadly organized into three layers:

1. **A communication layer communicating between the SDN controller and controlled network devices.** Clearly, if an SDN controller is going to control the operation of a remote SDN-enabled switch, host, or other device, a protocol is needed to transfer information between the controller and that device. In addition, a device must be able to communicate locally observed events to the controller (e.g., a message indicating that an attached link has gone up or down, that a device has just joined the network, or a heartbeat indicating that a device is up and operational). These events provide the SDN controller with an up-to-date view of the network's state. This protocol constitutes the lowest layer of the controller architecture, as shown in **Figure 1**. The communication between the controller and the controlled devices cross what has come to be known as the controller's "**southbound**" interface. Examples of Southbound protocols are OpenFlow, Netconf & OVSDB.[2]
2. **A network-wide state-management layer.** The ultimate control decisions made by the SDN control plane—e.g., configuring flow tables in all switches to achieve the desired end-end forwarding, to implement load balancing, or to implement a particular firewalling capability—will require that the controller have up-to-date information about state of the networks' hosts, links, switches, and other SDN-controlled devices. A switch's flow table contains counters whose values might also be profitably used by network-control applications; these values should thus be available to the applications. Since the ultimate aim of the control plane is to determine flow tables for the various controlled devices, a controller might also maintain a copy of these tables. These pieces of information all constitute examples of the network-wide "state" maintained by the SDN controller.[2]

3. **The interface to the network-control application layer.** The controller interacts with network control applications through its “**northbound**” interface. This API allows network-control applications to read/write network state and flow tables within the state management layer. Applications can register to be notified when state-change events occur, so that they can take actions in response to network event notifications sent from SDN-controlled devices. An example of northbound interface is RESTful API & Intent.[2]

SDN Controllers Examples:

- ONOS: Open Network Operating System is designed to be distributed, stable and scalable with a focus on Service Provider networks. It is written in Java.
- ODL: is a modular open platform for customizing and automating networks of any size and scale. The OpenDayLight Project arose out of the SDN movement, with a clear focus on network programmability. It was designed from the outset as a foundation for commercial solutions that address a variety of use cases in existing network environments. It is written in Java.
- Ryu: is structured differently from other solutions in that it provides simple supporting infrastructure that users of the platform must write code to utilize as desired. While this requires development expertise, it also allows complete flexibility of the SDN solution. It is written in Python.

Primary Advantages of SDN:

1. Centralized network provisioning: SDN helps centralize enterprise management and provisioning by offering a unified perspective on the whole network. SDN can also speed up service delivery and boost agility in provisioning virtual and physical network devices in a central location.
2. Lower operating costs: Several benefits to SDN, such as having an efficient administration, server utilization improvements, and improved virtualization control, can dually help cut operating costs.
3. Hardware savings and reduced capital expenditures: SDN adoption helps revive older network devices and simplifies the process of optimizing commoditized hardware. By following the instructions from the SDN controller, older hardware can be repurposed while less costly hardware can be deployed to optimal effect.

4. More granular security: Virtual machines pose a challenge for firewalls and content filtering, a challenge that's further compounded by personal devices. By establishing a central control point for regulating security and policy information for your enterprise, the SDN controller quickly becomes a boon for your IT department. [3]

SDN Challenges:

1. Security is both a benefit and a concern with SDN technology. The centralized SDN controller presents a single point of failure and, if targeted by an attacker, can prove detrimental to the network.
2. There is no established definition of "software-defined networking" in the networking industry. Different vendors offer various approaches to SDN, ranging from hardware-centric models and virtualization platforms to hyper-converged networking designs and controllerless methods.[1]

2.1.2 OpenFlow Switch

An OpenFlow Switch consists of one or more flow tables and a group table, which perform packet lookups and forwarding, and an OpenFlow channel to an external controller. The switch communicates with the controller and the controller manages the switch via the OpenFlow protocol. Using the OpenFlow protocol, the controller can add, update, and delete flow entries in flow tables, both reactively (in response to packets) and proactively.[4]

Matching starts at the first flow table and may continue to additional flow tables. Flow entries match packets in priority order, with the first matching entry in each table being used. If a matching entry is found, the instructions associated with the specified flow entry are executed. If no match is found in a flow table, the outcome depends on configuration of the table-miss flow entry where Instructions associated with each flow entry either contain actions or modify pipeline processing. Actions included in instructions describe packet forwarding, packet modification and group table processing. Flow entries may forward to a port. This is usually a physical port, but it may also be a logical port or a reserved port.

Switch designers are free to implement the internals in any way convenient, provided that correct match and instruction semantics are preserved.

2.2 Machine Learning

Machine Learning is basically classified into four categories: supervised, unsupervised, semi-supervised and reinforcement learning. Brief explanations of these algorithms are outlined below.

- **Supervised Machine Learning:** It is built by supplying the “system” with a “training data” i.e., inputs and their known outcomes/outputs to enable it to create a relation. It is then provided new inputs to predict based on the relationship learnt. The training data is described as “labelled”. Supervised learning problems are categorized into “regression” and “classification” problems. In regression problems, the model tries to predict results within a continuous output whereas in classification, the model predicts results in a discrete output. Some of the commonly used supervised learning algorithms include decision trees, k-nearest neighbors, random forest, neural networks, and support vector machines.
- **Unsupervised Machine Learning:** It is the exact opposite of the supervised approach. In this model, the system is given a set of input data without their corresponding outcomes. The model clusters the data based on relationships among the “features” in the data. Popular examples of such algorithms include k-means and self-organizing maps.
- **Semi-Supervised Machine Learning:** It is midpoint between supervised and unsupervised learning. In this model, part of the input data has their corresponding outputs(labelled) while the remainder do not have their corresponding outputs(unlabeled).
- **Reinforcement learning:** It is the task of learning how agents ought to take sequence of actions in an environment to maximize cumulative rewards. An agent relates with an environment to learn the best actions to take to maximize the long-term reward.

Machine Learning has been applied in several fields to improve human living. Examples include the use of machine learning for natural language processing (NLP), medical diagnosis, customer segmentation, product recommendation and facial recognition. A typical machine learning application process entail: Data collection and preprocessing, feature engineering and training model choice, training and evaluating the model, fine tuning the trained model, using the model to predict new instances.

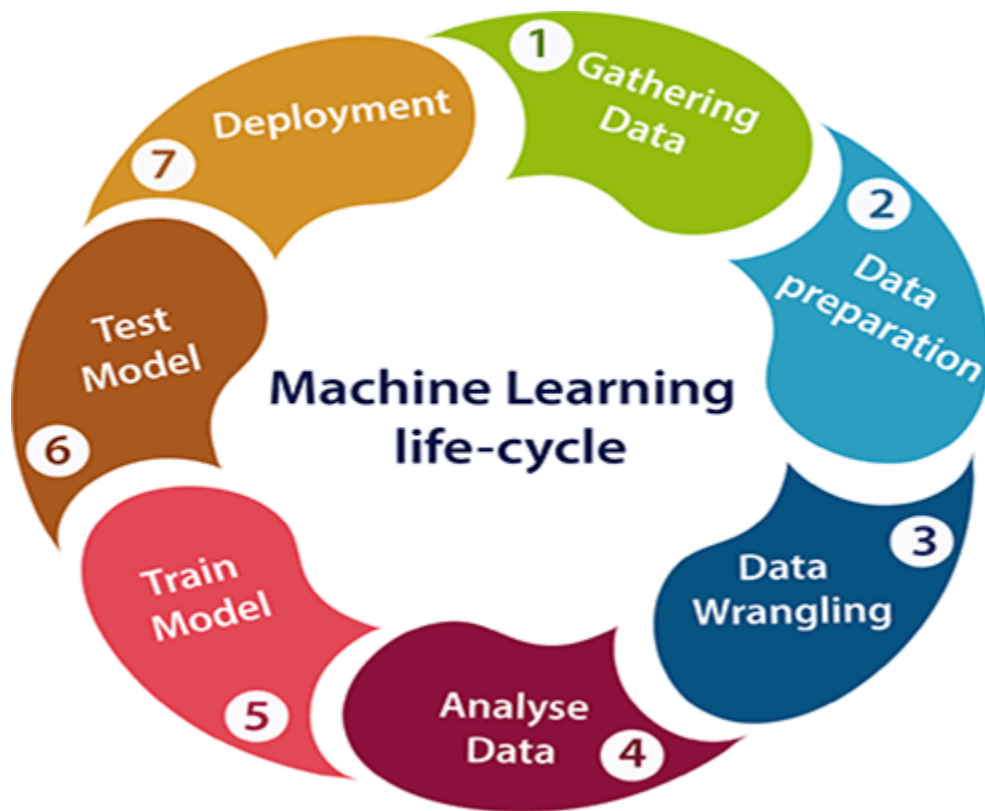


Figure 2: Machine Learning Cycle

2.2.1 Data collection and preprocessing

Data collection and preparation is a part of building machine learning models. Based on the outlined machine learning problem, corresponding data is collected. A lot of datasets are available online depending on the research problem. Data is required to be cleaned and preprocessed to make sure the ML model is being feed with valid data that correctly models the problem considered.

2.2.2 Feature Selection

Feature Selection is another vital step in the machine learning process. A feature is an individual measurable property or characteristic of a phenomenon observed. It is information that can aid in the prediction or classification. Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. The goal is to achieve features that improve the accuracy of the model. In feature engineering, better features can be created by combining other features in the dataset. Feature selection aims at removing irrelevant and redundant attributes for the purpose of increasing learning accuracy. Feature selection enhances the learning accuracy and speed up the learning process of algorithm.

2.2.3 Choosing a Training Mode

The next step is choosing the best algorithm to use. There are several algorithms to choose from. A ML algorithm can be based on supervised, unsupervised, semi-supervised or reinforcement learning. Figure 3 outlines a list of commonly used ML algorithms. The ones widely used in network traffic classification and routing are Random Forest, Decision Tree Classifier, K-Nearest Neighbors and Neural networks.

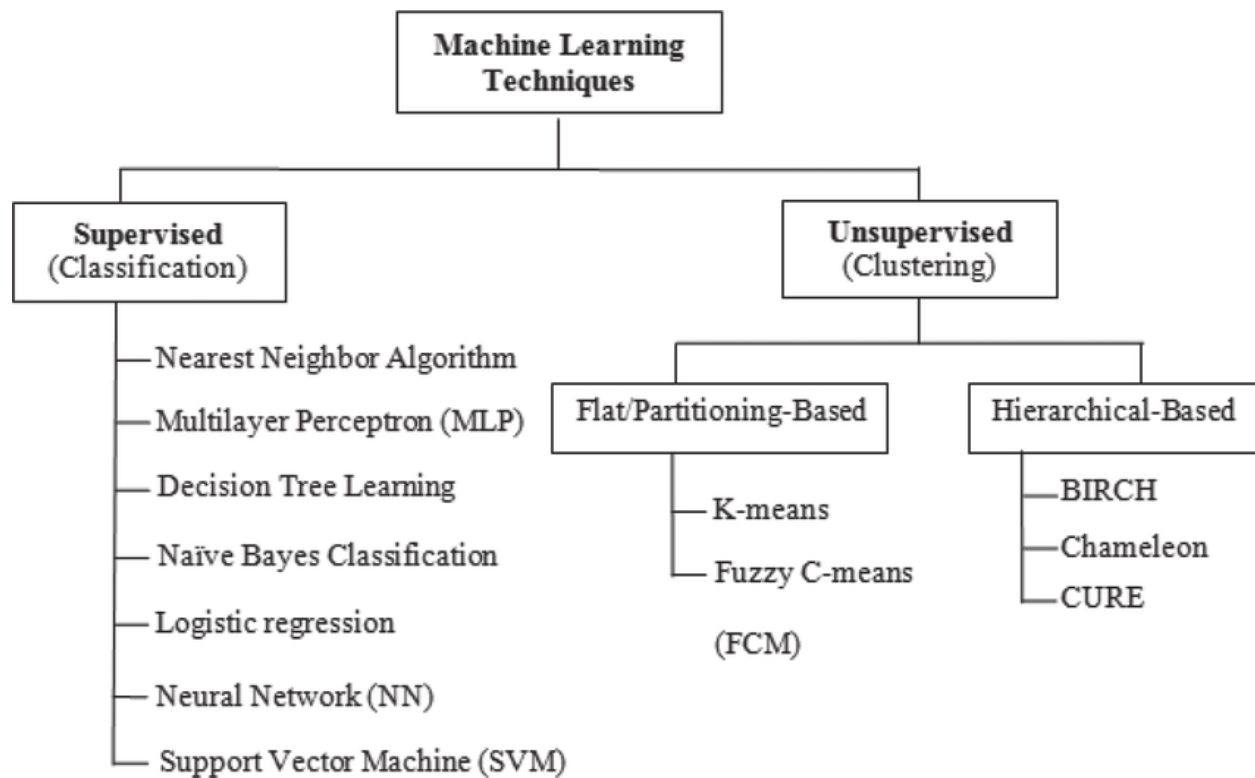


Figure 3: Machine Learning Algorithms

SVM

Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well (figure 3).

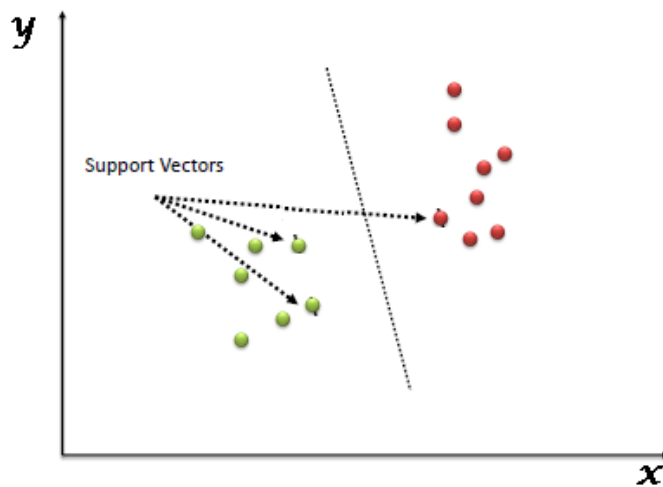


Figure 4: SVM

Support Vectors are simply the co-ordinates of individual observation. The SVM classifier is a frontier which best segregates the two classes (hyper-plane/ line).

SVM, formally described by Vapnik [14], has the ability to define nonlinear decision boundaries in high-dimensional variable space by solving a quadratic optimization problem. The key idea of this concept is to identify the hyperplane separating different classes such that the induced distance between the hyperplane and the training patterns is maximal. Basic SVM theory states that for a nonlinearly separable dataset containing points from two classes there are an infinite number of hyperplanes that divide classes.

The selection of a hyperplane that optimally separates two classes (i.e., the decision boundary) is carried out using only a subset of training samples known as support vectors. The maximal margin M (distance) between the support vectors is taken to represent the optimal decision boundary. In non-separable linear cases, SVM finds M while incorporating a cost parameter C , which defines a penalty for misclassifying support vectors. High values of C generate complex decision boundaries in order to miss-classify as few support vectors as possible.

For problems where classes are not linearly separable, SVM uses an implicit transformation of input variables using a kernel function, which allows SVM to separate nonlinearly separable support vectors using a linear hyperplane. Selection of an appropriate kernel function and kernel width, s , are required to optimize performance for most applications. SVM can be extended to multiclass problems by constructing $c(c-1)/2$ binary classification models, the so-called one-against-one method, which generates predictions based on a majority vote.

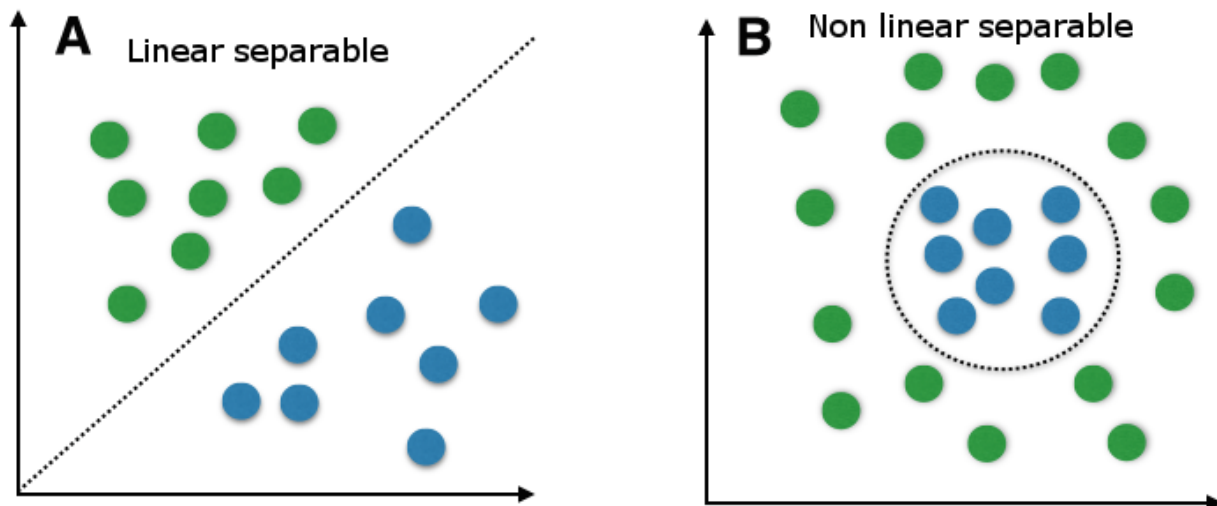


Figure 5: SVM Linear and Non-Linear Planes

Decision Tree Algorithm

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too.

The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data(training data).

In Decision Trees, for predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with the record's attribute. Based on comparison, we follow the branch corresponding to that value and jump to the next node.

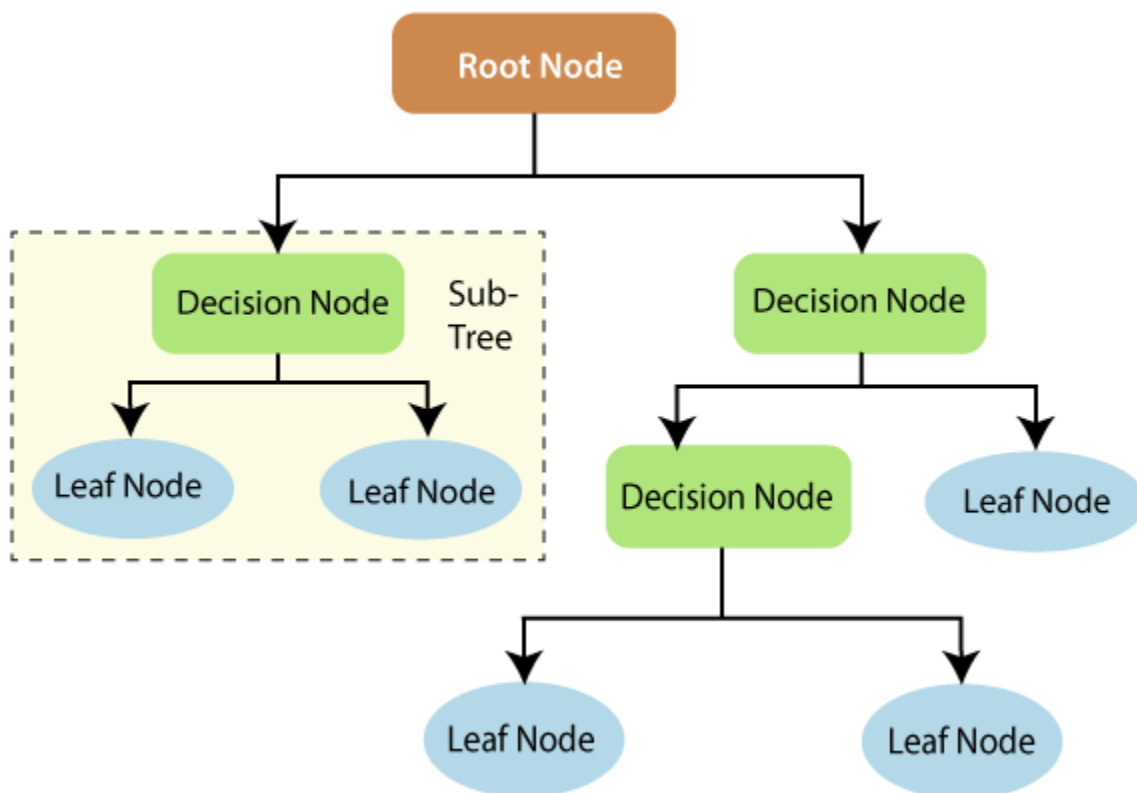


Figure 6: Decision Tree

Random Forest Algorithm

Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

The (random forest) algorithm establishes the outcome based on the predictions of the decision trees. It predicts by taking the average or mean of the output from various trees. Increasing the number of trees increases the precision of the outcome.

A random forest eradicates the limitations of a decision tree algorithm. It reduces the overfitting of datasets and increases precision. It generates predictions without requiring many configurations in packages (like scikit-learn).

Features of a Random Forest Algorithm:

- It's more accurate than the decision tree algorithm.
- It provides an effective way of handling missing data.
- It can produce a reasonable prediction without hyper-parameter tuning.
- It solves the issue of overfitting in decision trees.
- In every random forest tree, a subset of features is selected randomly at the node's splitting point.

Classification in random forests:

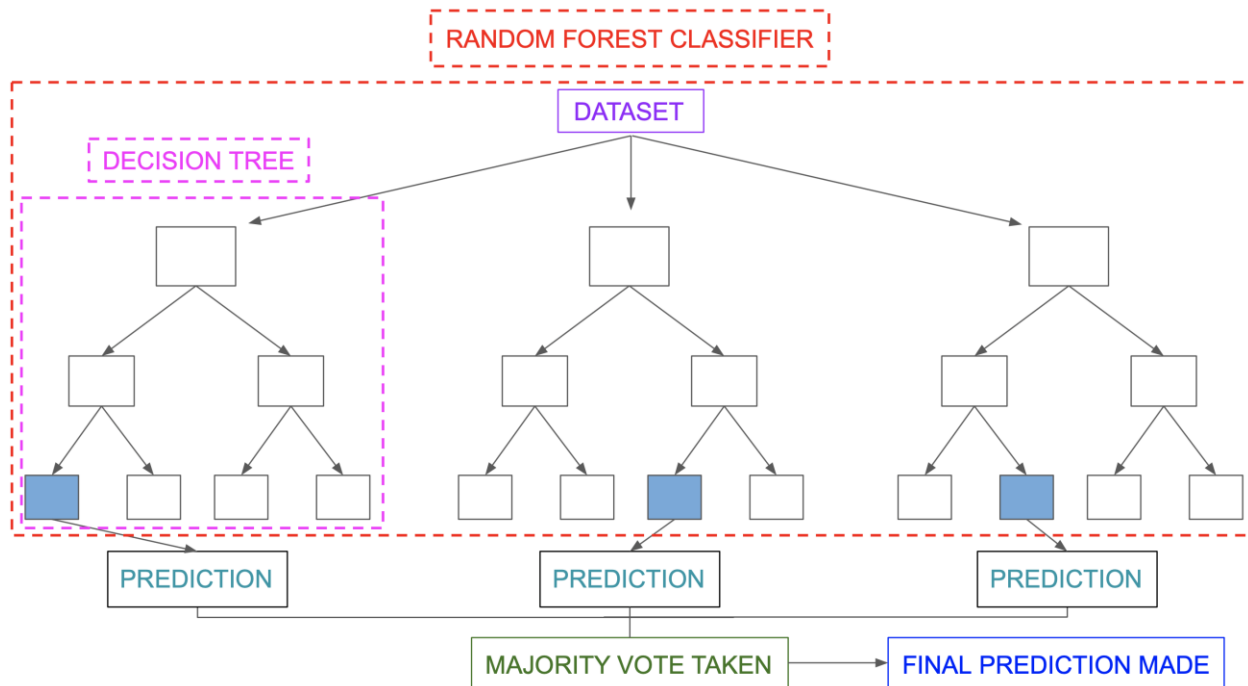


Figure 7: Random Forest Classifier

Classification in random forests employs an ensemble methodology to attain the outcome. The training data is fed to train various decision trees. This dataset consists of observations and features that will be selected randomly during the splitting of nodes.

A rain forest system relies on various decision trees. Every decision tree consists of decision nodes, leaf nodes, and a root node. The leaf node of each tree is the final output produced by that specific decision tree. The selection of the final output follows the majority-voting system. In this case, the output chosen by the majority of the decision trees becomes the final output of the rain forest system. Figure 7 shows a simple random forest classifier.[13]

2.2.4 Performance Metrics

Performance metrics are a part of every machine learning pipeline. They are used to evaluate Machine Learning Algorithms and judge their performance. They are not only used as the criteria to evaluate learning algorithms, but also used as the heuristics to construct learning models.

Metrics are different from loss functions. Loss functions are functions that show a measure of the model performance and are used to train a machine learning model (using some kind of optimization) and are usually differentiable in model's parameters. On the other hand, metrics are used to monitor and measure the performance of a model (during training, and test), and do not need to be differentiable.[7]

Different performance metrics are used to evaluate different Machine Learning Algorithms. For example some of the metrics that are mostly used for classification are (accuracy, precision, recall).

Basic terminologies used in classification model:

Confusion matrix: is a tabular visualization of the model predictions versus the ground-truth labels. Each row of confusion matrix represents the instances in a predicted class and each column represents the instances in an actual class.

The Confusion matrix is not a performance measure as such, but almost all the performance metrics are based on Confusion Matrix and the numbers inside it.[8]

Actual	Positive	TP	FN
	Negative	FP	TN
		Positive	Negative
		Predicted	

Figure 8: Confusion Matrix

- **True Positives (TP):** True positives are the cases when the actual class of the data point was (True) and the predicted is also (True).
- **True Negatives (TN):** True negatives are the cases when the actual class of the data point was (False) and the predicted is also (False).
- **False Positives (FP):** False positives are the cases when the actual class of the data point was (False) and the predicted is (True). False is because the model has predicted incorrectly and positive because the class predicted was a positive one. Should be minimized.
- **False Negatives (FN):** False negatives are the cases when the actual class of the data point was (True) and the predicted is (False). False is because the model has predicted incorrectly and negative because the class predicted was a negative one. Should be minimized.

Classification Metrics:

- **Accuracy:**
Accuracy in classification problems is defined as the number of correct predictions divided by the total number of predictions.[8]

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+FN+TN}$$

Accuracy is a good measure when the target variable classes in the data are nearly balanced and should never be used as a measure when the target variable classes in the data are a majority of one class.[8]

- **Precision:**
By using the confusion matrix example given above, Precision is defined as a measure that tells us what proportion True Positives were actually true positives.

$$\text{Precision} = \frac{TP}{TP+FP}$$

- **Recall:**
Recall is another important metric, which is defined as the fraction of samples from a class which are correctly predicted by the model.[7]

$$\text{Recall} = \frac{TP}{TP+FN}$$

When to use Precision and When to use Recall: if we want to focus more on minimizing False Negatives, we would want our Recall to be as close to 100% as possible without precision being too bad and if we want to focus on minimizing False positives, then our focus should be to make Precision as close to 100% as possible.[8]

- **F1 Score:**

Depending on application, you may want to give higher priority to recall or precision. But there are many applications in which both recall, and precision are important. Therefore, it is natural to think of a way to combine these two into a single metric. One popular metric which combines precision and recall is called F1-score, which is the harmonic mean of precision and recall defined as:

$$\text{F1-score} = 2 * \text{Precision} * \text{Recall} / (\text{Precision} + \text{Recall})$$

3. PROPOSED DESIGN AND IMPLEMENTATION

In this chapter we will view each component in our project detailed design.

3.1 Machine Learning Model

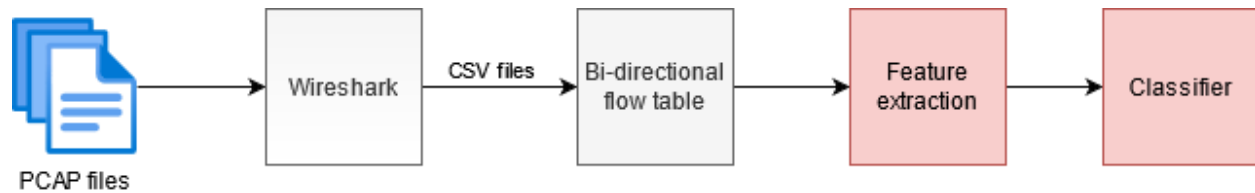


Figure 9: Machine Learning Model Design

3.1.1 Data Collection and Pre-Processing

As stated before, the collection and pre-processing of data is a very important and challenging step in the machine learning cycle as we need to have diverse data of both benign and ransomware traffic to proceed with feature extraction.

For these reasons we chose the ISOT Ransomware Dataset which is a combination of the behavior data for a collection of ransomware samples and benign applications. The ransomware samples were obtained from Virustotal1 under academic license, in addition to obtaining several samples from anti-malware companies. The dataset consists of a total number of 669 ransomware samples representing most of the popular ransomware families and variants that emerged in the wild. In addition to the ransomware samples, the dataset include data from 103 benign applications representing the most popular software applications used by Window users.[9]

The data contained various ransomware families we chose to use Cerber family samples as they were the largest then we used the pcap dump files to get the following info from each packet header (figure 9):

- Source Address
- Destination Address
- Source Port
- Destination Port
- Protocol
- Packet Length
- Packet Time to Live
- The packets interarrival time (Delta Time)
- The time stamp of each packet with respect to the beginning of sample testing

Source	Destination	Source port	Destination port	Protocol	Length	Time to Live	Delta Time	Time
23.59.154.25	192.168.50.11	80	49162	TCP	66	53	0.003026	10.736554
192.168.50.11	23.59.154.25	49162	80	TCP	54	128	0.000216	10.736770
192.168.50.11	23.59.154.25	49162	80	HTTP	151	128	0.033314	10.770084
23.59.154.25	192.168.50.11	80	49162	TCP	54	53	0.005677	10.775761
23.59.154.25	192.168.50.11	80	49162	HTTP	233	53	0.004915	10.780676
192.168.50.11	23.59.154.25	49162	80	TCP	54	128	0.000409	10.781085
23.59.154.25	192.168.50.11	80	49162	TCP	54	53	0.000126	10.781211
192.168.50.11	23.59.154.25	49162	80	TCP	54	128	0.000155	10.781366
23.59.154.25	192.168.50.11	80	49162	TCP	54	53	0.002608	10.783974
192.168.50.11	239.255.255.250	64673	1900	SSDP	175	1	0.110784	10.894758
192.168.50.11	239.255.255.250	64673	1900	SSDP	175	1	0.000004	10.894762
192.168.50.11	239.255.255.250	64673	1900	SSDP	167	1	0.032061	10.926823
192.168.50.11	239.255.255.250	64673	1900	SSDP	167	1	0.000003	10.926826
192.168.50.11	239.255.255.250	64673	1900	SSDP	165	1	0.030791	10.957617
192.168.50.11	239.255.255.250	64673	1900	SSDP	165	1	0.000002	10.957619
192.168.50.11	239.255.255.250	64673	1900	SSDP	175	1	0.047564	11.005183
192.168.50.11	239.255.255.250	64673	1900	SSDP	175	1	0.000002	11.005185

Figure 10: pcap file output

We then started building bi-directional flow tables for each sample by taking a sample every 10 seconds of the duration of the test run of the sample (most were 30 minutes long) [10].

For the bi-directional flow tables, we used the following columns as the key to each flow:

- Source Address
- Destination Address
- Source Port
- Destination Port
- Protocol

After the bi-directional flow table is done, the data is now ready for feature extraction which we will discuss in the following part.

3.1.2 Feature Extraction Module

In this section, after creating the bi-directional flow dataset in the previous section, we define 29 features for each flow based on doing calculations on the extracted header details of each packet (packet length, TTL, delta time).

Calculations used in extracting the features are all based on aggregations and well-known descriptive statistics such as summation, minimum, maximum, mean and standard deviation [12] as well as some custom logic that will be defined in this section later. Here we describe in detail each of the 29 features and the definition of the attributes used.

- outPackets: The total number of outgoing packets sent from the source address to the destination address in the flow.
- Packet length: This attribute specifies the length of the entire IP packet, including both the header and data segments, in bytes. By applying some statistical computations and aggregations, we get the following features.

- sumLengthOut: The summation of the attribute (packet length) for all outgoing packets sent from the source address to the destination address in the flow.

$$sumLengthOut = \sum (packet\ length\ of\ outgoing\ packet_i)$$

- maxLengthOut: The maximum of the attribute (packet length) of all outgoing packets sent from the source address to the destination address in the flow.
- meanLengthOut: The mean of the attribute (packet length) of all outgoing packets sent from the source address to the destination address in the flow.

$$meanLengthOut = \frac{\sum (packet\ length\ of\ outgoing\ packet_i)}{Number\ of\ outgoing\ packets}$$

- stdLengthOut: The standard deviation of the attribute (packet length) of all outgoing packets sent from the source address to the destination address in the flow.

$$stdLengthOut = \sqrt{\frac{\sum |packet\ length\ of\ outgoing\ packet_i - mean\ of\ packet\ lengths|^2}{Number\ of\ outgoing\ packets - 1}}$$

```

outPackets = dict[key]['out'].sum()
inPackets = dict[key]['in'].sum()

try:
    packetRatio = inPackets / outPackets
except:
    packetRatio = float('NaN')

sumLengthOut = dict[key]['packet length'].where(dict[key]['out'] == 1).sum()
sumLengthIn = dict[key]['packet length'].where(dict[key]['in'] == 1).sum()
maxLengthOut = dict[key]['packet length'].where(dict[key]['out'] == 1).max()
maxLengthIn = dict[key]['packet length'].where(dict[key]['in'] == 1).max()
meanLengthOut = dict[key]['packet length'].where(dict[key]['out'] == 1).mean()
meanLengthIn = dict[key]['packet length'].where(dict[key]['in'] == 1).mean()
stdLengthOut = dict[key]['packet length'].where(dict[key]['out'] == 1).std()
stdLengthIn = dict[key]['packet length'].where(dict[key]['in'] == 1).std()

```

Figure 11: Code Snippet 1

- Delta time: This attribute specifies how much time elapsed between the prior and current packet in the uni-directional flow. By applying some statistical computations and aggregations, we get the following features.
 - sumInterArrivalOut: The summation of the attribute (delta time) for all outgoing packets sent from the source address to the destination address in the flow.

$$sumInterArrivalOut = \sum (\text{delta time of outgoing packet}_i)$$

- minInterArrivalOut: The minimum of the attribute (delta time) for all outgoing packets sent from the source address to the destination address in the flow.
- meanInterArrivalOut: The mean of the attribute (delta time) of all outgoing packets sent from the source address to the destination address in the flow.

$$meanInterArrivalOut = \frac{\sum (\text{delta time of outgoing packet}_i)}{\text{Number of outgoing packets}}$$

- stdInterArrivalOut: The standard deviation of the attribute (delta time) of all outgoing packets sent from the source address to the destination address in the flow.

$$stdInterArrivalOut = \sqrt{\frac{\sum |\text{delta time of outgoing packet}_i - \text{mean of delta times}|^2}{\text{Number of outgoing packets} - 1}}$$

```
sumInterArrivalOut = dict[key]['Delta time'].where(dict[key]['out'] == 1).sum()
sumInterArrivalIn = dict[key]['Delta time'].where(dict[key]['in'] == 1).sum()
minInterArrivalOut = dict[key]['Delta time'].where(dict[key]['out'] == 1).min()
minInterArrivalIn = dict[key]['Delta time'].where(dict[key]['in'] == 1).min()
meanInterArrivalOut = dict[key]['Delta time'].where(dict[key]['out'] == 1).mean()
meanInterArrivalIn = dict[key]['Delta time'].where(dict[key]['in'] == 1).mean()
stdInterArrivalOut = dict[key]['Delta time'].where(dict[key]['out'] == 1).std()
stdInterArrivalIn = dict[key]['Delta time'].where(dict[key]['in'] == 1).std()
```

Figure 12: Code Snippet 2

- Burst length: Generally, a burst means to break open a certain sequence or an action. By our definition, we assume that a burst length is the number of uni-directional packets in a flow, that occurs while there are no two consequent packets in the opposite direction. Once two consequent packets in the opposite direction arrive, this burst ends. We build a list of all burst lengths. By applying some statistical computations and aggregations, we get the following features.
 - maxBurstOut: The maximum (burst length) from the built burst lengths list from all outgoing packets sent from the source address to the destination address in the flow.
 - minBurstOut: The minimum (burst length) from the built burst lengths list from all outgoing packets sent from the source address to the destination address in the flow.
 - meanBurstOut: The mean of (burst length) from the built burst lengths list from all outgoing packets sent from the source address to the destination address in the flow.

$$meanBurstOut = \frac{\sum(burst\ lengths\ of\ outgoing\ packet_i)}{Number\ of\ outgoing\ packets}$$

```

if len(burstOut) != 0:
    minBurstOut = float(np.amin(burstOut, axis=0))
    maxBurstOut = float(np.amax(burstOut, axis=0))
    meanBurstOut = float(np.mean(burstOut))
if len(burstIn) != 0:
    minBurstIn = float(np.amin(burstIn, axis=0))
    maxBurstIn = float(np.amax(burstIn, axis=0))
    meanBurstIn = float(np.mean(burstIn))

```

Figure 13: Code Snippet 3

- The following 12 features have the same logic and computations as the last 12 features mentioned above, but with the difference that all computations are done on the ingoing packets not the outgoing.
 - inPackets
 - sumLengthIn
 - maxLengthIn
 - meanLengthIn
 - stdLengthIn
 - sumInterArrivalIn
 - minInterArrivalIn
 - meanInterArrivalIn
 - stdInterArrivalIn
 - maxBurstIn
 - minBurstIn
 - meanBurstIn
- packetRatio: This is the ratio between the number of ingoing packets to the number of outgoing packets in the flow.
- byteRatio: This is the ratio between the total number of bytes of all ingoing packets to the total number of bytes of all outgoing packets in the flow.
- periodicity: Generally, periodicity is the recurrence of something at regularly spaced periods of time. We here exploit the (delta time) attribute by forming a new list containing the difference between the delta times of the whole bi-directional flow in ascending order of their timestamps. By applying some statistical computations on that list, we get the following features.
 - periodicity: The mean of the built list.
 - stdPeriodicity: The standard deviation of the built list.

- TTL (Time to live): This attribute indicates the remaining lifetime of a packet on a network. By applying aggregations, we get the following feature.
 - TTLPercent: This is the ratio between the minimum TTL of the ingoing packets in the flow to the minimum TTL of the outgoing packets in the flow.

$$TTLPercent = \frac{\text{Minimum}(TTL \text{ of ingoing packets})}{\text{Minimum}(TTL \text{ of outgoing packets})}$$

```
minTTLOut = dict[key]['Time to Live'].where(dict[key]['out'] == 1).min()
minTTLIn = dict[key]['Time to Live'].where(dict[key]['in'] == 1).min()
TTLPercent = minTTLIn/minTTLOut

diffDataFrame = abs(dict[key]['Delta time'].diff())
periodicity = diffDataFrame.mean()
stdPeriodicity= diffDataFrame.std()
```

Figure 14: Code Snippet 4

3.1.3 Training Model

For our machine learning training module, we used IBM Watson machine learning service to train and deploy our model. IBM Watson is a powerful studio that guarantees quick training, accelerated AI and machine learning deployment and choosing the best classification algorithms based on results of their performance metrics as well as doing all hyperparameter tuning automatically. The classification algorithm we chose for our model is SVM. SVM or Support Vector Machine is a machine learning algorithm which is used in supervised classification and regression, in our case it is a binary classification where 0 indicates normal and 1 indicates malicious which means ransomware in our case.

In Support Vector Machine Algorithm, we put each data as a point in n-dimension plan then we find the best hyper-plane that differentiates between two classes and one of the most important feature of support vector machine algorithm is the ability to ignore outliers and tries to find the hyper-plane which has the best margin so Support Vector Machine has really high robustness against outliers so it doesn't over fit the model.

We chose SVM because of:

1. Support Vector Machine works well in binary classification problem.
2. Support Vector Machine is effective in higher dimensional space and plans and in our case we have a high dimensional space (29 Features).
3. Support Vector Machine relatively uses memory efficiently.
4. It results in high accuracy, precision and recall compared to the other algorithms that IBM Watson traversed through in our case.

Training and Testing Results:

We gathered some statistics and metrics to demonstrate the results after the training and testing.

1) Feature Importance:

Here we show the contribution of each feature in our model and how important each feature contribution is. This is done by the IBM Watson machine learning service by performing some correlation tests.

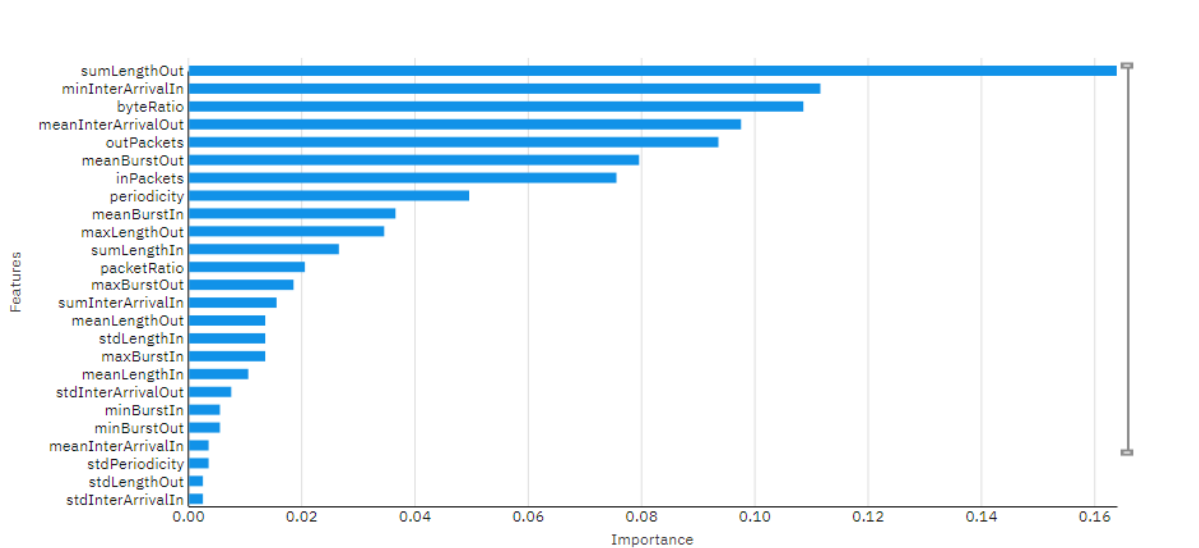


Figure 15: Feature Importance

2) Model Evaluation:

Here show all evaluation measures of the model like Accuracy, Precision, Recall, F1 score, Average Precision for both cross validation data and testing data.

Model evaluation measure		
Measures	Holdout score	Cross validation score
Accuracy	0.982	0.986
Area under ROC	0.997	1.000
Precision	0.875	0.892
Recall	1.000	1.000
F1	0.933	0.943
Average precision	0.982	1.000

Figure 16: Model Evaluation

3.2 Switch Controller

We will use the Ryu SDN Framework to implement the switch controller, Ryu is a component-based software defined networking framework. Ryu provides software components with well-defined APIs.

Although there are some well-known controllers like Beacon, POX, Chery, RYU etc. we chose the RYU controller because it is python-based, provides easy prototyping and easy to program.

Ryu offers an agile Framework for SDN application development instead of all-purpose big monolithic 'controller'. Ryu provides a bunch of components useful for SDN applications which you can modify and implement your new components then combines those components to build your application.

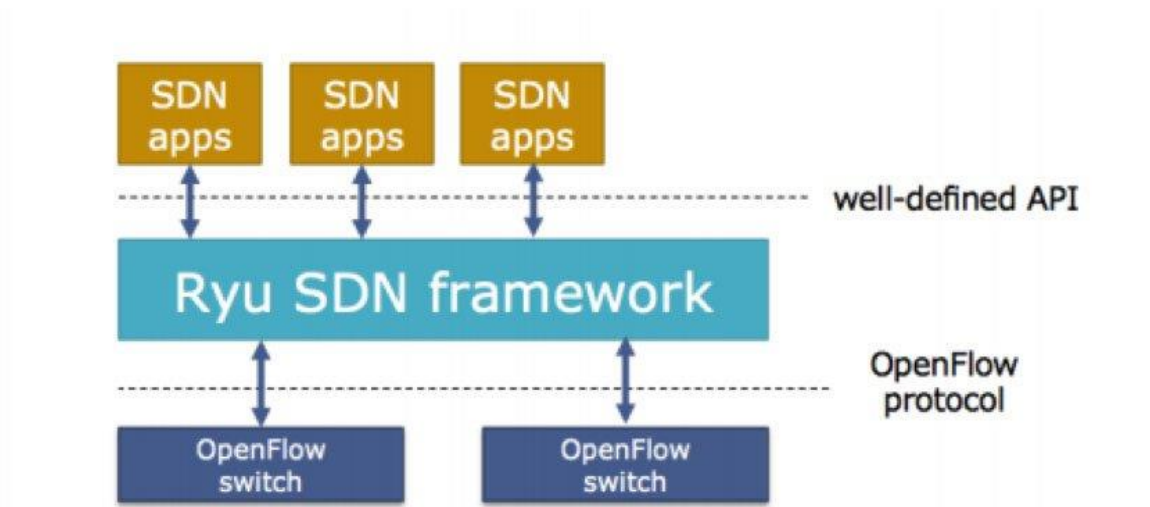


Figure 17: Ryu

The controller is the central unit of the system. The SDN controller enables the dynamic management of the network resources. The monitoring of the devices is possible because the controller obtains the globalized network view. The ryu controller is used for data forwarding within the devices.

The OpenFlow protocol is used for flow rules manipulation. The communication between the networking devices in the topology and the controller is carried out using the southbound interface. The northbound interface enables communication with the application plane. The East/Westbound interface is used in ease of communication between multiple controllers.

In this project, we customized our own RYU controller that, in addition to its basic operation which is learning the IPs of the network topology by creating the OpenFlow tables for matching and performing packet switching operations.

We implemented our own customized program that:

- Creates bi-directional flows from the real time stream of packets with flow duration of 10 seconds.
- Calls the built supervised machine learning model we created and evaluated in chapter (3).
- Flags the pre-built bi-directional flows with the label resulting from the prediction of the flow either malicious or normal in the traffic monitor class.

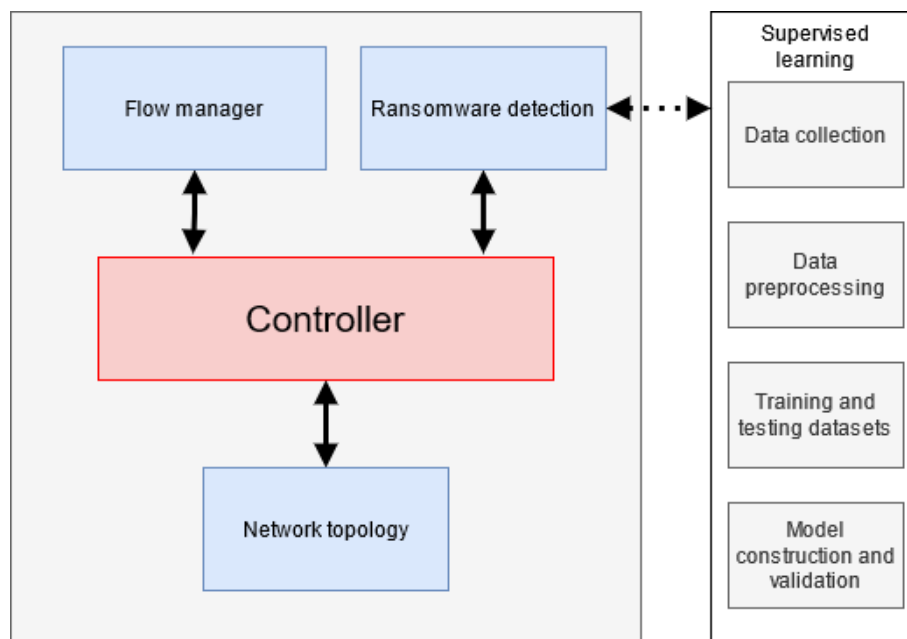


Figure 18: System Block Diagram

Full GitHub Repo: <https://github.com/seiftaher/IntelligentSecureNetworkSwitch>

4. TESTING AND RESULTS

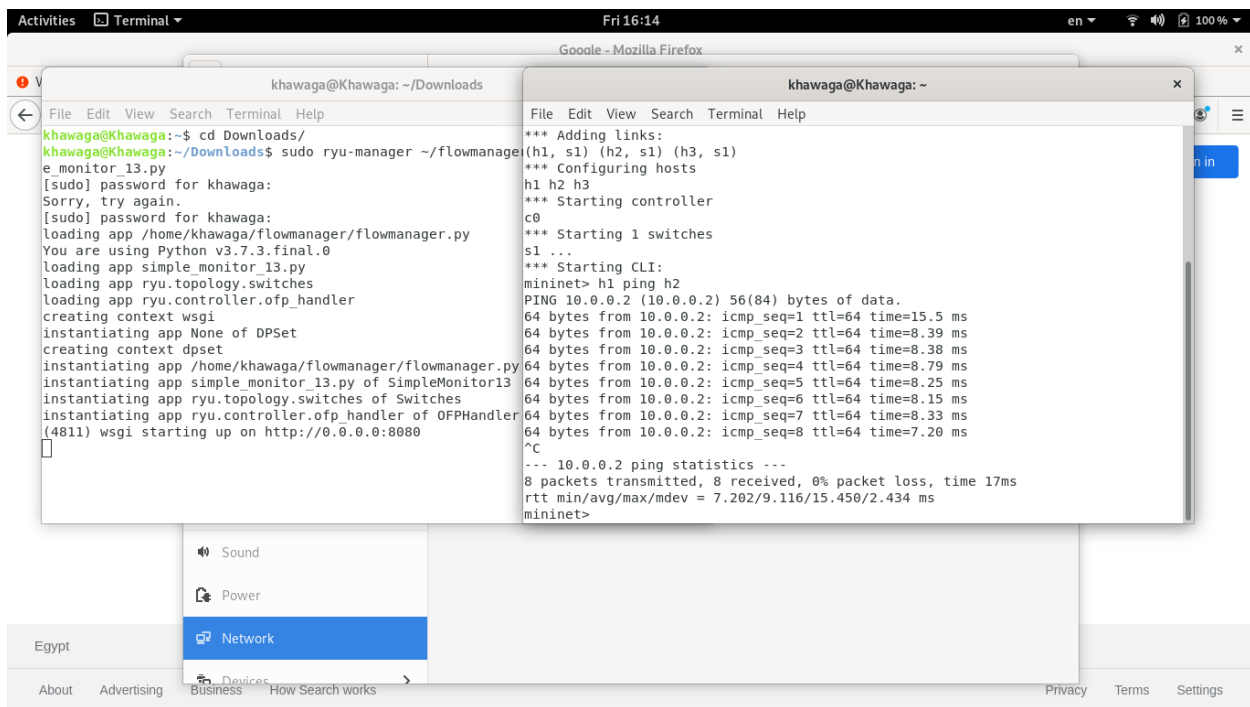
4.1 Testing on Mininet simulator

4.1.1 Testing Scenario 1

The scenario goes as follows:

- We will build an ovs switch with 3 hosts connected to it using Mininet
- We will start our controller
- If the controller is functioning probably all the hosts can reach each other (Tested using pings).

Results:



The screenshot shows a terminal window with two panes. The left pane shows the command `sudo ryu-manager ~/flowmanager.py` being executed, which starts the Ryu controller and loads various applications. The right pane shows the Mininet CLI where links are added, hosts are configured, and a ping test is performed between host1 and host2. The ping output shows 8 successful packets with a 0% loss rate.

```
khawaga@Khawaga: ~/Downloads
khawaga@Khawaga:~/Downloads$ sudo ryu-manager ~/flowmanager.py
[sudo] password for khawaga:
Sorry, try again.
[sudo] password for khawaga:
loading app /home/khawaga/flowmanager/flowmanager.py
You are using Python v3.7.3.final.0
loading app simple_monitor_13.py
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
creating context wsgi
instantiating app None of DPSet
creating context dpset
instantiating app /home/khawaga/flowmanager/flowmanager.py
instantiating app simple_monitor_13.py of SimpleMonitor13
instantiating app ryu.topology.switches of Switches
instantiating app ryu.controller.ofp_handler of OFPHandler
(4811) wsgi starting up on http://0.0.0.0:8080

khawaga@Khawaga: ~
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=15.5 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=8.39 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=8.38 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=8.79 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=8.25 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=8.15 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=8.33 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=7.20 ms
--- 10.0.0.2 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 17ms
rtt min/avg/max/mdev = 7.202/9.116/15.450/2.434 ms
mininet>
```

Figure 19: Ping Output

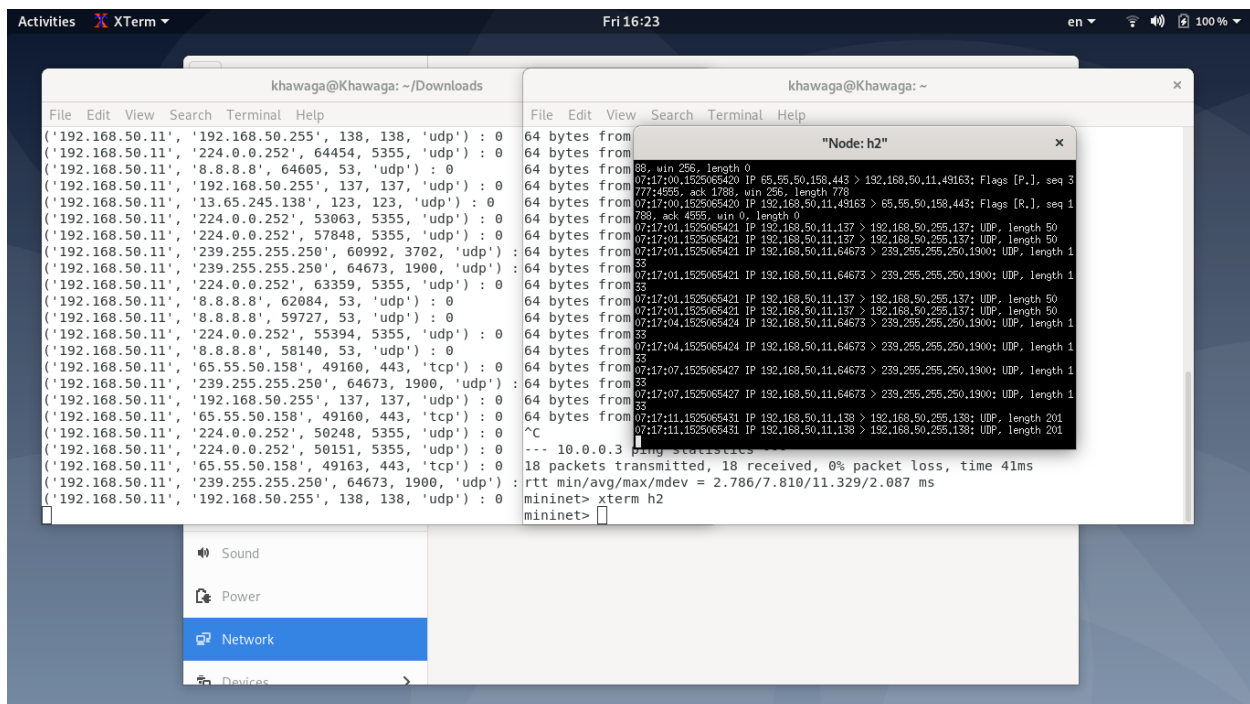
As figure 17 shows the internet ping was successful.

4.1.2 Testing Scenario 2

The scenario goes as follows:

- We will build an ovs switch with 3 hosts connected to it using Mininet
- We will start our controller.
- On one of the hosts, we will start dumping benign traffic from the dataset (Note: this traffic is different than that used for training).
- All classification for flows should return “0” flag

Results:



The screenshot shows a Mininet terminal window with the following output:

```
khawaga@Khawaga: ~/Downloads
File Edit View Search Terminal Help
('192.168.50.11', '192.168.50.255', 138, 138, 'udp') : 0
('192.168.50.11', '224.0.0.252', 64454, 5355, 'udp') : 0
('192.168.50.11', '8.8.8.8', 64605, 53, 'udp') : 0
('192.168.50.11', '192.168.50.255', 137, 137, 'udp') : 0
('192.168.50.11', '13.65.245.138', 123, 123, 'udp') : 0
('192.168.50.11', '224.0.0.252', 53063, 5355, 'udp') : 0
('192.168.50.11', '224.0.0.252', 57848, 5355, 'udp') : 0
('192.168.50.11', '239.255.255.250', 60992, 3702, 'udp') : 0
('192.168.50.11', '239.255.255.250', 64673, 1900, 'udp') : 0
('192.168.50.11', '224.0.0.252', 63359, 5355, 'udp') : 0
('192.168.50.11', '8.8.8.8', 62084, 53, 'udp') : 0
('192.168.50.11', '8.8.8.8', 59727, 53, 'udp') : 0
('192.168.50.11', '224.0.0.252', 55394, 5355, 'udp') : 0
('192.168.50.11', '8.8.8.8', 58140, 53, 'udp') : 0
('192.168.50.11', '65.55.50.158', 49160, 443, 'tcp') : 0
('192.168.50.11', '239.255.255.250', 64673, 1900, 'udp') : 0
('192.168.50.11', '192.168.50.255', 137, 137, 'udp') : 0
('192.168.50.11', '65.55.50.158', 49160, 443, 'tcp') : 0
('192.168.50.11', '224.0.0.252', 50248, 5355, 'udp') : 0
('192.168.50.11', '224.0.0.252', 50151, 5355, 'udp') : 0
('192.168.50.11', '65.55.50.158', 49163, 443, 'tcp') : 0
('192.168.50.11', '239.255.255.250', 64673, 1900, 'udp') : 0
('192.168.50.11', '192.168.50.255', 138, 138, 'udp') : 0
mininet>
```

A packet capture window titled "Node: h2" is overlaid on the terminal, showing network traffic details. The statistics at the bottom of the packet capture window are:

```
--- 10.0.0.3 Ping Statistics ---
18 packets transmitted, 18 received, 0% packet loss, time 41ms
rtt min/avg/max/mdev = 2.786/7.810/11.329/2.087 ms
mininet> xterm h2
mininet>
```

Figure 20: Benign Output

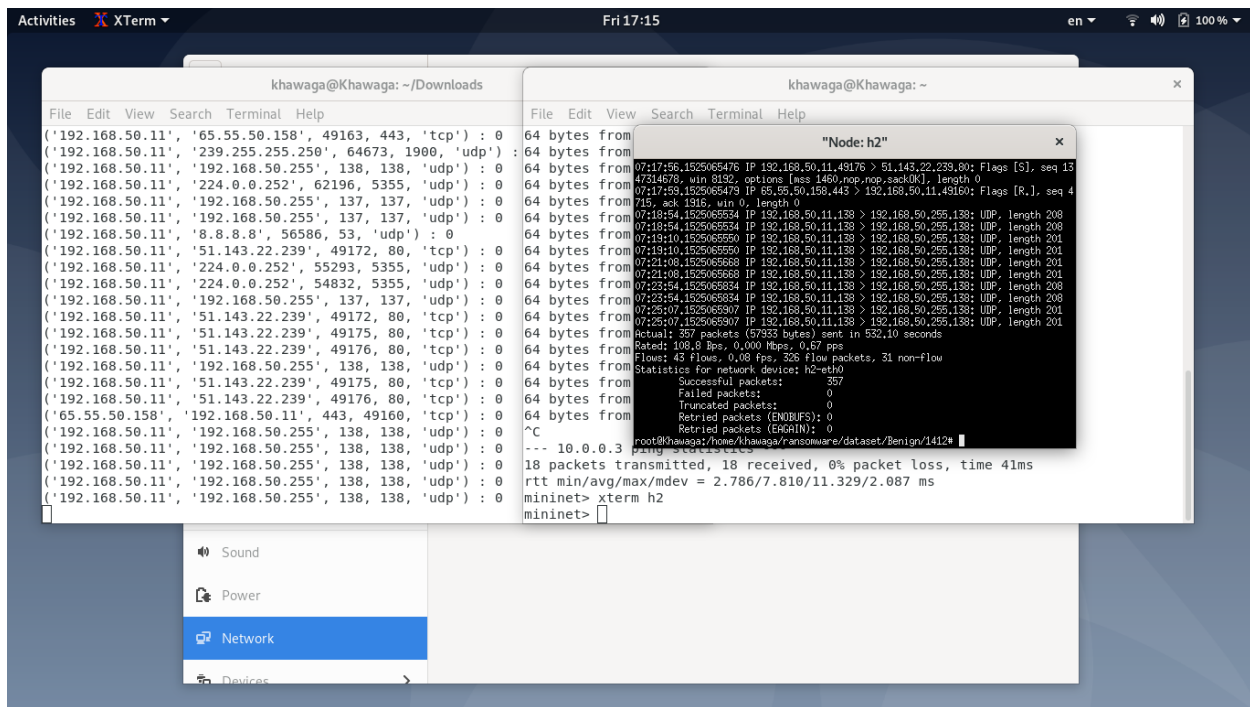


Figure 21: Full Benign Dump

As figure 19 shows all traffic is branded safe.

4.1.3 Testing Scenario 3

The scenario goes as follows:

- We will build an ovs switch with 3 hosts connected to it using Mininet
- We will start our controller.
- On one of the hosts, we will start dumping ransomware traffic from the dataset (Note: this traffic is different than that used for training).
- Some flows should return “1” flag for having ransomware

Results:

The screenshot shows a Visual Studio Code editor with a Python script named `Transformation.py` open. The script contains a function `BuildFinalDataSet` that processes network traffic data. The terminal window displays the output of the script, showing a list of network traffic records. Each record is a tuple containing source IP, destination IP, source port, destination port, protocol, and a flag. The flag is 0 for most records, indicating no ransomware, and 1 for some records, indicating the presence of ransomware. The terminal output is as follows:

```
khawaga@Khawaga: ~/Downloads
khawaga@K... khawaga@K... khawaga@K... khawaga@K...
('192.168.50.11', '94.23.175.255', 53064, 6893, 'udp') : 0
('192.168.50.11', '192.168.50.255', 138, 138, 'udp') : 0
('192.168.50.11', '192.168.50.255', 137, 137, 'udp') : 0
('192.168.50.11', '192.168.50.255', 137, 137, 'udp') : 0
('192.168.50.11', '192.168.50.255', 138, 138, 'udp') : 0
('192.168.50.11', '192.168.50.255', 138, 138, 'udp') : 0
('192.168.50.11', '192.168.50.255', 138, 138, 'udp') : 0
('192.168.50.11', '8.8.8.8', 60376, 53, 'udp') : 0
('192.168.50.11', '224.0.0.252', 63256, 5355, 'udp') : 0
('192.168.50.11', '192.168.50.255', 137, 137, 'udp') : 0
('192.168.50.11', '8.8.8.8', 63359, 53, 'udp') : 0
('192.168.50.11', '8.253.231.254', 49522, 80, 'tcp') : 0
('192.168.50.11', '224.0.0.252', 59727, 5355, 'udp') : 0
('192.168.50.11', '8.8.8.8', 49455, 53, 'udp') : 0
('192.168.50.11', '65.55.50.190', 49523, 443, 'tcp') : 0
('192.168.50.11', '8.8.8.8', 55394, 53, 'udp') : 0
('192.168.50.11', '192.168.50.11', 443, 49523, 'tcp') : 0
('192.168.50.11', '8.253.231.254', 49522, 80, 'tcp') : 1
('8.253.231.254', '192.168.50.11', 80, 49522, 'tcp') : 1
('65.55.50.190', '192.168.50.11', 443, 49523, 'tcp') : 0
('192.168.50.11', '8.253.231.254', 49522, 80, 'tcp') : 0
('192.168.50.11', '192.168.50.255', 138, 138, 'udp') : 0
('192.168.50.11', '239.255.255.250', 64673, 1900, 'udp') : 0
```

Figure 22: Ransomware Dump

```
khawaga@Khawaga: ~/Downloads
File Edit View Search Terminal Tabs Help

khawaga@K... x khawaga@K... x khawaga@K... x khawaga@K... x + ▾

('192.168.50.11', '94.23.175.255', 53064, 6893, 'udp') : 0
('192.168.50.11', '192.168.50.255', 138, 138, 'udp') : 0
('192.168.50.11', '192.168.50.255', 137, 137, 'udp') : 0
('192.168.50.11', '192.168.50.255', 137, 137, 'udp') : 0
('192.168.50.11', '192.168.50.255', 138, 138, 'udp') : 0
('192.168.50.11', '192.168.50.255', 138, 138, 'udp') : 0
('192.168.50.11', '192.168.50.255', 138, 138, 'udp') : 0
('192.168.50.11', '8.8.8.8', 60376, 53, 'udp') : 0
('192.168.50.11', '224.0.0.252', 63256, 5355, 'udp') : 0
('192.168.50.11', '192.168.50.255', 137, 137, 'udp') : 0
('192.168.50.11', '8.8.8.8', 63359, 53, 'udp') : 0
('192.168.50.11', '8.253.231.254', 49522, 80, 'tcp') : 0
('192.168.50.11', '224.0.0.252', 59727, 5355, 'udp') : 0
('192.168.50.11', '8.8.8.8', 49455, 53, 'udp') : 0
('192.168.50.11', '65.55.50.190', 49523, 443, 'tcp') : 0
('192.168.50.11', '8.8.8.8', 55394, 53, 'udp') : 0
('65.55.50.190', '192.168.50.11', 443, 49523, 'tcp') : 0
('192.168.50.11', '8.253.231.254', 49522, 80, 'tcp') : 1
('8.253.231.254', '192.168.50.11', 80, 49522, 'tcp') : 1
('65.55.50.190', '192.168.50.11', 443, 49523, 'tcp') : 0
('192.168.50.11', '8.253.231.254', 49522, 80, 'tcp') : 0
('192.168.50.11', '192.168.50.255', 138, 138, 'udp') : 0
('192.168.50.11', '239.255.255.250', 64673, 1900, 'udp') : 0
```

Figure 23: Ransomware Classification

As figure 21 shows the communication between the host ip (192.168.50.11) and the ip (8.253.231.254) is considered to have ransomware within it.

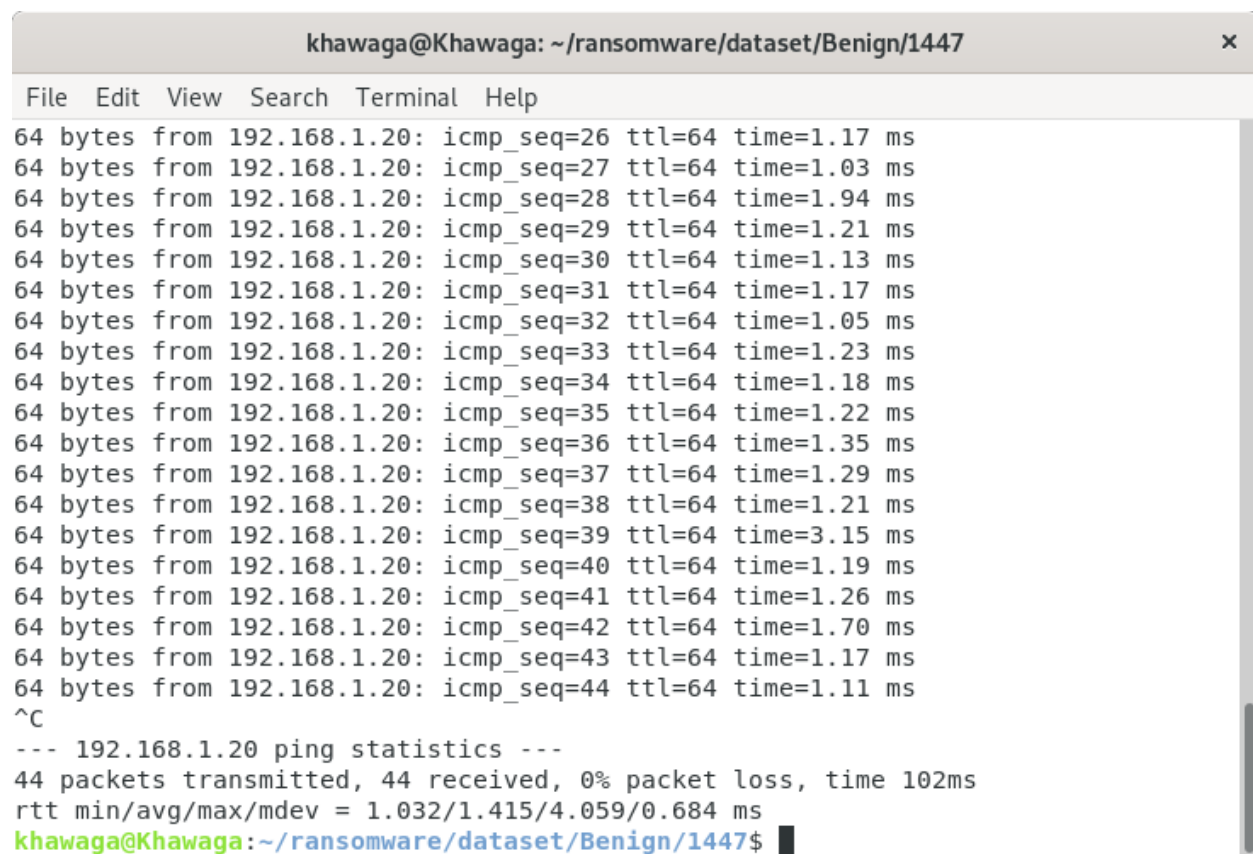
4.2 Testing on TL-mr4320 switch

4.2.1 Testing Scenario 1

The scenario goes as follows:

- We will build an ovs switch on the switch
- We will start our controller
- If the controller is functioning properly all the hosts can reach each other and the internet.

Results:

A terminal window titled 'khawaga@Khawaga: ~/ransomware/dataset/Benign/1447' with a close button 'x' in the top right corner. The terminal has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The output shows a series of 44 ping requests to 192.168.1.20, each receiving 64 bytes with a TTL of 64 and varying response times between 1.03ms and 3.15ms. After the pings, the user presses '^C' to stop the command. The terminal then displays the ping statistics: '--- 192.168.1.20 ping statistics ---', '44 packets transmitted, 44 received, 0% packet loss, time 102ms', and 'rtt min/avg/max/mdev = 1.032/1.415/4.059/0.684 ms'. The prompt 'khawaga@Khawaga:~/ransomware/dataset/Benign/1447\$' is shown at the bottom.

```
khawaga@Khawaga: ~/ransomware/dataset/Benign/1447
File Edit View Search Terminal Help
64 bytes from 192.168.1.20: icmp_seq=26 ttl=64 time=1.17 ms
64 bytes from 192.168.1.20: icmp_seq=27 ttl=64 time=1.03 ms
64 bytes from 192.168.1.20: icmp_seq=28 ttl=64 time=1.94 ms
64 bytes from 192.168.1.20: icmp_seq=29 ttl=64 time=1.21 ms
64 bytes from 192.168.1.20: icmp_seq=30 ttl=64 time=1.13 ms
64 bytes from 192.168.1.20: icmp_seq=31 ttl=64 time=1.17 ms
64 bytes from 192.168.1.20: icmp_seq=32 ttl=64 time=1.05 ms
64 bytes from 192.168.1.20: icmp_seq=33 ttl=64 time=1.23 ms
64 bytes from 192.168.1.20: icmp_seq=34 ttl=64 time=1.18 ms
64 bytes from 192.168.1.20: icmp_seq=35 ttl=64 time=1.22 ms
64 bytes from 192.168.1.20: icmp_seq=36 ttl=64 time=1.35 ms
64 bytes from 192.168.1.20: icmp_seq=37 ttl=64 time=1.29 ms
64 bytes from 192.168.1.20: icmp_seq=38 ttl=64 time=1.21 ms
64 bytes from 192.168.1.20: icmp_seq=39 ttl=64 time=3.15 ms
64 bytes from 192.168.1.20: icmp_seq=40 ttl=64 time=1.19 ms
64 bytes from 192.168.1.20: icmp_seq=41 ttl=64 time=1.26 ms
64 bytes from 192.168.1.20: icmp_seq=42 ttl=64 time=1.70 ms
64 bytes from 192.168.1.20: icmp_seq=43 ttl=64 time=1.17 ms
64 bytes from 192.168.1.20: icmp_seq=44 ttl=64 time=1.11 ms
^C
--- 192.168.1.20 ping statistics ---
44 packets transmitted, 44 received, 0% packet loss, time 102ms
rtt min/avg/max/mdev = 1.032/1.415/4.059/0.684 ms
khawaga@Khawaga:~/ransomware/dataset/Benign/1447$
```

Figure 24: Internet Ping

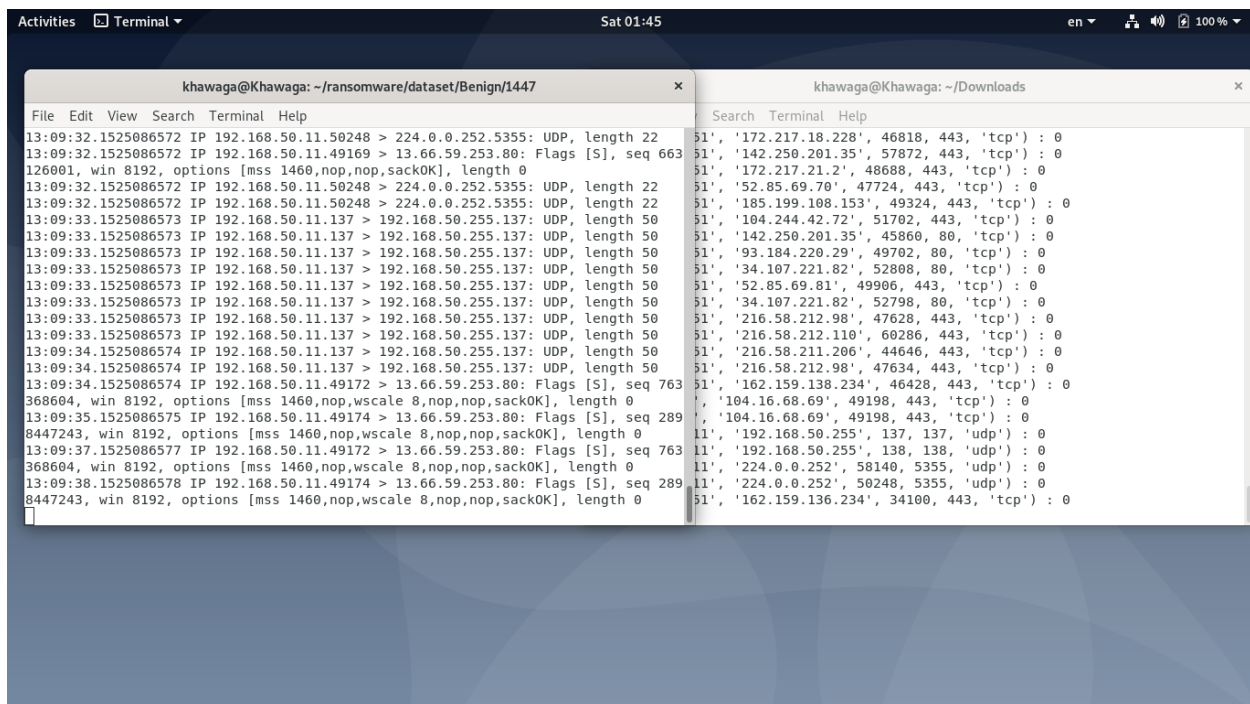
As figure 22 shows the internet ping was successful.

4.2.2 Testing Scenario 2

The scenario goes as follows:

- We will build an ovs switch on the switch
- We will start our controller.
- On one of the hosts, we will start dumping benign traffic from the dataset (Note: this traffic is different than that used for training).
- All classification for flows should return “0” flag

Results:



```
khawaga@Khawaga: ~/ransomware/dataset/Benign/1447
File Edit View Search Terminal Help
13:09:32.1525086572 IP 192.168.50.11.50248 > 224.0.0.252.5355: UDP, length 22
13:09:32.1525086572 IP 192.168.50.11.49169 > 13.66.59.253.80: Flags [S], seq 663
126001, win 8192, options [mss 1460,nop,nop,sackOK], length 0
13:09:32.1525086572 IP 192.168.50.11.50248 > 224.0.0.252.5355: UDP, length 22
13:09:32.1525086572 IP 192.168.50.11.50248 > 224.0.0.252.5355: UDP, length 22
13:09:33.1525086573 IP 192.168.50.11.137 > 192.168.50.255.137: UDP, length 50
13:09:33.1525086573 IP 192.168.50.11.137 > 192.168.50.255.137: UDP, length 50
13:09:33.1525086573 IP 192.168.50.11.137 > 192.168.50.255.137: UDP, length 50
13:09:33.1525086573 IP 192.168.50.11.137 > 192.168.50.255.137: UDP, length 50
13:09:33.1525086573 IP 192.168.50.11.137 > 192.168.50.255.137: UDP, length 50
13:09:33.1525086573 IP 192.168.50.11.137 > 192.168.50.255.137: UDP, length 50
13:09:33.1525086573 IP 192.168.50.11.137 > 192.168.50.255.137: UDP, length 50
13:09:33.1525086573 IP 192.168.50.11.137 > 192.168.50.255.137: UDP, length 50
13:09:33.1525086573 IP 192.168.50.11.137 > 192.168.50.255.137: UDP, length 50
13:09:33.1525086573 IP 192.168.50.11.137 > 192.168.50.255.137: UDP, length 50
13:09:34.1525086574 IP 192.168.50.11.137 > 192.168.50.255.137: UDP, length 50
13:09:34.1525086574 IP 192.168.50.11.137 > 192.168.50.255.137: UDP, length 50
13:09:34.1525086574 IP 192.168.50.11.49172 > 13.66.59.253.80: Flags [S], seq 763
368604, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
13:09:35.1525086575 IP 192.168.50.11.49174 > 13.66.59.253.80: Flags [S], seq 289
8447243, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
13:09:37.1525086577 IP 192.168.50.11.49172 > 13.66.59.253.80: Flags [S], seq 763
368604, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
13:09:38.1525086578 IP 192.168.50.11.49174 > 13.66.59.253.80: Flags [S], seq 289
8447243, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sackOK], length 0
11', '172.217.18.228', 46818, 443, 'tcp') : 0
11', '142.250.201.35', 57872, 443, 'tcp') : 0
11', '172.217.21.2', 48688, 443, 'tcp') : 0
11', '52.85.69.70', 47724, 443, 'tcp') : 0
11', '185.199.108.153', 49324, 443, 'tcp') : 0
11', '104.244.42.72', 51702, 443, 'tcp') : 0
11', '142.250.201.35', 45860, 80, 'tcp') : 0
11', '93.184.220.29', 49702, 80, 'tcp') : 0
11', '34.107.221.82', 52808, 80, 'tcp') : 0
11', '52.85.69.81', 49906, 443, 'tcp') : 0
11', '34.107.221.82', 52798, 80, 'tcp') : 0
11', '216.58.212.98', 47628, 443, 'tcp') : 0
11', '216.58.212.110', 60286, 443, 'tcp') : 0
11', '216.58.211.206', 44646, 443, 'tcp') : 0
11', '216.58.212.98', 47634, 443, 'tcp') : 0
11', '162.159.138.234', 46428, 443, 'tcp') : 0
11', '104.16.68.69', 49198, 443, 'tcp') : 0
11', '104.16.68.69', 49198, 443, 'tcp') : 0
11', '192.168.50.255', 137, 137, 'udp') : 0
11', '192.168.50.255', 138, 138, 'udp') : 0
11', '224.0.0.252', 58140, 5355, 'udp') : 0
11', '224.0.0.252', 50248, 5355, 'udp') : 0
11', '162.159.136.234', 34100, 443, 'tcp') : 0
```

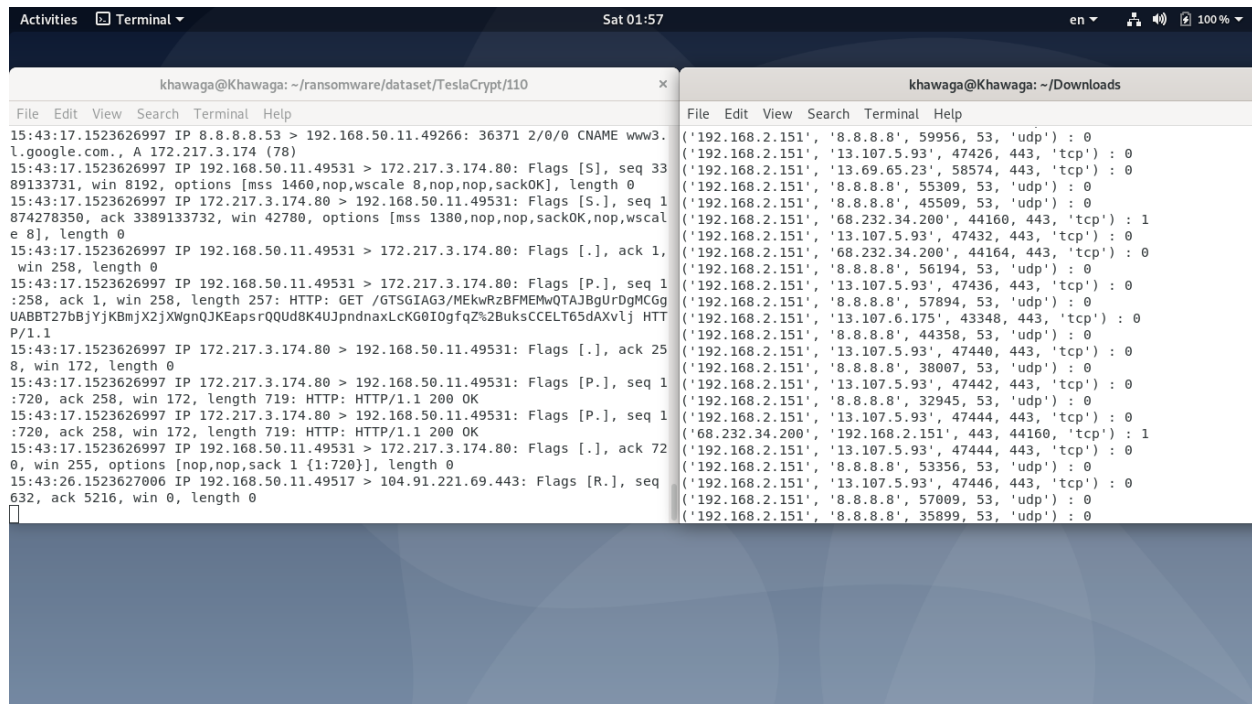
Figure 25: Benign traffic result

As figure 23 shows all traffic is branded safe.

4.2.3 Testing Scenario 3

The scenario goes as follows:

- We will build an ovs switch on the switch
- We will start our controller.
- On one of the hosts, we will start dumping ransomware traffic from the dataset (Note: this traffic is different than that used for training).
- Some flows should return “1” flag for having ransomware



```
khawaga@Khawaga: ~/ransomware/dataset/TeslaCrypt/110
15:43:17.1523626997 IP 8.8.8.8.53 > 192.168.50.11.49266: 36371 2/0/0 CNAME www3.
l.google.com., A 172.217.3.174 (78)
15:43:17.1523626997 IP 192.168.50.11.49531 > 172.217.3.174.80: Flags [S], seq 33
89133731, win 8192, options [mss 1460,nop,wscale 8,nop,nop,sack0K], length 0
15:43:17.1523626997 IP 172.217.3.174.80 > 192.168.50.11.49531: Flags [S.], seq 1
874278350, ack 3389133732, win 42780, options [mss 1380,nop,nop,sack0K,nop,wscal
e 8], length 0
15:43:17.1523626997 IP 192.168.50.11.49531 > 172.217.3.174.80: Flags [.], ack 1,
win 258, length 0
15:43:17.1523626997 IP 192.168.50.11.49531 > 172.217.3.174.80: Flags [P.], seq 1
:258, ack 1, win 258, length 257: HTTP: GET /GTS6IAG3/MEkwrZBFMEMwQTAJBgUrDgMCGg
UABBT27bBjYjKBmjX2jXWgnQJKEapsrQQud8K4UJpndnaxLCKG0IogfQZ%2BuksCCELT65dAXvLj HT
P/1.1
15:43:17.1523626997 IP 172.217.3.174.80 > 192.168.50.11.49531: Flags [.], ack 25
8, win 172, length 0
15:43:17.1523626997 IP 172.217.3.174.80 > 192.168.50.11.49531: Flags [P.], seq 1
:720, ack 258, win 172, length 719: HTTP: HTTP/1.1 200 OK
15:43:17.1523626997 IP 172.217.3.174.80 > 192.168.50.11.49531: Flags [P.], seq 1
:720, ack 258, win 172, length 719: HTTP: HTTP/1.1 200 OK
15:43:17.1523626997 IP 192.168.50.11.49531 > 172.217.3.174.80: Flags [.], ack 72
0, win 255, options [nop,nop,sack 1 {1:720}], length 0
15:43:26.1523627006 IP 192.168.50.11.49517 > 104.91.221.69.443: Flags [R.], seq
632, ack 5216, win 0, length 0

khawaga@Khawaga: ~/Downloads
('192.168.2.151', '8.8.8.8', 59956, 53, 'udp') : 0
('192.168.2.151', '13.107.5.93', 47426, 443, 'tcp') : 0
('192.168.2.151', '13.69.65.23', 58574, 443, 'tcp') : 0
('192.168.2.151', '8.8.8.8', 55309, 53, 'udp') : 0
('192.168.2.151', '8.8.8.8', 45509, 53, 'udp') : 0
('192.168.2.151', '68.232.34.200', 44160, 443, 'tcp') : 1
('192.168.2.151', '13.107.5.93', 47432, 443, 'tcp') : 0
('192.168.2.151', '68.232.34.200', 44164, 443, 'tcp') : 0
('192.168.2.151', '8.8.8.8', 56194, 53, 'udp') : 0
('192.168.2.151', '13.107.5.93', 47436, 443, 'tcp') : 0
('192.168.2.151', '8.8.8.8', 57894, 53, 'udp') : 0
('192.168.2.151', '13.107.6.175', 43348, 443, 'tcp') : 0
('192.168.2.151', '8.8.8.8', 44358, 53, 'udp') : 0
('192.168.2.151', '13.107.5.93', 47440, 443, 'tcp') : 0
('192.168.2.151', '8.8.8.8', 32945, 53, 'udp') : 0
('192.168.2.151', '13.107.5.93', 47444, 443, 'tcp') : 0
('192.168.2.151', '8.8.8.8', 38007, 53, 'udp') : 0
('192.168.2.151', '13.107.5.93', 47442, 443, 'tcp') : 0
('192.168.2.151', '8.8.8.8', 32945, 53, 'udp') : 0
('192.168.2.151', '13.107.5.93', 47444, 443, 'tcp') : 0
('192.168.2.151', '68.232.34.200', 192.168.2.151', 443, 44160, 'tcp') : 1
('192.168.2.151', '13.107.5.93', 47444, 443, 'tcp') : 0
('192.168.2.151', '8.8.8.8', 53356, 53, 'udp') : 0
('192.168.2.151', '13.107.5.93', 47446, 443, 'tcp') : 0
('192.168.2.151', '8.8.8.8', 57009, 53, 'udp') : 0
('192.168.2.151', '8.8.8.8', 35899, 53, 'udp') : 0
```

Figure 26: Ransomware traffic result

As figure 24 shows the communication between the host ip (192.168.2.151) and the ip (68.232.34.20) is considered to have ransomware within it.

5. CONCLUSION

In this thesis, we have implemented and integrated many applications in both network and computer science level with many freshly and newly concepts like Machine Learning and Software Defined Networks.

First in Software Defined Networks we implemented the simple network switch using the RYU controller and we wanted to add another high-level features to controller to show the importance of Software Defined Network and how it can be used and tamed to better the controlling of networks and increase networks' security, so the idea was to integrate a machine learning model to the controller to add to it a substantial importance and security in network traffic and monitoring.

The Machine Learning model had to be trained to something that affects all our lives and causes a huge amount of damage in both material and moral levels, so we chose the ransomware pandemic.

Ransomware causes a huge amount of losses every year, how many people and companies have agreed to pay the blackmailers just to get their valuable data back and made a material lost by losing their money to get the data, how many people couldn't pay the blackmailers to as they don't have the money to do so and their data were lost forever causing for a moral damage?

We tried to do something to help those people and prevent this type of extortion so we had to train a machine learning model so it can detect if someone is about to be blackmailed.

The first step of getting our machine learning goal is to get enough data so we can extract features from it to pass it to the machine learning model, and we searched for the suitable data and got it.

The second step was to determine the suitable features from our data so it can train and predict well, and after a long searching of ransomware traffic behavior we could get a lot of suitable features that we can use it to train and test our model with, and we built bi-directional flow table to achieve this step.

The third step was to choose a suitable model and we chose Support Vector Machine algorithm for our model as Support Vector Machine works well in binary classification problem. Support Vector Machine is effective in higher dimensional space and plans and in our case we have a high dimensional space (29 Features). Support Vector Machine relatively uses memory efficiently.

The fourth step was training and testing our data and after trials of success and failures we could get the suitable model for our case.

Now we are done with our machine learning model now gets to another issue, that issue is integrating our model in our RYU controller and taking the packet info we used in training to build the bi-directional flow table so we can pass it to the model to make the prediction.

As RYU is written in high level language (Python) we could integrate between the machine learning model and the controller and make them work together harmoniously.

Now we have done all of that remains another important and huge step, this step is to make it really beneficial and make it applicable to be a true product in the future and showing the importance of software defined network and its many application that it can be tamed and used.

We had to make at first simulation in our case to verify and test that everything works fine and like predicted so we made virtual hosts connected to each other and communicating with each other to test basic functionalities is working fine, then we had to simulate that one of hosts is trying to ransomware the other host, so we sent some of malicious packets from one host to another to make sure that our model and network can predict if there is something abnormal is happening, and it succeeded to do so.

Now like I said earlier we had not to stop in only the simulation, but we wanted also to show that it is applicable and can be turned into product in the future, so we had to run a physical and real scenario.

We got an access point to make it like a switch connecting different hosts and controller so we can run a physical and true scenario, but this step is hard as we have to get a switch, they can use open flow protocol and can communicate with the controller.

We got the idea of getting a special model of access point that we can modify in its operating system so we can manipulate it, after getting the access point we installed on it OpenWrt so we can install on it the openvswitch so it can communicate properly with the controller.

After setting up the environment and connecting hosts to ports and connecting the controller to the switch we tested the basic functionality of our network and it worked fine, after it we simulated an attack from one of the physical computer hosts to the other host so we can make sure that results we got from the virtual simulation is the same to the physical simulation and indeed it could predict that there is a ransom ware attack is happening between the hosts.

After doing and finishing all the above steps we could show and verify the importance of Software Defined Networks and how it can be used to better our daily lives and not only that we have proven that this model and system can be used in physical environment and be turned into a product that can help and save millions of lives.

5.1 Future Work

Machine learning detection model works on the concept of feature engineering. So far, we have extracted our set of features that are constructed from flow-based concept. For future enhancement, packet-based features can also be extracted to help in detecting ransomware. Combination of both kinds of features will improve identifying novel attacks as ransomware will always evolve to evade detection system.

For better results of machine learning training, we will use more data to avoid overfitting. We will choose multiple classification algorithms such as deep forest, logistic regression, KNN etc. other than the SVM used in this thesis. We can then construct a hybrid model to combine some of these algorithms and take the best result from this hybrid model based on pre-defined threshold.

Our current work is based on offline machine learning training. Online training has an advantage over offline training because it guarantees keeping up with the evolving ransomware attacks that are continuously improving to evade detections systems, so for future work, we will adopt online training on real time network traffic. Due to extra latency that online training adds in addition to the higher processing power it needs, we can do periodic offline machine learning retraining.

Our current work is based on supervised machine learning models. Deep learning neural network has been successful in upgrading supervised machine learning settings. With more data available, constructing deep neural networks can achieve better results. In real time case, if we adopt the online machine learning training path, it is kind of certain that the data used for training will be much larger than the data we used to build the offline training model we constructed in this thesis. Therefore, deep learning will be better certainly.

Since RYU controller already adopts multithreading techniques using hub. On current work, we used the two threads already created, one for the switching operations and the other for the traffic monitoring, provided that we enhanced both threads to do our customized operations. On using more sophisticated algorithms, the processing time may increase tremendously, so it will affect the traffic monitor processing quality. Hence, we could separate the machine learning operations to another thread also known as hub in RYU controller.

We focused in this thesis on flagging the flow by the aid of machine learning. We can make further improvement to the whole process by making the controller take actions based on this flag such as blocking any packet coming from this malicious IP address.

For further enhancement of packet parsing and feature extraction module, we can make a more sophisticated parser based on the binary exe of the packet instead of the packet library provided by RYU controller packages. As the default library has some drawbacks and limitations to some of the protocols as TLSv1 for instance.

Currently, we make use of the packet headers only and the statistical analysis of their flow behavior. For future enhancement, we can use the packet payload through deep packet inspection of encrypted and unencrypted payloads to enhance our features module.

APPENDIX I

In order to simulate the controller on a real router we need to follow the following steps:

1. Obtain a router that is compatible with Openwrt (OpenWrt is the framework to build an application without having to build a complete firmware around it; for users this means the ability for full customization).

You can see the list of devices through this link: https://openwrt.org/supported_devices

2. Install Openwrt on the router you can find an installation guide for each model through their website openwrt.org.

Here is the installation guide for the model we used tl-mr4320 v5:

openwrt.org/toh/tp-link/tp-link_tl-mr3420_v5

Installation

Model	Version	Current Release	Firmware OpenWrt Install	Firmware OpenWrt Upgrade	Firmware OEM Stock
TL-MR3420	v5	19.07.7	http://downloads.openwrt.org/releases/19.07.7/targets/ramips/mt76x8/openwrt-19.07.7-ramips-mt76x8-tp-link_tl-mr3420-v5-squashfs-tftp-recovery.bin	http://downloads.openwrt.org/releases/19.07.7/targets/ramips/mt76x8/openwrt-19.07.7-ramips-mt76x8-tp-link_tl-mr3420-v5-squashfs-sysupgrade.bin	https://www.tp-link.com/en/download/TL-MR3420_V5.html

The only way to flash an OpenWrt image to TL-MR3420 v5 is to use tftp recovery mode in U-Boot:

1. Configure PC with static IP 192.168.0.225/24 and tftp server.
2. Rename `...squashfs-tftp-recovery.bin` to `tp_recovery.bin` and place it in the tftp server directory.
3. Connect PC with one of LAN ports, press the reset button, power up the router and keep button pressed for around 6-7 seconds, until device starts downloading the file.
4. Router will download file from server, write it to flash and reboot.

3. Log in to the router by using ssh
4. Start Configuring the network parameters you want through the network configuration file(/etc/config/network)
5. Configure the DHCP Server through the DHCP configuration file (/etc/config/dhcp)
6. Restart the network `/etc/init.d/network restart`
7. Restart the DHCP `/etc/init.d/dnsmasq restart`

8. Add the ovs bridge and add its interface to the network configuration file (Note: Don't forget to change the IP address of the controller of the bridge to your bridge controller).
9. Finally, Run the controller and the flow manager (It is a RYU controller application that gives the user manual control over the flow tables in an OpenFlow network. The user can also monitor the OpenFlow switches and view statistics. The Flow Manager is ideal for learning OpenFlow in a lab environment, or in conjunction with other applications to tweak the behavior of network flows in a production environment.)

REFERENCES

- [1] Mittal, Sangeeta. (2018). Performance Evaluation of Openflow SDN Controllers.
- [2] Computer Networking A Top Down Approach 7th edition By Kurose & Ross
- [3] Hu F., Hao Q. and Bao K.: A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation. IEEE Communications Surveys & Tutorials 16(4), 2181-2206 (2014).
- [4] OpenFlow Switch Specification 1.3.1
- [5] Yu, Changhe, et al. "QoS-aware Traffic Classification Architecture Using Machine Learning and Deep Packet Inspection in SDNs." Procedia computer science 131 (2018): 1209-1216.
- [6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69–74, 2008
- [7] Christopher M. Bishop, "Pattern recognition and machine learning", springer, 2006.
- [8] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. "The elements of statistical learning", Springer series in statistics, 2001.
- [9] Brijesh Jethva, Issa Traoré, Asem Ghaleb, Karim Ganame, Sherif Ahmed, "Multilayer Ransomware Detection using Grouped Registry Key Operations, File Entropy and File Signature Monitoring" Journal of Computer Security, Volume 28, Number 3, 2020, pages 337-373.
- [10] Y.-L. Wan, J.-C. Chang, R.-J. Chen, and S.-J. Wang, "Feature-Selection-Based Ransomware Detection with Machine Learning of Data Analysis," in 2018 3rd International Conference on Computer and Communication Systems (ICCCS), 2018.

[11] Hilary Tuttle. 2016. Ransomware attacks pose growing threat. Risk Management 63, 4 (2016), 4.

[12] Ting-Fang Yen and Michael K Reiter. 2008. Traffic aggregation for malware detection. Lecture Notes in Computer Science 5137 (2008).

[13] Yanjun Qi. 2012. Random Forest for bioinformatics. In Ensemble machine learning. Springer, 307–323.

[14] Cortes, C., Vapnik, V.: Support-vector network. Mach. Learn. 20, 273–297 (1995)