НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

«На правах рукопису» «До захисту допущено»

УДК кафедри Завідувач кафедри

Стіренко С.Г.

(підпис) (ініціали, прізвище)

“07” 05 2020 р.

Магістерська дисертація

зі спеціальності: *121. Інженерія програмного забезпечення*

(код та назва напряму підготовки або спеціальності)

Спеціалізація: *121.Інженерія програмного забезпечення комп’ютерних систем*

на тему: Ecrypted Network Classification With Deep Learning

Виконав (-ла): студент (-ка) VI курсу, групи IO-84мн

(шифр групи)

Абдаллах Мухаммед Зайяд

(прізвище, ім’я, по батькові) (підпис)

Науковий керівник Роковий .О.П

(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант ____нормокотроль проф., д.т.н. Кулаков Юрій Олексійович

____

(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент ____

______

(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.

Студент _________________

(підпис)

Київ – 2020 року

РЕФЕРАТ

Дисертація складається з 84 сторінок, 59 Цифри та 29 джерел у довідковому списку.

Проблема: Оскільки світ стає більш безпечним, для забезпечення належної передачі даних між сторонами, що спілкуються, було використано більше протоколів шифрування. Класифікація мережі стала більше клопоту з використанням деяких прийомів, оскільки перевірка зашифрованого трафіку в деяких країнах може бути незаконною. Це заважає інженерам мережі мати можливість класифікувати трафік, щоб відрізняти зашифрований від незашифрованого трафіку.

Мета роботи: Ця стаття спрямована на проблему, спричинену попередніми методами, використовуваними в шифрованій мережевій класифікації. Деякі з них обмежені розміром даних та обчислювальною потужністю. У даній роботі використовується рішення алгоритму глибокого навчання для вирішення цієї проблеми.

Основні завдання дослідження:

1. Порівняйте попередні традиційні методи та порівняйте їх переваги та недоліки

2. Вивчити попередні супутні роботи у сучасній галузі досліджень.

3. Запропонуйте більш сучасний та ефективний метод та алгоритм для зашифрованої класифікації мережевого трафіку

Об'єкт дослідження: Простий алгоритм штучної нейронної мережі для точної та надійної класифікації мережевого трафіку, що не залежить від розміру даних та обчислювальної потужності.

Предмет дослідження: На основі даних, зібраних із приватного потоку трафіку у нашому власному інструменті моделювання мережі. За допомогою запропонованого нами методу визначаємо відмінності корисних навантажень мережевого трафіку та класифікуємо мережевий трафік. Це допомогло відокремити або класифікувати зашифровані від незашифрованого трафіку.

Методи дослідження: Експериментальний метод.
Ми провели наш експеримент із моделюванням мережі та збиранням трафіку різних незашифрованих протоколів та зашифрованих протоколів. Використовуючи мову програмування python та бібліотеку Keras, ми розробили згорнуту нейронну мережу, яка змогла прийняти корисне навантаження зібраного трафіку, навчити модель та класифікувати трафік у нашому тестовому наборі з високою точністю без вимоги високої обчислювальної потужності

Ключові слова: конволюційна нейронна мережа, дані, модель, глибокі нейронні мережі, глибоке навчання, протоколи, шифрування, Python.

ABSTRACT

This dissertation consists of 84 pages, 59 Figures and 29 sources in the reference list.

Problem: As the world becomes more security conscious, more encryption protocols have been employed in ensuring suecure data transmission between communicating parties. Network classification has become more of a hassle with the use of some techniques as inspecting encrypted traffic can pose to be illegal in some countries. This has hindered network engineers to be able to classify traffic to differentiate encrypted from unencrypted traffic.

Purpose of work: This paper aims at the problem caused by previous techniques used in encrypted network classification. Some of which are limited to data size and computational power. This paper employs the use of deep learning algorithm to solve this problem.

The main tasks of the research:

1. Compare previous traditional techniques and compare their advantages and disadvantages

2. Study previous related works in the current field of research.

3. Propose a more modern and efficient method and algorithm for encrypted network traffic classification

The object of research: Simple artificial neural network algorithm for accurate and reliable network traffic classification that is independent of data size and computational power.

The subject of research: Based on data collected from private traffic flow in our own network simulation tool. We use our proposed method to identify the differences in network traffic payloads and classify network traffic. It helped to separate or classify encrypted from unencrypted traffic.

Research methods: Experimental method.

We have carried out our experiment with network simulation and gathering traffic of different unencrypted protocols and encrypted protocols. Using python programming language and the Keras library we developed a convolutional neural network that was able to take in the payload of the traffic gathered, train the model and classify the traffic in our test set with high accuracy without the requirement of high computational power.

Keywords: Convolutional Neural Network, Data, Model, Deep Neural Networks, Deep learning, Protocols, Encryption, Python.

CONTENTS

ABBREVIATIONS

POP3 – Post Office Protocol

FTP – File Transfer Protocol

VPN – Virtual Private Network

P2PP – Peer-to-Peer Protocol

DNS – Domain Name System

SFTP – SSH File Transfer Protocol

SSH – Secure Shell Protocol

CNN – Convolutional Neural Network

ANN – Artificial Neural Network

LTSM – Long Short-Term Memory

NN – Neural Network

TCP – Transmission Control Protol

UDP – User Datagram Protocol

ICMP – Internet Control Message Protocol

AE – Auto encoders

MLP – Multi layer Perceptron.

SECTION I

1.1 INTRODUCTION

Most networks traffic are identified by features which maybe port numbers or statistics characteristics and so on. The fast development of the internet and communication devices has created bigger and more complicated network structures, adapting and developing bigger hubs, routers, switches, etc. This complexity in networks has introduced an overflow of vast amounts of traffic data and contributed to the challenges in network management and traffic optimization, including traffic measurement (e.g. traffic classification) and traffic prediction.

1.2 RELATED WORKS

Methods have been proposed on easy detection and classification of network traffic.

K.Muthamil et all[1], proposed work is to detect the malicious activities in the SDN environment with high accuracy. Initially, the flow information is collected from OVS switches at regular intervals and by using that information essential features are extracted. After that by applying hybrid machine learning technique, we construct classifier module to detect attacks in the flow. In our proposed work, we have implemented K-Means clustering, Modified K-Means clustering, C4.5 decision tree and Modified K-Means+C4.5 (MKMC4) decision tree hybrid algorithm.

The IDS module consists of flow statistics collection module, traffic classification module, feature extraction module and hybrid machine learning testing and training phase to detect the attacks. From the controller, flow statistics are collected for every second. If a flow is inactive for more than two seconds, it is considered as idle. The message type indicates the reason for arrival of packets towards the controller. It may be due to table miss or flow rule installed in the flow table directing the packets towards the controller.

Feature Name	Description
Protocol_type	Indicates nature of the protocol (TCP,ICMP,UDP,etc.)
Duration	Length of the connection (Number of seconds)
source_bytes	Number of bytes from source to destination
destination_bytes	Number of bytes from destination to source
Count	For the past two seconds, number of connections to the same host.
service_count	For the past two seconds, number of connections to the same service.

Fig 1.1 Features and Descriptions

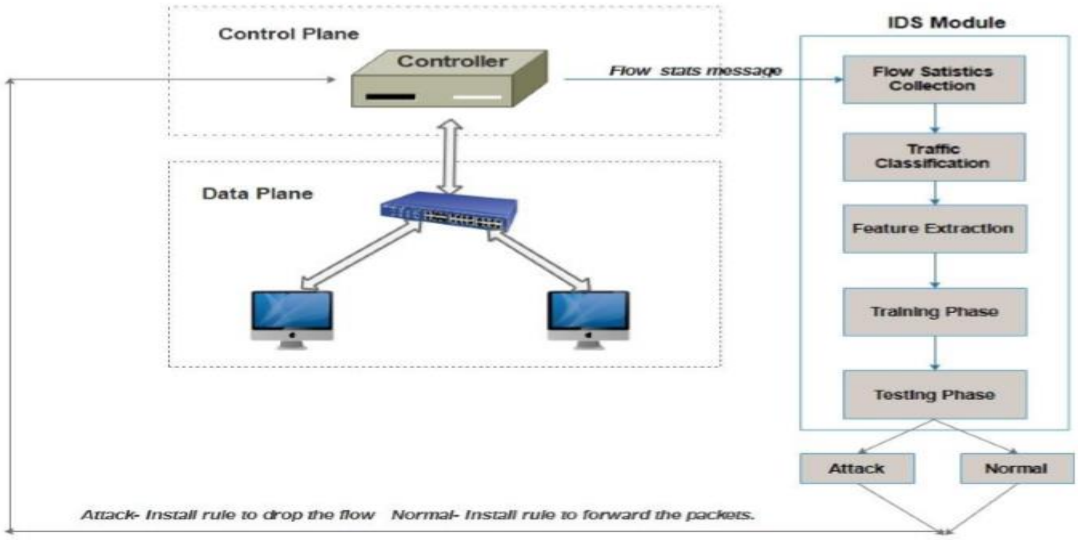


Fig 1.2 System Architecture

When a packet arrives towards the controller, feature extraction and traffic classification could happen by analysing header fields from the packet. For TCP and UDP traffic, source and destination IP, source and destination port, protocol type will have same values. Same is applicable for ICMP traffic also but with different port numbers. In addition to that, this module will eliminate the symmetric flow. If source IP address and source port number of one flow are similar to destination port number and IP address of another flow for TCP or UDP traffic respectively, then these flows are considered as symmetric flow. For ICMP symmetric flows, the two flows are request and response types. The main reason for eliminating symmetric flows is that attackers mainly spoof their IP addresses in order to restrict the responses from victims. So, this module installs the flow rules only for normal traffic and avoids the saturation in flow tables. For their proposed work, they extracted six essential features such as protocol_type, duration, sorce_bytes, destination_bytes, count, service_count.. Then the machine learning based detection module will process the packets and classify it as normal or attack packets. Once the attack is detected, the OpenFlow protocol modifies the flow table immediately to drop the particular flow. Their results were accurate.

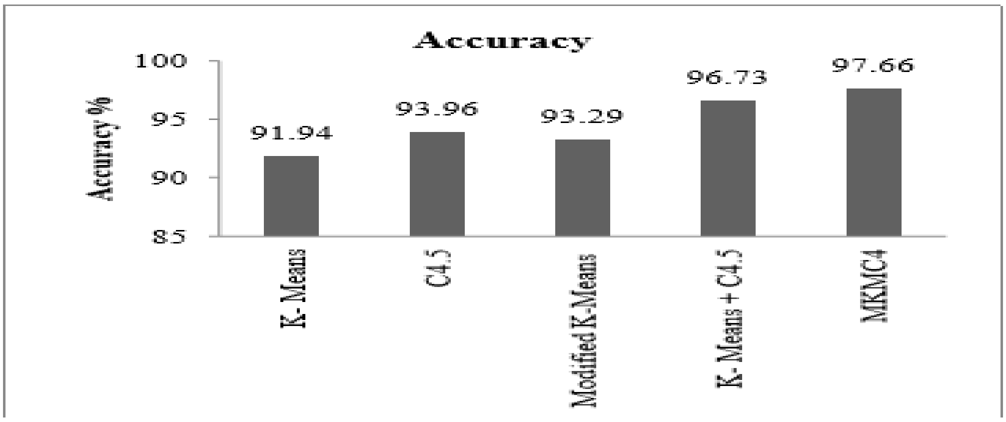


Fig 1.3 Result

Loftallahi et all[2], presented Deep Packet, a framework that automatically extracts features from computer networks traffic using deep learning algorithms to classify traffic. To the best of their knowledge, Deep Packet is the first traffic classification system using deep learning algorithms, namely SAE and 1D-CNN that can handle both application identification and traffic characterization tasks. Proposed CNN as shown below.

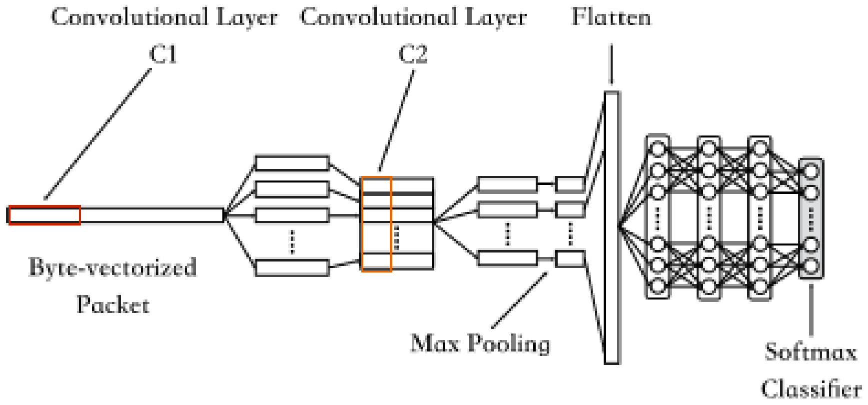


Fig 1.3 Proposed CNN Architecture

Results showed that Deep Packet outperforms all of the similar works on the "ISCX VPN-nonVPN" traffic dataset, in both application identification and traffic characterization tasks, to the date. Moreover, with state-of-the-art results achieved by Deep Packet, they envisage that Deep Packet is the first step toward a general trend of using deep learning algorithms in traffic classification and more generally network analysis tasks.

Class name	CNN		
	Rc	Pr	F_1
Tor: Google	0.00	0.00	0.00
Tor: Facebook	0.24	0.10	0.14
Tor: YouTube	0.44	0.55	0.49
Tor: Twitter	0.17	0.01	0.01
Tor: Vimeo	0.36	0.44	0.40
Wtd. average	**0.35**	**0.40**	**0.36**

Fig 1.4 Results of proposed CNN

Furthermore, Deep Packet can be modified to handle more complex tasks like multi-channel (e.g., distinguishing between different types of Skype traffic including chat, voice call, and video call) classification, accurate classification of Tor's traffic, etc. Finally, the automatic feature extraction procedure from network traffic can save the cost of employing experts to identify and extract handcrafted features from the traffic which eventually leads to more accurate traffic classification.

Naseer et all[3], analyzed the usage of deep learning algorithms, specifically CNN, AE, and Intrusion Detection models were proposed, implemented and trained using different deep neural network architectures including Convolutional Neural Networks, Autoencoders, and Recurrent Neural Networks.

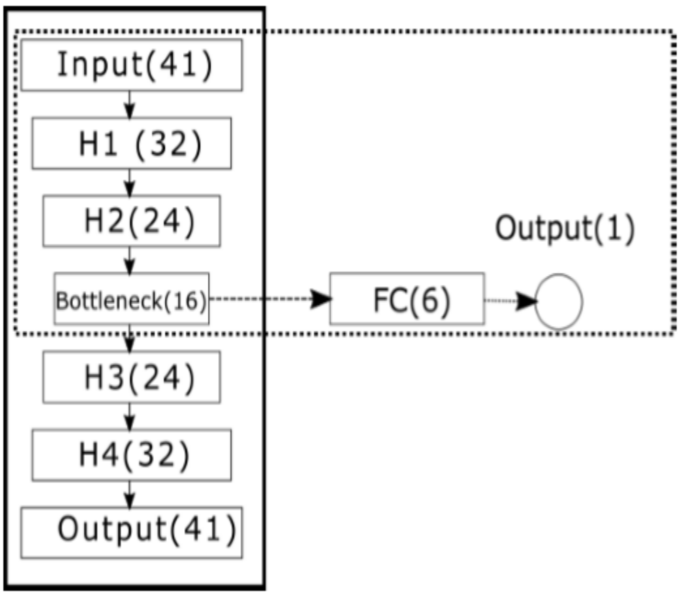


Fig 1.5 Auto-encoders Architecture

These deep models were trained on NSLKDD training dataset and evaluated on both test datasets provided by NSLKDD namely NSLKDDTest+ and NSLKDDTest21. For training and evaluation of deep models, a GPU powered test-bed using keras with theano backend was employed. To make model comparisons more credible, they implemented conventional ML IDS models with different well-known classification techniques including Extreme Learning Machine, k-NN, Decision-Tree, Random-Forest, Support Vector Machine, Naive-Bays, and QDA. Both DNN and conventional ML models were evaluated using well-known classification metrics including RoC Curve, Area under RoC, Precision-Recall Curve, mean average precision and accuracy of classification.

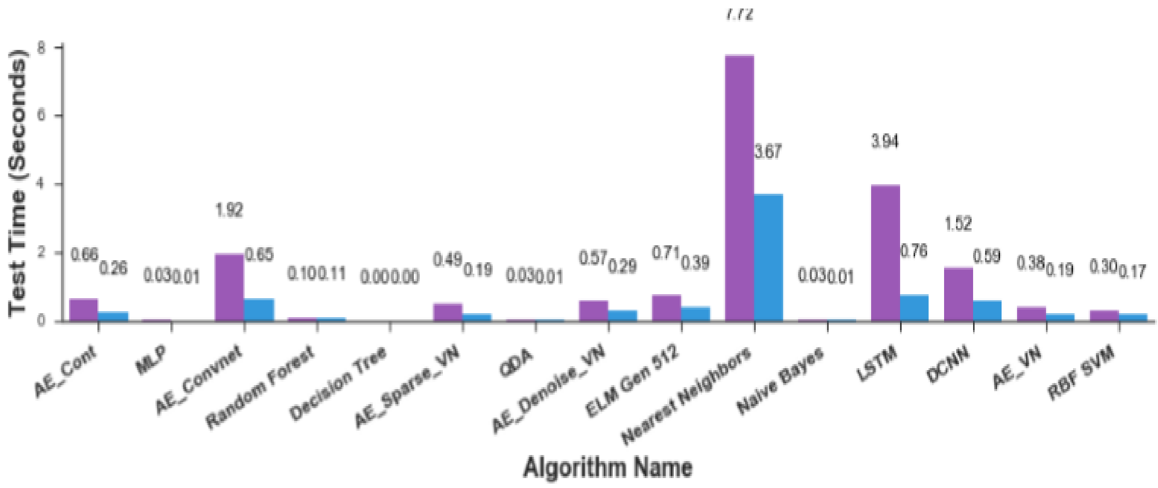


Fig 1.6 Test Times for Datasets

Both DCNN and LSTM models showed exceptional performance with 85% and 89% Accuracy on test dataset which demonstrates the fact that Deep learning is not only viable but rather promising technology for information security applications like other application domains.

Model	mAP for NSLKDDPlus	Model	mAP for NSLKDD21
DCNN	0.97	DCNN	0.98
LSTM	0.97	LSTM	0.97
Decision Tree	0.96	Convolutional AE	0.97
Contractive AE	0.95	Contractive AE	0.97
k-NN	0.95	k-NN	0.96
AutoEncoder	0.95	Decision Tree	0.95

Fig 1.7 Algorithm Mean Averages

1.3 ENCRYPTED PROTOCOLS DESCRIPTION

There a good range out traffic encryption protocols out there. We shall discuss two commonly used protocols in this section: TLS, SSH. Encryptions simply means encoding data in such a way that it not recognizable to anyone except people with the keys to decrypt and read what the data says. Of course, this means that the keys will only be available to the parties communicating. All protocols that provide encryption look to provide the same service, which is, confidentiality, some level of authentication between the communicating parties data integrity and non repudiation.

A greater portion of encryption protocols work in the same manner: the initialization of the connection and transport of encrypted data. It involves a handshake and a shared secret key for ecample. During this step the communicating parties exchange what kind of algorithm is used for encryption, communicating parties are authenticated and then the secret key established. These keys are used to encrypt the data to be transferred between parties.

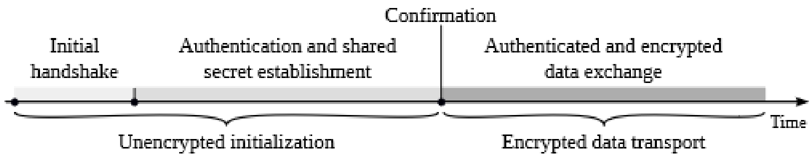


Figure 1. A general scheme of network security protocols.

Fig 1.8 general encryption scheme

Transport Layer Security (TLS) [25] is based on Secure Socket Layer version 3 (SSLv3) protocol [26]. It provides security directly on TCP which is a transport layer protocol. It provides the features mentioned above which include but not limited to: data integrity, confidentiality and authentication. It does this using certificates. Protocols like HTTP, FTP, SMTP, are know to use TLS as security. It is also used in VPN and VoIP.

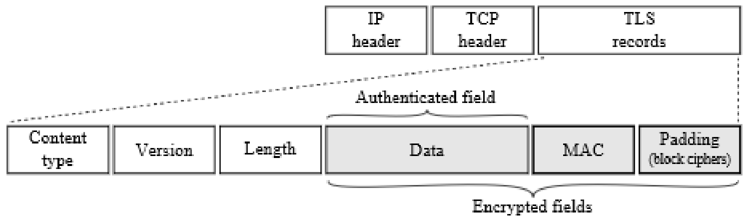


Fig 1.9 TLS packet format

In the first phase of a TLS connection, communicating parties are authenticated using an X.509 certificates chain as shown in the general scheme in Fig 1.8. Alternatively, a previous connection can be resumed without authentication. TLS messages exchanged during this phase are unencrypted and do not contain MAC until the shared keys are established and confirmed. In the second phase, these keys are used directly by the Record Protocol, which is based on the selected algorithms ensuring communication security

Secure Shell Protocol (SSH): SSH is an application that runs over tcp. It uses a client-server model. The server listens on port 22 (standard port for SSH). It replaced telnet for remote login as telnet is unsecure. As tie went on, it developed into being used for more than just secure login. It can be use for secure file transfer through SFTP and SCP. It also provides authentication, data integrity and confidentiality like TLS

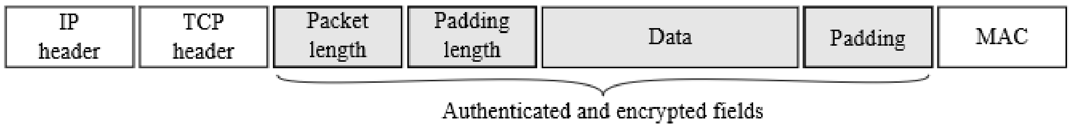


Fig1.10 SSH protocol packet format

Every SSH connection goes through the same phases which were depicted in Figure 1. In the first phase, a TCP connection is established and information about preferred algorithms is exchanged. During authentication, a server sends its public key which must be verified by the client. The shared keys are subsequently established and confirmed. All following packets are then encrypted and authenticated

Note that there are other encryption traffic protocols available that have not been discussed in this paper such as BitTorrent[27], Skype[28] etc.

SECTION II

2.1 NETWORK TRAFFIC CLASSIFICATION TECHNIQUES

A great deal of interest has suddenly erupted in the field of network traffic classification. This has led to a great number of researches and seen researchers employ different methods and techniques to classify network traffic.

The more technology evolved the more methods and techniques have been developed. In the last two decades, a number of techniques have been introduced into the industry by researcher or engineers looking to classify network for a number of reasons. This chapter discusses several techniques that have been employed in network traffic classification.

2.1.1 Port-based classification: identifying and classifying network traffic in the early days, did not pose any hassle. Simply inspecting the packet header and matching the TCP or UDP port number with the appropriate authority was enough. What this means was there are applications that were known to specific ports, for example, HTTP port 80, SSH port 22. This was used for a long time until of course, applications started to use unregistered or non standard ports. Some applications used random port numbers. Some unknown applications hid behind well known applications in order to bypass restrictions access controls or firewalls. This led to a decline in use of this technique because it became inaccurate and unreliable as different for the reasons mentioned above.

2.1.2 Behavioral classification: this technique observes the whole network traffic that comes in a node and tries to identify or classify traffic based on a pattern from the target node. This takes into consideration the number of hosts the port number and number of ports. Some works like in[7,8] sought to analyze network traffic patterns by exploiting heuristic information such as the number of distinct ports contacted, as well as transport layer protocols to distinguish the type of application running on a host. Other works[9, 10] showed that a lot of information can be utilized to classify network traffic. They analyzed the connections between endpoints graphically, and they show that generated connection patterns and graphs from client-server applications are very different than those of P2P.

2.1.3 Payload classification: This is sometimes called deep packet inspection(DPI). The widely used payload-based technique involves matching some stored signatures or pattern with feature of packets that are inspected. Thus technique has been employed in several researches and tools because of its high accuracy and reliability. A good example of this is in the Linux Kernel Firewall[11]. This techniques is also employed in intrusion detection systems (IDS) to identify threatening or suspicious traffic that can cause damage and leak of information to a network. Although a very efficient and accurate technique, it poses some disadvantages or weakness, when dealing with encrypted traffic, its abilities a minute as it cant inspect these kind of packets and they remain unclassified. Also present is the act of privacy breach. Inspecting encrypted traffic could break laws of certain countries. It uses a lot of computer resources hence doing this technique comes at a cost. It is also limited when it comes to a high number of traffic flows and network speed in real time.

2.1.4 Statistical classification: this method uses some flow features of packet for classification. Some features may include, duration of packet, packet size, flow idle time etc. some of the above mentioned features are unique for some applications this enables the technique classify between traffic for different applications. To perform the actual classification based on statistical characteristics, classifiers need to employ data mining techniques, specifically ML algorithms, because they need to deal with different traffic patterns from large datasets[12]. ML algorithms are very lightweight and less computationally expensive than payload-based classification techniques, because they do not depend on DPI but rather utilize the information from flow-level analysis. The effectiveness of the classifier in statistical classification depends on the features extracted from the flow, which require extensive knowledge due to their complexity. However, these techniques outperform payload-based techniques since they do not deal with packet contents, and thus can analyze encrypted traffic without any difficulty.

2.2 UNDERSTANDING ARTIFICIAL NEURAL NETWORKS(ANN)

Artificial neural networks[5] are sometimes just called neural networks.it investigates how biological brains can solve tough tasks like prediction tasks in machine learning. The strength of neural networks is their ability to learn the representations in training data and relating it to output variable that needs to be predicted. In other words they learn a mapping. They are capable of mapping any function and are proven to be good approximation algorithm. The hierarchical and multilayer structure they have ensures their predictive capabilities. They can learn features at different scales or resolutions and combine them into features of a higher lever or order. In other words learn features such as lines, and combine them to learn the shapes those lines form and then the full image as the case may be.

Neurons are the building block of neural networks just like in a biological brain. They contain simple computational units with inputs signals that are weighted and with the help of an activation function produces an output.

Weights, like linear regressions the neurons have biases which may have the value 1.0. larger weights means more complex and fragility. Techniques can be used to keep the weights in a network small as this is best practice.

Activation this encompasses the threshold at which the neuron is activated and also how strong the output signal is.

A row of neurons is called a layer. So having multiple layers of neurons that are connected is know as a network, hence artificial neural network. Basically there are input layer, which takes in the training data and is visible, hidden layer, which trains the network. There can be multiple layers is the hidden layer. The deeper it is the slower it is to train the network. The hidden layer is not visible to the input layer. And lastly the output layer also hidden produces a value that correspond to the format needed to solve the problem.

2.3 WHAT IS DEEP LEARNING?

Deep learning is a model based on Artificial Neural Networks (ANN), more specifically Convolutional Neural Networks (CNN)s. There are several architectures used in deep learning such as deep neural networks, deep belief networks, recurrent neural networks, and convolutional neural networks. These networks have been successfully applied in solving the problems of computer vision, speech recognition, natural language processing, bioinformatics, drug design, medical image analysis, and games.

2.3.1 DEEP LEARNING TECHNIQUES

There are considerable ranges of deep learning techniques used across the globe for various tasks. Such tasks could vary from image recognition, voice recognition and or other classification tasks. What technique is used depends on the researcher and the aim of the research being carried out. Deep learning is based on Artificial Neural Networks(ANN). For example, convolutional neural networks are best for image classification and prediction tasks and has grown quite popular among researcher in recent years. We discuss below a few deep learning techniques.

2.3.1.1 Multilayer Perceptron: this consists of an input layer that receives signal, a hidden layer that trains the network and an output layer that predicts or makes a decision based on the input. It is mostly used in supervised learning. Weights and biases are adjusted as needed to reduce error. It is a feed forward network. In the forward pass, the signal moves from the input layer that contains the data set and through the hidden layer that trains the network and then to the output layer that gives a value as needed for the problem to be solved. The networks uses a backward propagation and a rule or rules of calculus to reduce error. This keeps happening until the error can go no lower. This is a convergence state. In regards to network traffic classification, this technique is rarely used due to low accuracy and a high complexity.

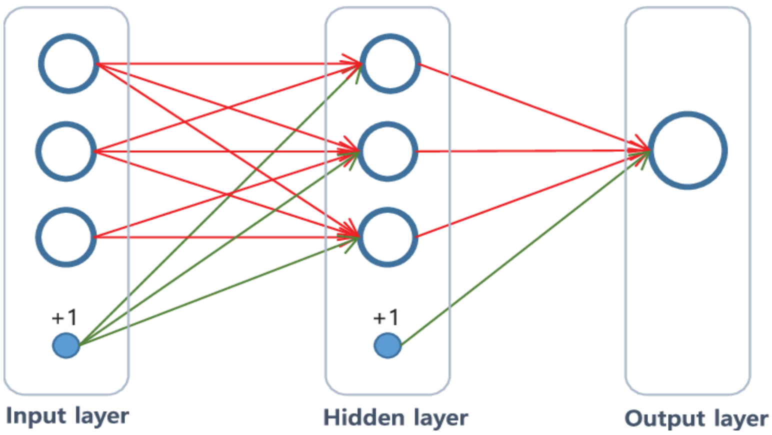


Fig 2.1 Basic Multilayer Architecture.

2.3.1.2 Convolutional Neural Network: this is similar to multi-layer perceptron in architecture but has more capabilities and can handle a lot more data. The objective is to extract high level features such as edges from input images. They are not limited to one convolutional layer. The first layer, the convolutional layer, extracts low level features like line, edges, color etc. as more layers are added, higher level features can be identified. This can enable the network have an understanding of images as humans would. This is the convolutional layer.

Pooling layer is responsible for reducing the spatial size of the convolved feature. This helps decrease the computational power required to process the data. There are two types of pooling the average pooling and the max pooling. Max pooling returns the maximum value from the portion of the image covered by the kernel while average pooling is the average value of the portion of the image covered by the kernel.Fully connected layer or FC layer learns the output of the convolutional layer. It learns a non-linear function in that space. After formatting the input image into a suitable form, it is flattened into a single column vector. This is then passed into a feed forward neural network and backpropagation is applied to every iteration of training. After a few epochs, the model is able to differentiate between features and classify them using the provided activation technique.This technique has been the most widely used for traffic classification and it is used in this paper for or task as well.

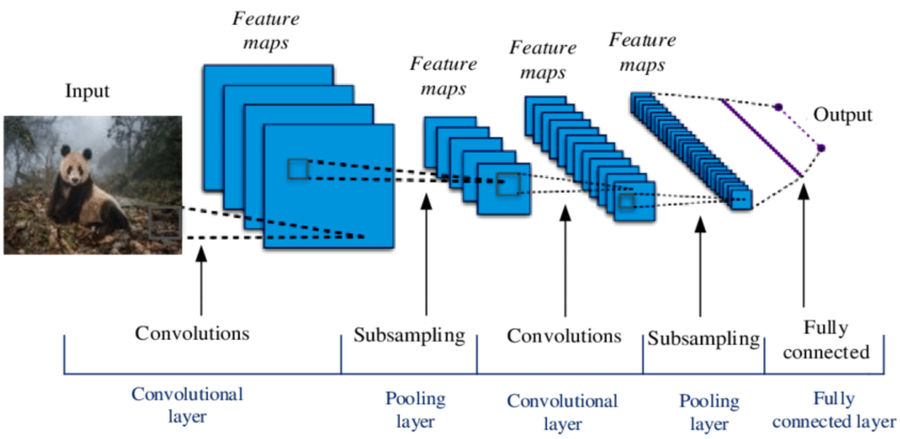


Fig2.2. A simple CNN architecture.

2.3.1.3 Autoencoders: take any input and break down don into a compressed version. It is then used to reconstruct the input data. Usually the hidden layer has limitations thereby keeping just the important information about the input data. It does this automatically without human intervention. Basically, there are an input layer that should be either encoded, an encoding function usually in the hidden layer then a decoding function that takes the encoded input and decodes it, loss function. An autoencoder is considered good when the decoded version is close or similar to the input data.

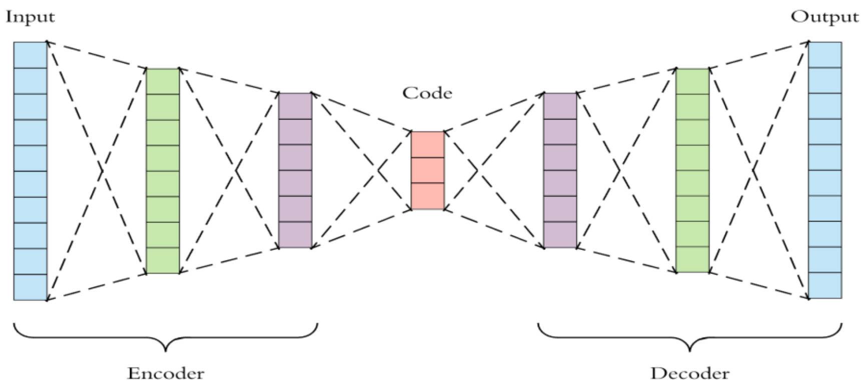


Fig 2.3 Auto-encoder architecture.

2.4 BUILDING A CONVOLUTIONAL NEURAL NETWORK

It took about 14 years for the research work by Yann LeCun on CNN to be noticed. It was brought into public view by a team of researchers during the 2012 ImageNet Computer vision competition. As at the time the architecture called AlexNet after Alex Krizhevsky was quite successful with an error or only 15.8%. it classified millions of images from thousands of categories. Currently CNN are capable of accuracies that surpass even the human performance.

To build a CNN, a programming language such as python or R is used. Python is widely used in researches across the globe as it has more libraries and packages that greatly improve and cater to Machine learning tasks.

In order to build a CNN we need a problem to solve and the dataset.

i.e. train dataset and test dataset.

Datasets have to be preprocessed and provided with labels. And then a one hot encoder can be used depending on data to be preprocessed.

```
def label_img(img):
        word_label = img.split('.')[-3]

# DIY One hot encoder
        if word_label == 'cat': return [1, 0]
        elif word_label == 'dog': return [0, 1]
```

Libraries required:

- **TFLearn** – Deep learning library featuring a higher-level API for TensorFlow used to create layers of our CNN
- **tqdm** – Instantly make your loops show a smart progress meter, just for simple designing sake
- **numpy** – To process the image matrices
- **open-cv** – To process the image like converting them to grayscale and etc.
- **os** – To access the file system to read the image from the train and test directory from our machines
- **random** – To shuffle the data to overcome the biasing
- **matplotlib** – To display the result of our predictive outcome.
- **tensorflow** – Just to use the tensorboard to compare the loss and adam curve our result data or obtained log.

The mentioned libraries above are then imported

```
# Python program to create

# Image Classifier using CNN
```

```
# Importing the required libraries
import cv2
import os
import numpy as np
from random import shuffle
from tqdm import tqdm

'''Setting up the env'''

TRAIN_DIR = 'E:/dataset / Cats_vs_Dogs / train'
TEST_DIR = 'E:/dataset / Cats_vs_Dogs / test1'
IMG_SIZE = 50
LR = 1e-3



'''Setting up the model which will help with tensorflow models'''
MODEL_NAME = 'dogsvscats-{}-{}.model'.format(LR, '6conv-basic')

'''Labelling the dataset'''
def label_img(img):
    word_label = img.split('.')[-3]
    # DIY One hot encoder
    if word_label == 'cat': return [1, 0]
    elif word_label == 'dog': return [0, 1]

'''Creating the training data'''
def create_train_data():
    # Creating an empty list where we should store the training data
    # after a little preprocessing of the data
    training_data = []

    # tqdm is only used for interactive loading
    # loading the training data
    for img in tqdm(os.listdir(TRAIN_DIR)):

        # labeling the images
        label = label_img(img)

        path = os.path.join(TRAIN_DIR, img)
```

```
        # loading the image from the path and then converting them
into
        # greyscale for easier covnet prob
        img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)

        # resizing the image for processing them in the covnet
        img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))

        # final step-forming the training data list with numpy array
of the images
        training_data.append([np.array(img), np.array(label)])

    # shuffling of the training data to preserve the random state of
our data
    shuffle(training_data)

    # saving our trained data for further uses if required
    np.save('train_data.npy', training_data)
    return training_data

'''Processing the given test data'''
# Almost same as processing the training data but
# we dont have to label it.
def process_test_data():
    testing_data = []
    for img in tqdm(os.listdir(TEST_DIR)):
        path = os.path.join(TEST_DIR, img)
        img_num = img.split('.')[0]
        img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
        testing_data.append([np.array(img), img_num])

    shuffle(testing_data)
    np.save('test_data.npy', testing_data)
    return testing_data

'''Running the training and the testing in the dataset for our
model'''
```

```
train_data = create_train_data()
test_data = process_test_data()

# train_data = np.load('train_data.npy')
# test_data = np.load('test_data.npy')
'''Creating the neural network using tensorflow'''
# Importing the required libraries
import tflearn
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression

import tensorflow as tf
tf.reset_default_graph()
convnet = input_data(shape =[None, IMG_SIZE, IMG_SIZE, 1], name
='input')

convnet = conv_2d(convnet, 32, 5, activation ='relu')
convnet = max_pool_2d(convnet, 5)

convnet = conv_2d(convnet, 64, 5, activation ='relu')
convnet = max_pool_2d(convnet, 5)

convnet = conv_2d(convnet, 128, 5, activation ='relu')
convnet = max_pool_2d(convnet, 5)

convnet = conv_2d(convnet, 64, 5, activation ='relu')
convnet = max_pool_2d(convnet, 5)

convnet = conv_2d(convnet, 32, 5, activation ='relu')
convnet = max_pool_2d(convnet, 5)

convnet = fully_connected(convnet, 1024, activation ='relu')
convnet = dropout(convnet, 0.8)

convnet = fully_connected(convnet, 2, activation ='softmax')
convnet = regression(convnet, optimizer ='adam', learning_rate = LR,
      loss ='categorical_crossentropy', name ='targets')
```

```
model = tflearn.DNN(convnet, tensorboard_dir ='log')

# Splitting the testing data and training data
train = train_data[:-500]
test = train_data[-500:]

'''Setting up the features and lables'''
# X-Features & Y-Labels

X = np.array([i[0] for i in train]).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
Y = [i[1] for i in train]
test_x = np.array([i[0] for i in test]).reshape(-1, IMG_SIZE,
IMG_SIZE, 1)
test_y = [i[1] for i in test]

'''Fitting the data into our model'''
# epoch = 5 taken
model.fit({'input': X}, {'targets': Y}, n_epoch = 5,
    validation_set =({'input': test_x}, {'targets': test_y}),
    snapshot_step = 500, show_metric = True, run_id = MODEL_NAME)
model.save(MODEL_NAME)

'''Testing the data'''
import matplotlib.pyplot as plt
# if you need to create the data:
# test_data = process_test_data()
# if you already have some saved:
test_data = np.load('test_data.npy')

fig = plt.figure()

for num, data in enumerate(test_data[:20]):
    # cat: [1, 0]
    # dog: [0, 1]

    img_num = data[1]
    img_data = data[0]

    y = fig.add_subplot(4, 5, num + 1)
```

```
        orig = img_data
        data = img_data.reshape(IMG_SIZE, IMG_SIZE, 1)

        # model_out = model.predict([data])[0]
        model_out = model.predict([data])[0]

        if np.argmax(model_out) == 1: str_label ='Dog'
        else: str_label ='Cat'

        y.imshow(orig, cmap ='gray')
        plt.title(str_label)
        y.axes.get_xaxis().set_visible(False)
        y.axes.get_yaxis().set_visible(False)
    plt.show()
```

Obviously for the task depending on the task at hand the program can be re written to suit the network as needed. This is just an example of a flow on how a basic CNN can be programmed.

SECTION III

3.1 ENVIRONMENT SETUP

Tools and version used:

GNS3 v2.22

VMWare workstation player 15

3.1.1How to install vmware workstation player 15

First, we visit the website (https://my.vmware.com/web/vmware/downloads) to download the player (We use this to run our gns3 server). Or visit the direct link (https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html)

Step 1 – Run the installer

Start the installer by double clicking it. You might see User Account Control Warning. Click Yes to continue.

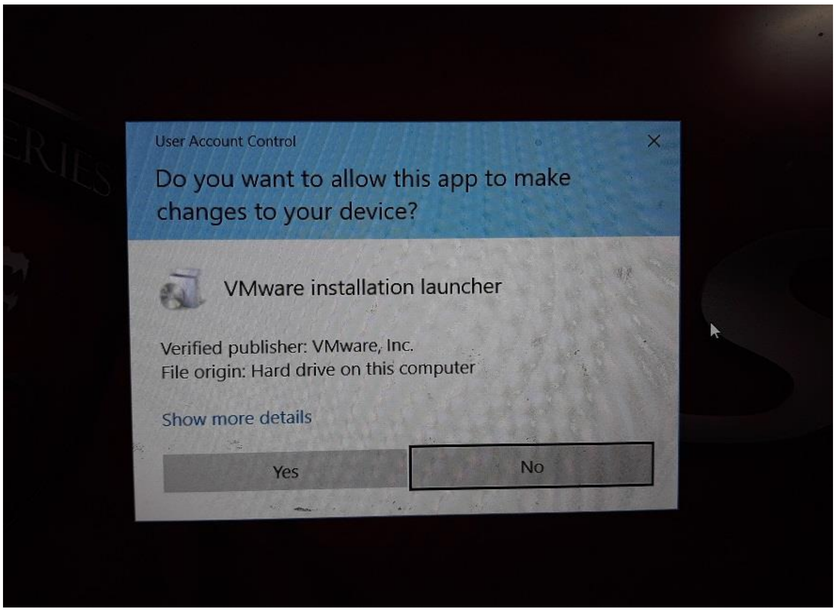


Fig 3.1 User Access Control

Then, you will see a splash screen. It will prepare the system for installation and then the installation wizard opens.

Fig 3.2 Splash Screen

Fig 3.3 setup wizard

Click next and accept the license terms and click next again to move on to the next screen.

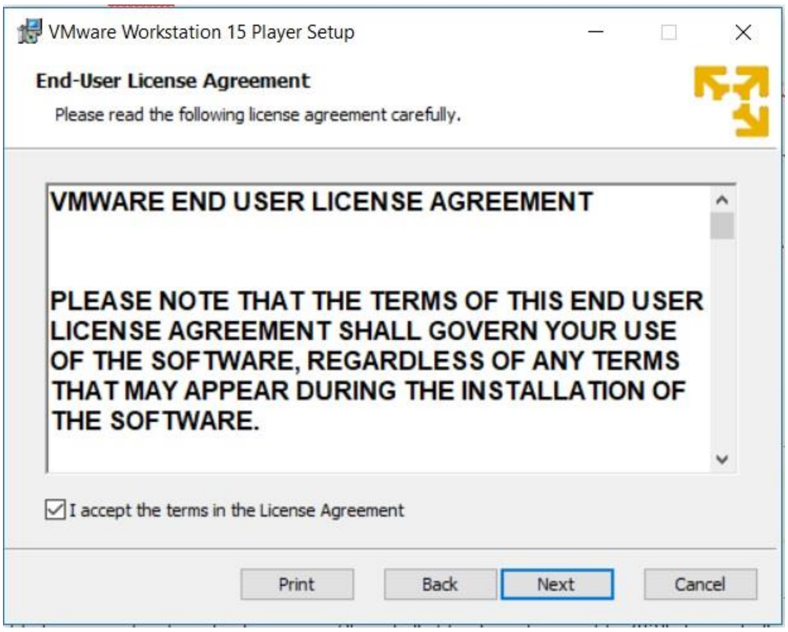


Fig 3.4 User agreement

Step 2 – Custom setup – Enhanced Keyboard driver and Installation directory

In this dialog box, please select the folder in which you want to install the application. I leave it as it is. Also check the box Enhanced Keyboard Drivers option. Click next.

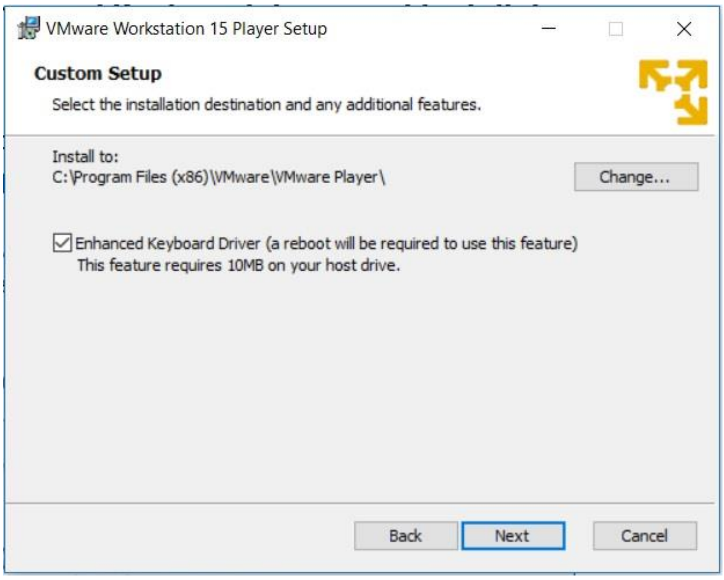


Fig 2.5 Keyboard Driver

Step 3 – User Experience Settings

Check the options for Check the product update at Startup and Join the VMware Customer Program. I normally leave it as it is. You can unchecked it if you so desire. Click next

Step 4 – Select where the shortcuts will be installed

Check the box where the shortcut to run the application will be created. I leave it as it is. Click on next.

Step 5 – Ready to install

Now the installation wizard is ready to install. Click on install to begin the installation.

Installation begins, wait for it to complete.

After sometime, you will see installation compete message. You are done.

Click on Finish to Complete the installation.

You will be asked to restart your system. Click on Yes to restart. Click No, if you want to restart later. But you must restart before using the application, else some features will not work properly.

Step 6 – License

Now Run the application. You should see a desktop icon. Douple click on that or use the start menu to navigate to VMware Player option.

Once you run the application for the first time, you will be asked for licence. Select the option Use VMware Workstation Player 15 for for free for non commercial use.

Click continue.

Fig 3.6 liscence

Click on Finish.

Fig 3.7 finished install

Now you will see VMware Workstation Player 15 ready to be used for free for non-commercial purpose.

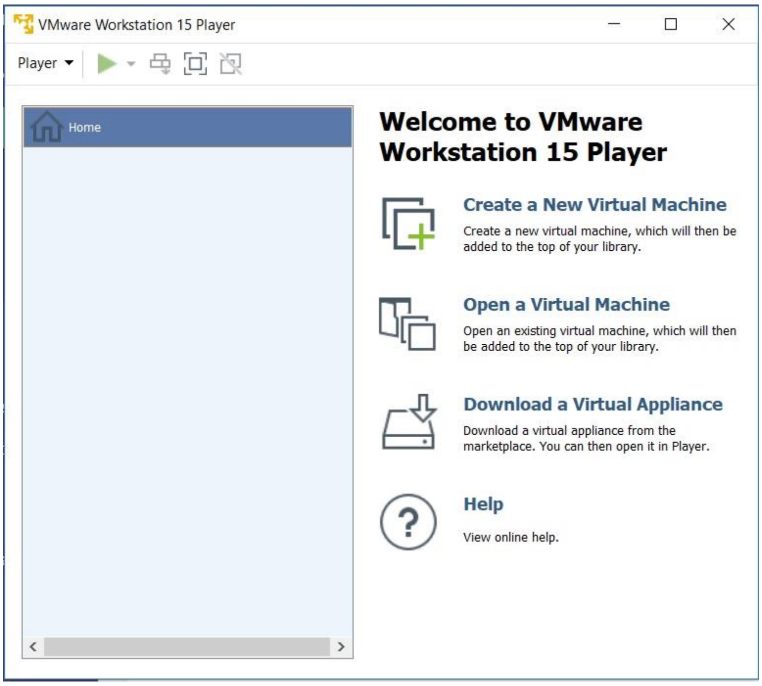


Fig 3.8 vmware window

3.1.2 GNS3 SETUP

- Visit the gns3 link to download the installer (https://www.gns3.com/software/download)

- Click twice on your downloaded GNS3 Windows installer file (GNS3-2.2-all-in-one.exe). A security warning window will appear. Inside this window, click on Run button.

- GNS3-2.2 Setup starting window will appear to welcome you. Nothing to do in this window. Just click on Next button.

- License Agreement window will appear. Accept the license agreement clicking the I Agree button.

- Choose Start Menu Folder window will appear. Keep default name (GNS3) or if you wish you can change it. Click on Next button.

- Choose Components window will appear where available GNS3 features will be listed. Among these features uncheck only Wireshark, SolarWinds Response and Npcap features because initially we don't require these features. Now click Next button.

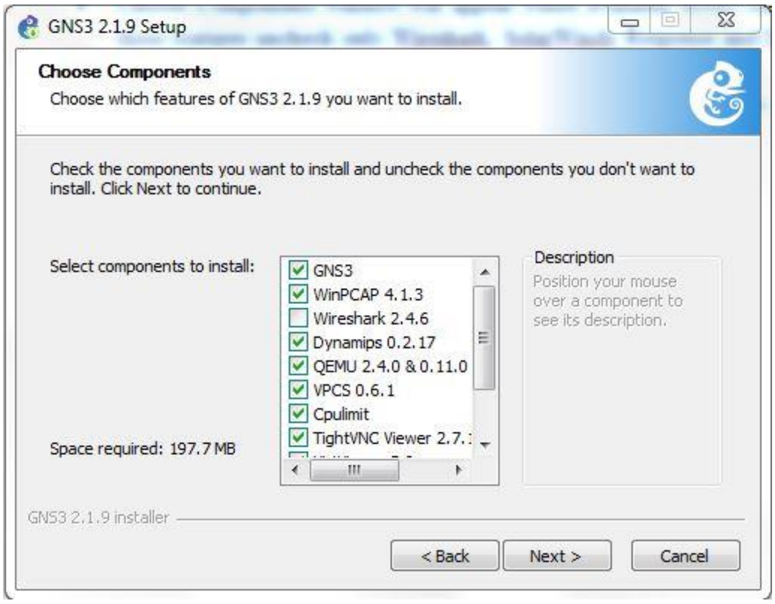


Fig 3.9 features list

- Choose Install Location window will appear. Keep default location or if you wish you can change browsing destination folder. Now click Install button.

- GNS3 features installation will be started and installation progress will be found on progress bar. During GNS3 installation, WinPCAP installation will be appeared separately. Follow some easy instructions as indicated. Also keep your internet connection OK because virt-viewer will be downloaded during GNS3 installation.

- Within a few minutes, GNS3 installation will be completed and Installation Complete window will appear with success message. Click Next button from this window.

- Solarwinds Standards Toolset window will appear. We don't need any toolset now. So, click on No radio button and then click on Next button.

- GNS3 Setup close window will appear. Click Finish button. GNS3 installation will be finished and GNS3 will start to run now.

3.1.3 INSTALLING THE GNS3 VM ON WMWARE PLAYER

Visit the gns3 vm download link https://www.gns3.com/software/download-vm
Since we are using vmware player we only download the virtual machine image for wmware player

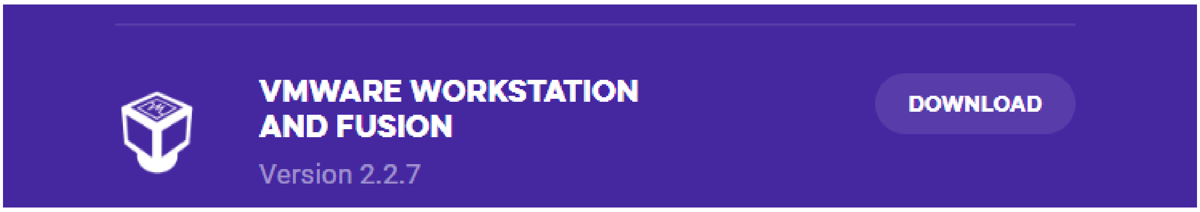


Fig 3.10 VMware version

This is the downloaded vm image

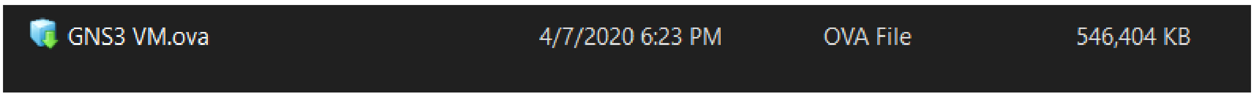


Fig3.11 Image file

Next run the vmware player

Click on File > Open

Select the file path for th gns3 vm image (the .ova file)

Then click import

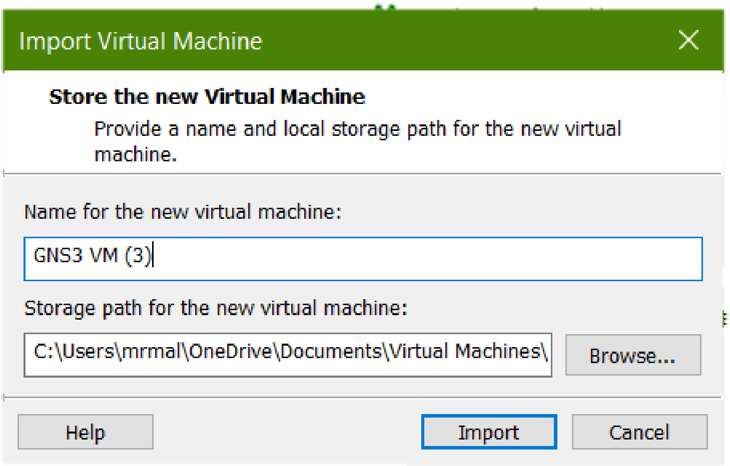


Fig3.12 Import VM

Depending on your system resources, you can decide to adjust ram size as you see fit.

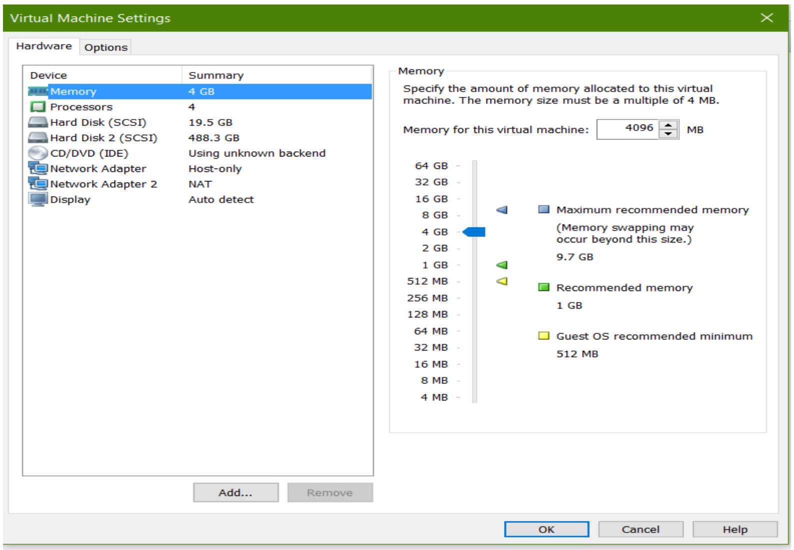


Fig 3.13 resources

I used 4GB of Ram.

Start the GNS3 version 2.x, and then from the Help tab click on Setup Wizard.

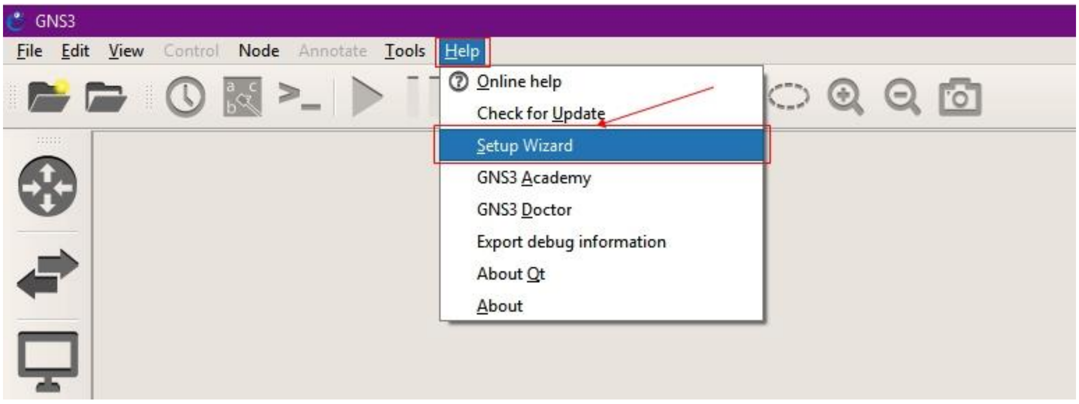


Fig 3.14 setup wizard

Select the Server option 'Run Modern IOS (IOSv or IOU), ASA and appliances from non-Cisco manufacturers' and click on Next

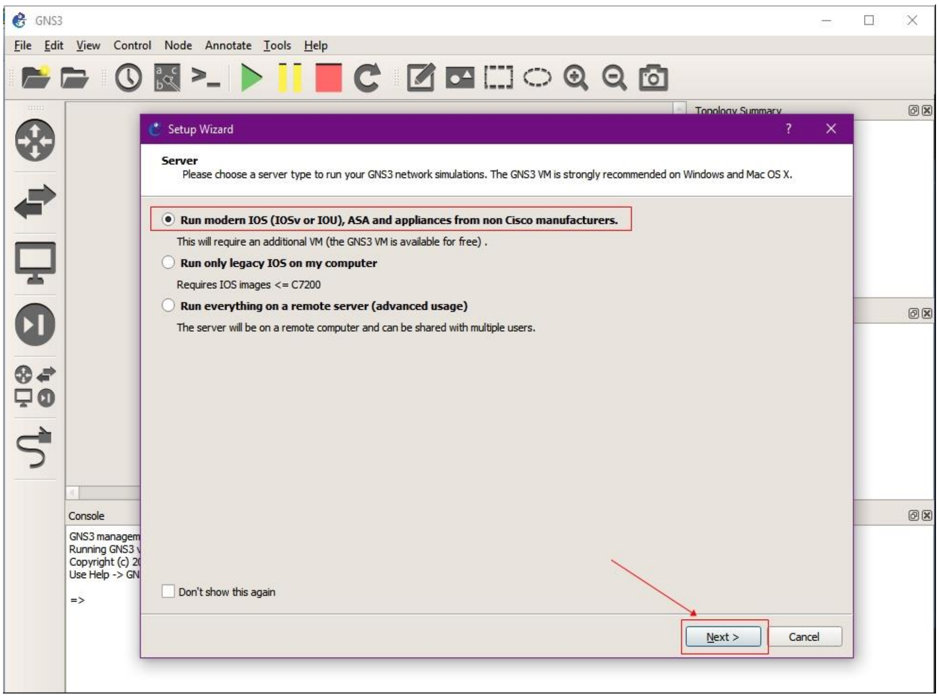


Fig 3.15 Appliance

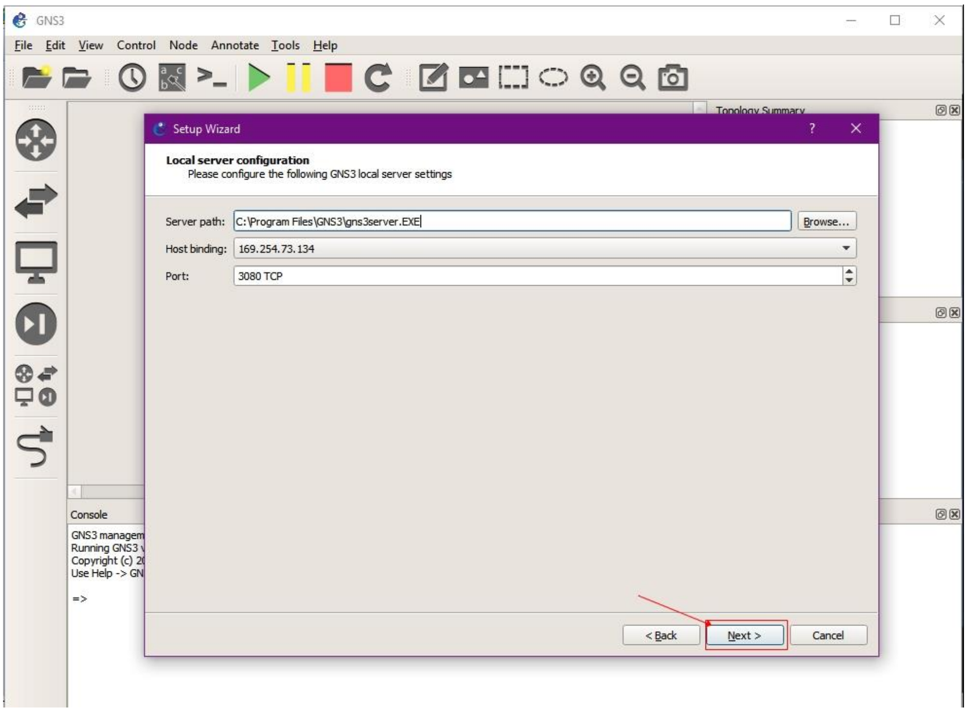


Fig 3.16 server configuration

In the Local server configuration, whatever the IP address and TCP Port no. which is 3080 we will select now, next time it will use the same combination for running the Local server.

If you will face such type of error select the IP address 127.0.0.1 from the list.

From this point we will associate our GNS3 VM with GNS3. Click on Refresh button in case of error.

Select the GNS3 VM.

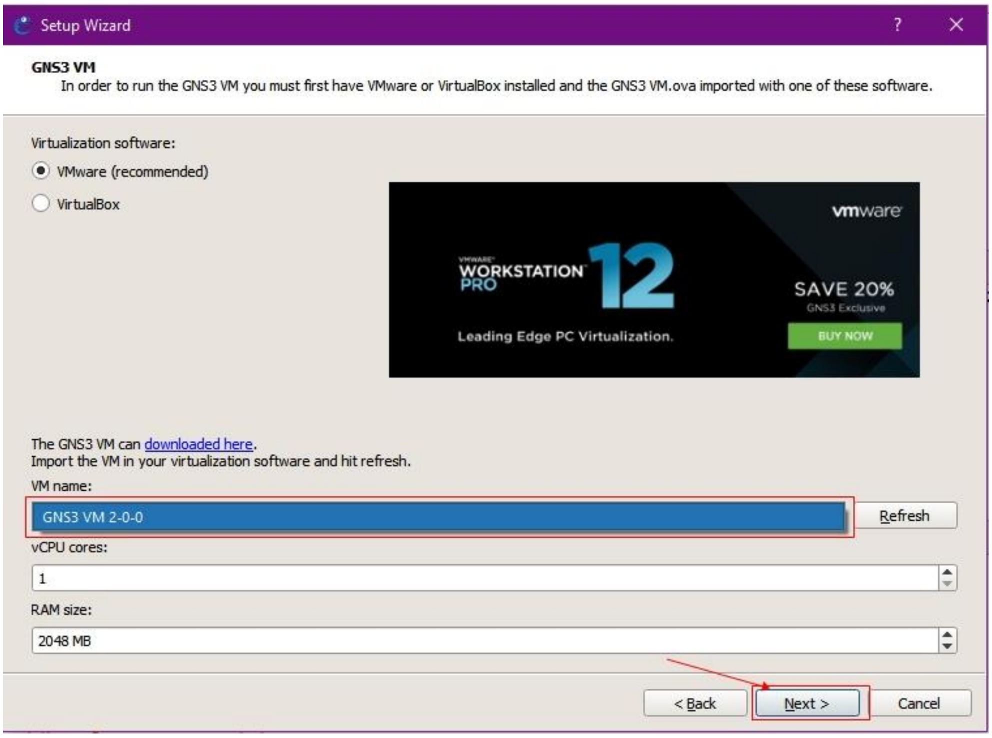


Fig 3.17 select gns3 vm

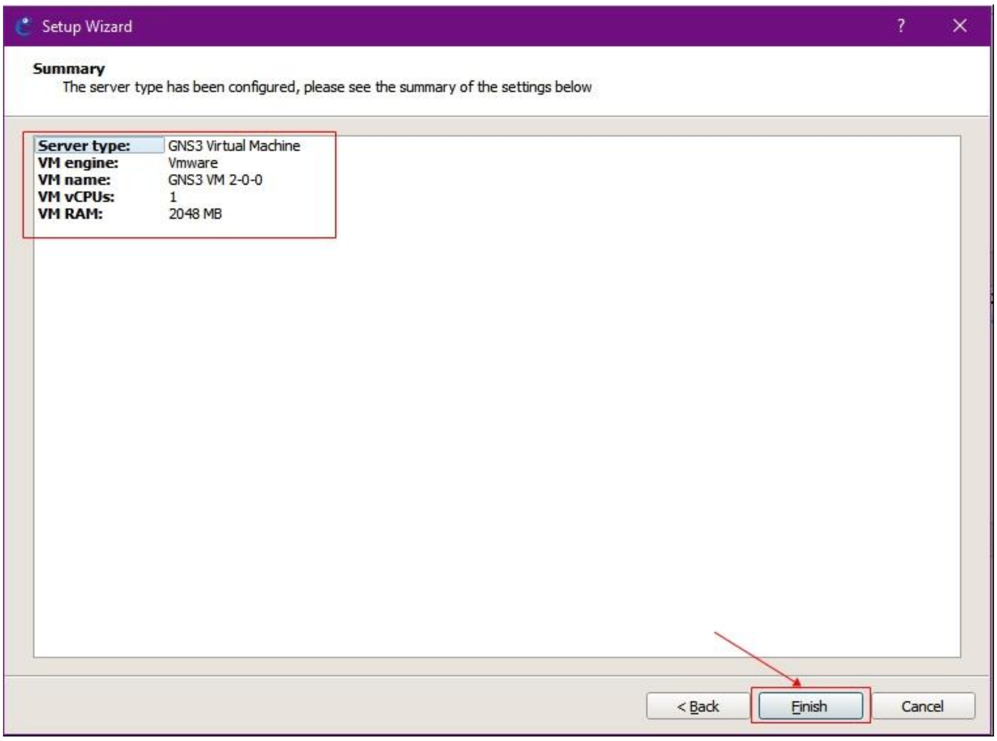


Fig 3.18 finish setup

SECTION IV

EXPERIMENT

4.1 GATHHERING PROTOCOL TRAFFIC

Traffic for three protocols were gathered. POP3, FTP and DNS traffic.

Using the gns3 vm we were able to setup a network environment consisting of a client a switch a NAT cloud for internet connection and a server to run the services on. All servers were Ubuntu docker containers as well as clients. Both plain traffic and secure traffic were gathered using this method.

4.1.2 POP3 TRAFFIC:

We setup two network devices, Ubuntu docker containers, one serves as the SMTP POP3 server and the other as the client.

We also introduce a NAT cloud to help us have connectivity to the internet so we can download the necessary packages to run the services we need.

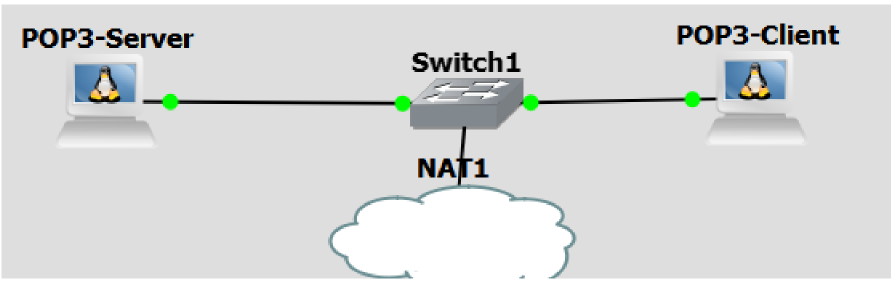


Fig 4.1 POP3 topology

Next we open a command line interface to the server node and run the following command:

-sudo apt-get install postfix

Postfix is a mail transfer agent. This enables us sent mail from one user to the other using SMTP(simple mail transfer protocol).

We edit the lines in

-vi /etc/postfix/main.cnf

Next we install dovecot. This is a mail delivery agent that lest clients check and read their emails either downloaded from the mail server(POP3) or on the mail server (IMAP).

-sudo apt-get install dovecot-pop3d dovecot-imapd

After installation and configuration, we can now use telnet to send mail and then check mail box. Also we start a wire capture on the client link

We use ‘telnet ip address of smtp server and then the port’

-telnet 192.168.122.251 25

Fig 4.2 Telnet session

Next command is the ‘ehlo’ this is the first command when using smtp to send messages. We say ehlo and our FQDN in my case ‘server.example.com’

-ehlo server.example.com

Next we use the command ‘mail from:’ to choose the sender and ‘rcpt to:’ for the receiver. ‘data’ indicates the start of the mail body.

-mail from: tony@server.example.com

-rcpt to: ghost@server.example.com

-data

-subject: test

Then we type our message and then indicate the end of the message by typing ‘.’ on a new line alone.

```
220 server.example.com ESMTP
ehlo server.example.com
250-server.example.com
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
250-ENHANCEDSTATUSCODES
250-8BITMIME
250-DSN
250 SMTPUTF8
mail from:tony@server.example.com
250 2.1.0 Ok
rcpt to:ghost@server.example.com
250 2.1.5 Ok
data
354 End data with <CR><LF>.<CR><LF>
subject: reply
this is a reply from me to you
.
250 2.0.0 Ok: queued as 36818801464
quit
221 2.0.0 Bye
Connection closed by foreign host.
```

Fig 4.3 POP3 commands

-quit

The ‘quit’ command terminates the connection to the server.

Now to access the mail that ghost just received, we start a telnet connection to the server this time on a different port 143 since pop3 runs on that port.

-telnet 192.168.122.251 110

We login to our user account to check our mailbox

-user ghost

-pass 12345

We can check our messages using the command ‘list’ -list

```
list
+OK 1 messages:
1 365
```

Fig 4.4 List command

We can see that there’s one message in ghost’s inbox.

The command ‘retr’ helps download the message from the server for us to read

-retr 1

```
retr 1
+OK 365 octets
Return-Path: <tony@server.example.com>
X-Original-To: ghost@server.example.com
Delivered-To: ghost@server.example.com
Received: from server.example.com (unknown [192.168.122.62])
        by server.example.com (Postfix) with ESMTP id 36818801464
        for <ghost@server.example.com>; Mon,  4 May 2020 14:15:57 +0000
 (UTC)
subject: reply

this is a reply from me to you
.
```

Fig 4.5 Retr command

4.1.2.1 ANALYSIS OF POP3

Using Wireshark (a packet sniffing tool) we were able to capture as packets were moving from the client to the server.

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

pop

No.	Time	Source	Destination	Protocol	Length	Info
39	68.457534	192.168.122.251	192.168.122.62	POP	86	S: +OK Dovecot ready.
60	98.875446	192.168.122.62	192.168.122.251	POP	78	C: user ghost
62	98.877014	192.168.122.251	192.168.122.62	POP	71	S: +OK
66	103.644408	192.168.122.62	192.168.122.251	POP	78	C: pass 12345
68	103.750727	192.168.122.251	192.168.122.62	POP	82	S: +OK Logged in.
73	108.877381	192.168.122.62	192.168.122.251	POP	72	C: list
75	108.880833	192.168.122.251	192.168.122.62	POP	93	S: +OK 1 messages:

Fig 4.6 wireshark capture

We see how packets containing our data which include username, passwords and even the contents of our email messages.

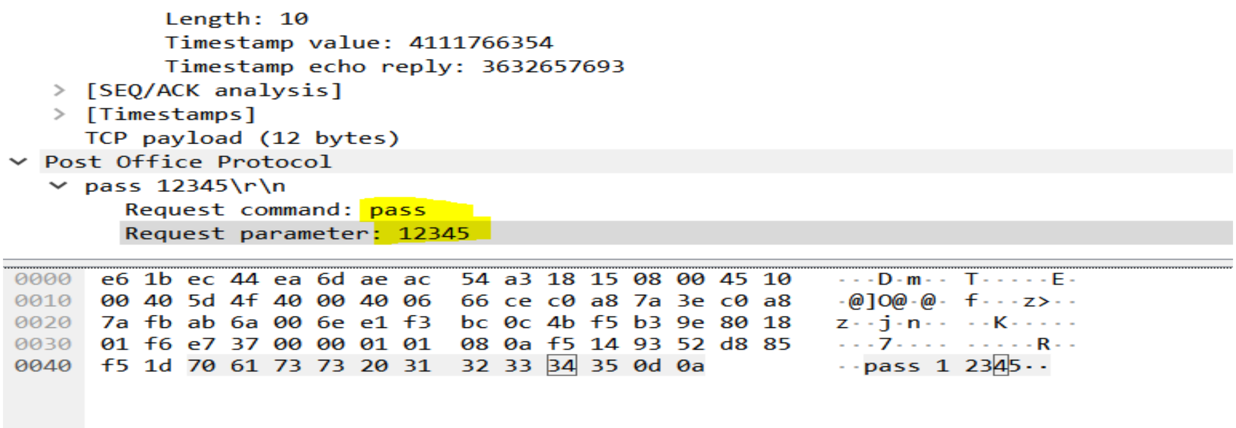


Fig 4.7unencrypted data

POP3 by itself is not a secure way of accessing our messages. This proves that the traffic is unencrypted and data can be accessed using a tool like Wireshark or Tcpdump.

To get encrypted traffic, we use SSL/TLS. We use a self-signed certificate and make sure that connection between the client and server is secure.

First, we generate a private key

-openssl genrsa -aes128 -out server123.key 2048

Then we use the key to generate a certificate signing request file .csr

- openssl req -new -days 3650 -key server123.key -out server123.csr

We use the generated csr and key to generate a certificate

- openssl x509 -in server123.csr -out server123.crt -req -signkey server123.key -days 3650

We now move the files we have generated to the /etc/ssl/private directory

-mv server123.* /etc/ssl/private/

Now we point postfix and dovecot to use SSL during connections.

We edit the main.cf file in /etc/postfix/main.cf and add the following line to the end of the file

smtpd_use_tls = yes

smtp_tls_mandatory_protocols = !SSLv2, !SSLv3

smtpd_tls_mandatory_protocols = !SSLv2, !SSLv3

smtpd_tls_cert_file = /etc/ssl/private/server123.crt

smtpd_tls_key_file = /etc/ssl/private/server123.key

smtpd_tls_session_cache_database = btree:/etc/postfix/smtpd_cache

In the master.cf file in /etc/postfix/master.cf we uncomment a few lines as shown in the photo below

```
submission inet n       -       y       -       -       smtpd
  -o syslog_name=postfix/submission
#  -o smtpd_tls_security_level=encrypt
  -o smtpd_sasl_auth_enable=yes
  -o smtpd_tls_auth_only=yes
#  -o smtpd_reject_unlisted_recipient=no
#  -o smtpd_client_restrictions=$mua_client_restrictions
#  -o smtpd_helo_restrictions=$mua_helo_restrictions
#  -o smtpd_sender_restrictions=$mua_sender_restrictions
#  -o smtpd_recipient_restrictions=
#  -o smtpd_relay_restrictions=permit_sasl_authenticated,reject
#  -o milter_macro_daemon_name=ORIGINATING
smtps     inet  n       -       y       -       -       smtpd
  -o syslog_name=postfix/smtps
  -o smtpd_tls_wrappermode=yes
#  -o smtpd_sasl_auth_enable=yes
```

Fig 4.8 ssl config file

Next, we edit point dovecot by editing the /etc/dovecot/conf.d/10-ssl.conf file as shown below. Make sure to write in the correct path to the certificate and key files.

```
ssl = yes

# PEM encoded X.509 SSL/TLS certificate and private key. They're opened
 before
# dropping root privileges, so keep the key file unreadable by anyone b
ut
# root. Included doc/mkcert.sh can be used to easily generate self-sign
ed
# certificate, just make sure to update the domains in dovecot-openssl.
cnf
ssl_cert = </etc/ssl/private/server123.crt
ssl_key = </etc/ssl/private/server123.key

# If key file is password protected, give the password here. Alternativ
ely
# give it when starting dovecot with -p parameter. Since this file is o
ften
# world-readable, you may want to place this setting instead to a diffe
rent
# root owned 0600 file by using ssl_key_password = <path.
ssl_key_password =12345
```

Fig 4.9 ssl config file 2

Now restart both postfix and dovecot services.

Also POP3 with SSL runs on port 995 and we test that our traffic is now encrypted by coonecting via openssl to this port. We don't use telnet for this as telnet is not a secure protocol.

We use the following command to securely connect to our pop3 server

-openssl s_client -connect 192.168.122.251:995

Based on the photos below we see that our client and successfully carried out a ssl handshake with our server and opened a secure connection. All traffic moving forward, are encrypted and will not be seen by or decrypted without the private key.

```
root@POP3-Client:~# openssl s_client -connect 192.168.122.251:995
CONNECTED(00000003)
depth=0 C = UA, ST = kiev, L = kyiv, O = New, OU = IT, CN = server.exam
ple.com, emailAddress = ghost@example.com
verify error:num=18:self signed certificate
verify return:1
depth=0 C = UA, ST = kiev, L = kyiv, O = New, OU = IT, CN = server.exam
ple.com, emailAddress = ghost@example.com
verify return:1
---
Certificate chain
 0 s:/C=UA/ST=kiev/L=kyiv/O=New/OU=IT/CN=server.example.com/emailAddres
s=ghost@example.com
   i:/C=UA/ST=kiev/L=kyiv/O=New/OU=IT/CN=server.example.com/emailAddres
s=ghost@example.com
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDiDCCAnACCQDzG4TBVi8LfTANBgkqhkiG9w0BAQsFADCBhTELMAkGA1UEBhMC
VUExDTALBgNVBAgMBGtpZXYxDTALBgNVBAcMBGt5aXYxDDAKBgNVBAoMA05ldzEL
MAkGA1UECwwCSVQxGzAZBgNVBAMMEnNlcnZlci5leGFtcGxlLmNvbTEgMB4GCSqG
SIb3DQEJARYRZ2hvc3RAZXhhbXBsZS5jb20wHhcNMjAwNTA0MTcxMTIzWhcNMzAw
NTAyMTcxMTIzWjCBhTELMAkGA1UEBhMCVUExDTALBgNVBAgMBGtpZXYxDTALBgNV
BAcMBGt5aXYxDDAKBgNVBAoMA05ldzELMAkGA1UECwwCSVQxGzAZBgNVBAMMEnNl
cnZlci5leGFtcGxlLmNvbTEgMB4GCSqGSIb3DQEJARYRZ2hvc3RAZXhhbXBsZS5j
b20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC26J+L4+iJHaMQ6sOa
gEfU3Ww4j/nvNZU+ZbcS0ONglpHc+vn7AVJUrUQ79D/eNbQIwc7XvuiXIz7v3h7Y
iP7Dd2YNcEFtNLp1CR/F2RWt7dnEk8yIr1/WtJhcTLcsHpq2/9VBf+ZtG1eXXII7
jw8a9wydkw8y4BCDcgSEyZLrKJ1mC5r/qLNL1JnILwtrAsjmCckLbd5BEc+NCZOZ
jfNx7lgLp8I/hpToVPwEnV3AvTfLpTjXUaHLdHOtrq5SfmhwRRpwGEbvfH3iPC50
cbtw5CTTOWLDlpkm22cBCMydVtJLFFHNP0nD65lnQrBfCQ3GBntVU3NTB2zpuvZV
1BW1AgMBAAEwDQYJKoZIhvcNAQELBQADggEBAK4OcbKUNtRxxwBMsxnQaOjB+DZa
1bj6ZP37moPbcAVb4NU/VQq4zJNEx113ky24ohWLV58kMDXJfavcNWcJR9GtCeVy
CRMtK8ecQzTybrdBgkRooAIwm5FGgxuJ32qQvPCVMQuqx4br2BPf9YLtfLbfp6rb
jCgW9WvNQN6N0YglUHk8ECEygCEuSpSMWta6bKigvi/WvV5UlmJ2q3FMCrrHiMbJ
H0ov8piQSr99bQxzeE2eijY32P04Y4nv6j9tem5MfqqZ5Ggist9+fEzb39fK+dwx
SOph1sDWAR8GGuAzk71ShbYQneCbpKZ2S0xpo3ZIxFVM81imnADwgQMVe6s=
-----END CERTIFICATE-----
subject=/C=UA/ST=kiev/L=kyiv/O=New/OU=IT/CN=server.example.com/emailAdd
ress=ghost@example.com
```

Fig 4.10 ssl Certificate

```
SSL handshake has read 1567 bytes and written 431 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES256-GCM-SHA384
    Session-ID: 58E070957926795525F8B662B6B8E7664AE3AC5978F37FB0F01A6C4
BC877D5CA
    Session-ID-ctx:
    Master-Key: E7695FA775DE00B3D241CF07903FDC7D46EF1F834C53254C901D4BB
6130FAEE649A179BE02880B4D3019A68B3AB57419
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket lifetime hint: 300 (seconds)
    TLS session ticket:
    0000 - e4 d4 f7 c3 03 90 78 fd-80 ea 7b d7 0a e2 31 bd   ......x...
{...1.
    0010 - 61 f4 f9 aa 39 48 fa c5-32 c7 bf 90 9c b7 12 e6   a...9H..2.
......
    0020 - ed 64 00 0d 26 0d e5 29-d6 4d 78 c3 bb a1 53 01   .d..&..).M
x...S.
    0030 - 11 07 a9 e8 cb 93 5c 37-b2 47 69 66 48 0b 5f 81   ......\7.G
ifH._.
    0040 - 31 4f 95 b6 3a 5c f7 70-ef 37 7d e4 f6 c3 5b 1a   1O..:\.p.7
}...[.
    0050 - d3 c6 4e 5f 88 da ba 8a-89 d1 0c a5 8a 56 1f 51   ..N_......
```

Fig 4.11 TLS handshake

Using our packet sniffing tool, we are able to see packets going to our POP3 server but we are not able to decipher the contents of the emails being downloaded. All data re encrypted

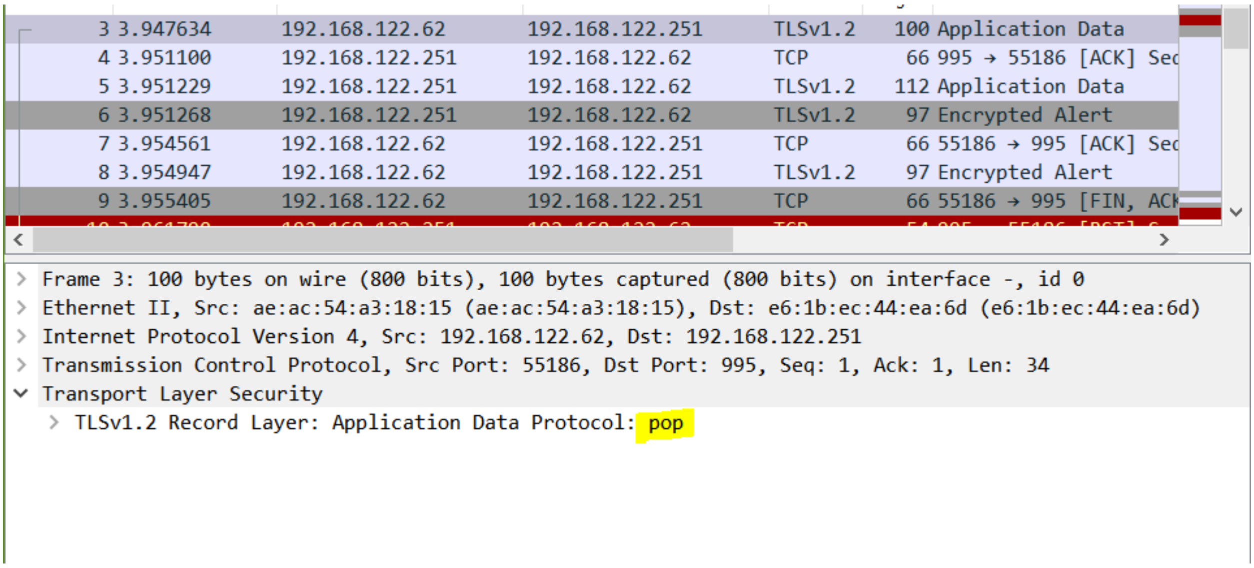


Fig 4.12 Protocol

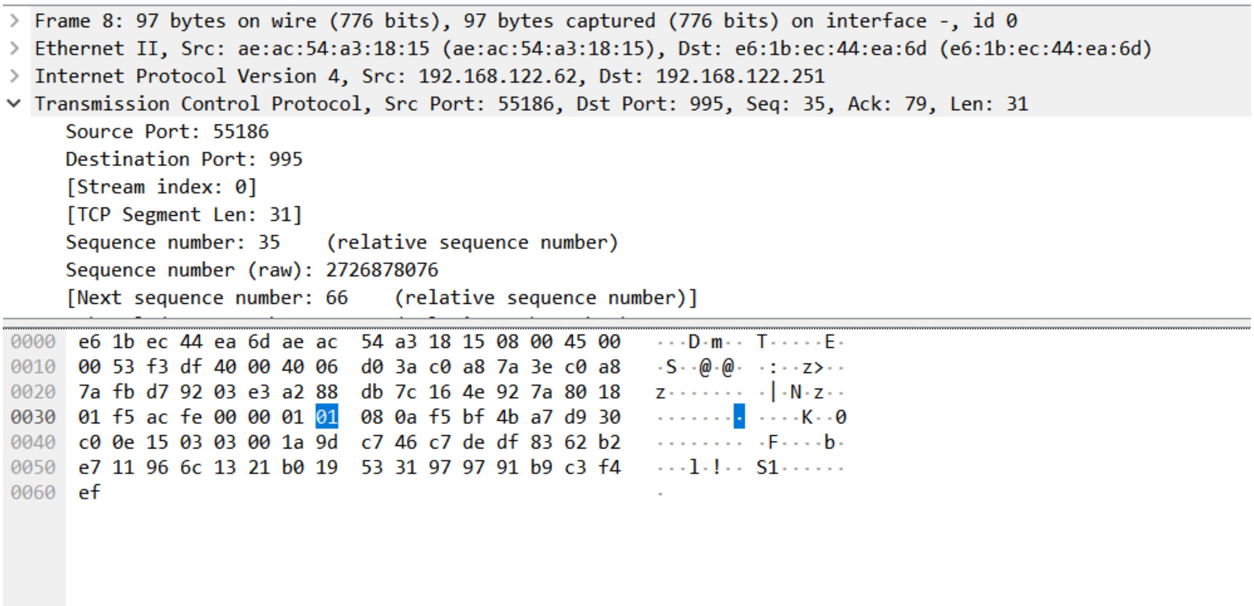


Fig 4.13 encrypted traffic

4.1.3 FTP TRAFFIC

FTP(file transfer protocol) runs on port 21. It is a file sharing service and on its own is no a secure way of file transfer and sharing.

To setup we add two linux nodes as clients and server. We install, the server software on the sever node and access it through the client node, we download some files and upload some and watch the unencrypted traffic flow. Also we introduce a NAT cloud node for internet access.

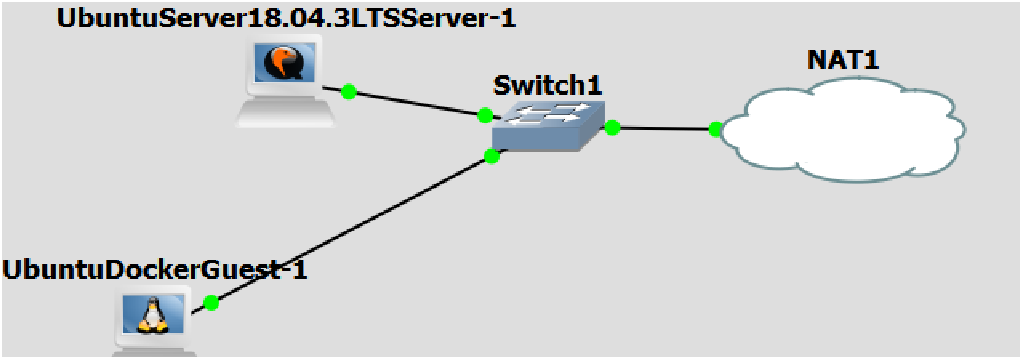


Fig 4.14 FTP topology

In the command line prompt of the server we type in the following command to install and configure FTP:

-sudo apt-get install vsftpd

In the /etc/vsftpd.conf file we edit the following line according to our liking.

anonymous_enable=NO # disable anonymous login

local_enable=YES # permit local logins

write_enable=YES # enable FTP commands which change the filesystem

local_umask=022 # value of umask for file creation for local users dirmessage_enable=YES # enable showing of messages when users first enter a new directory

xferlog_enable=YES # a log file will be maintained detailing uploads and downloads

connect_from_port_20=YES # use port 20 (ftp-data) on the server machine for PORT style connections

xferlog_std_format=YES # keep standard log file format

listen=NO # prevent vsftpd from running in standalone mode

listen_ipv6=YES # vsftpd will listen on an IPv6 socket instead of an IPv4 one

pam_service_name=vsftpd # name of the PAM service vsftpd will use

userlist_enable=YES # enable vsftpd to load a list of usernames

tcp_wrappers=YES # turn on tcp wrappers

We alo setup a chroot jail so users only have access to the directory and nowhere else on the system i.e users are restricted to their home directories.

chroot_local_user=YES

allow_writeable_chroot=YES

To login to the server we use the simple command

-ftp IP ADDRESS

We enter our name and password if enabled or anonymously.

After login we are able to Upload and download files to and from the server.

4.1.3.1 ANALYSIS OF FTP:

From our packet-sniffing tool, Wireshark, we see that all traffic is plain and can be accessed by anyone on the network with this tool.

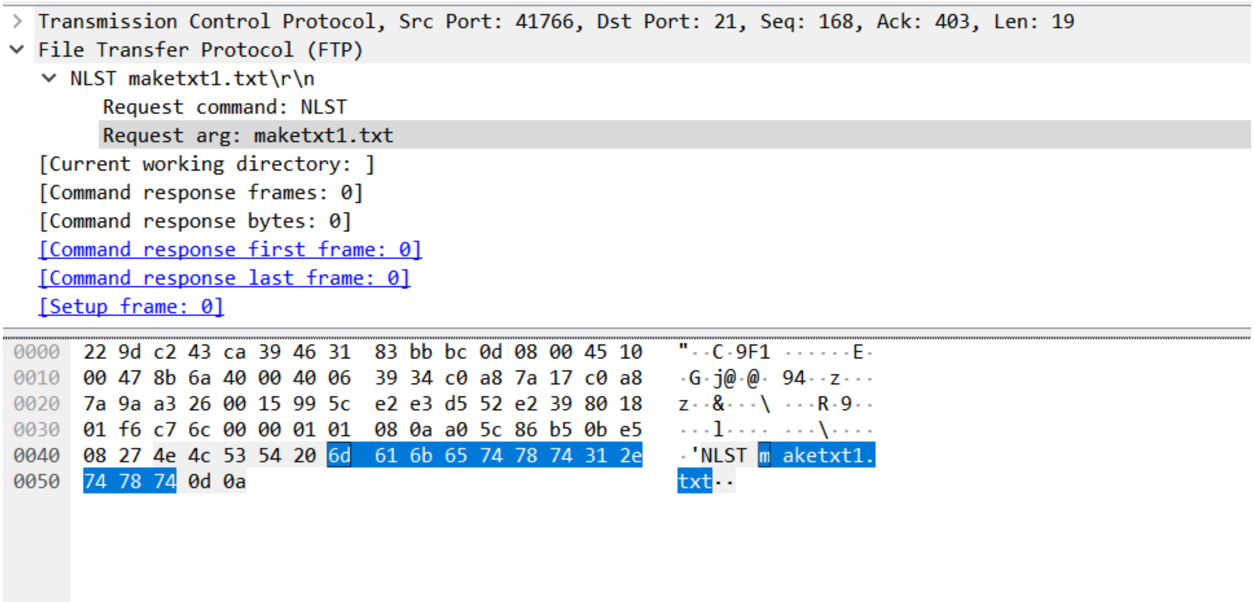


Fig 4.15 unencrypted ftp traffic

To get encrypted traffic we can either use SFTP or FTPS, for this project we have chosen to use SFTP. This is simply FTP over SSH connection.

To set this up firs we install the SSH server

-apt-get install openssh-server

Next we a directory to house our FTP data

-mkdir /sftp

-chmod 701 /sftp

The “chmod” command grants the necessary permissions for the directory.

Next we create a group for sftp users

-groupadd sftponly

We then add a user that doesn't have regular login privileges, but belongs to the newly created group 'sftponly'

-useradd –g sftponly –d /upload –s /sbin/nologin zed

Note that 'zed' is the user name we have chosen.

Next we give the user a password

-passwd zed

Now we create a directory specified to the new user and give the directory the proper permissions.

-mkdir –p /sftp/zed/upload

-chown –R root:sftponly /sftp/zed

-chown –R zed:sftponly /sftp/zed/upload

Now we configure our SSH daemon at /etc/ssh/sshd_config. At the bottom of the file, we add the following:

-Match Group sftponly

-ChrootDirectory /stftp/%u

-ForceCommand internal-sftp

Save the configuration file and the restart the ssh server.

Now on the client node we open the terminal and use the command:

-sftp IP ADDRESS

Fig 4.16 secure FTP connection

As seen above we enter our password and we are granted access to the server.

Using Wireshark this time we see that all commands, file and credentials are encrypted and cannot be accessed easily.

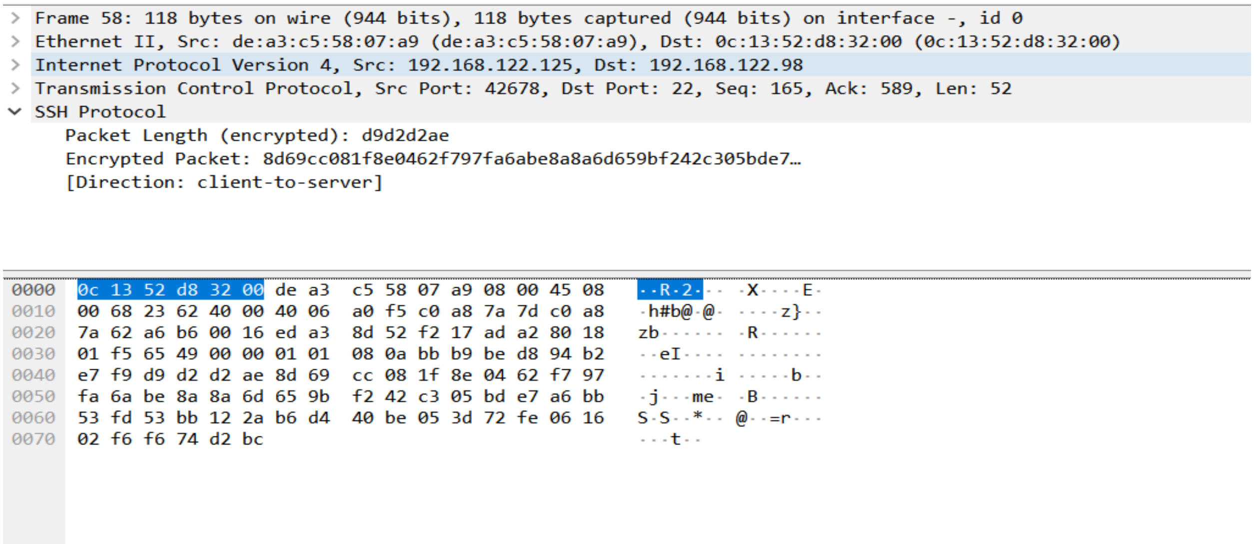


Fig 4.17 Encrypted SFTP traffic

4.1.4 DNS TRAFFIC

DNS stands for domain name system and it maps ip addresses to FQDN (fully qualified domain name)

DNS server software comes in different flavors: the most popular being bind. We have setup bind in out project to collect the necessary traffic.

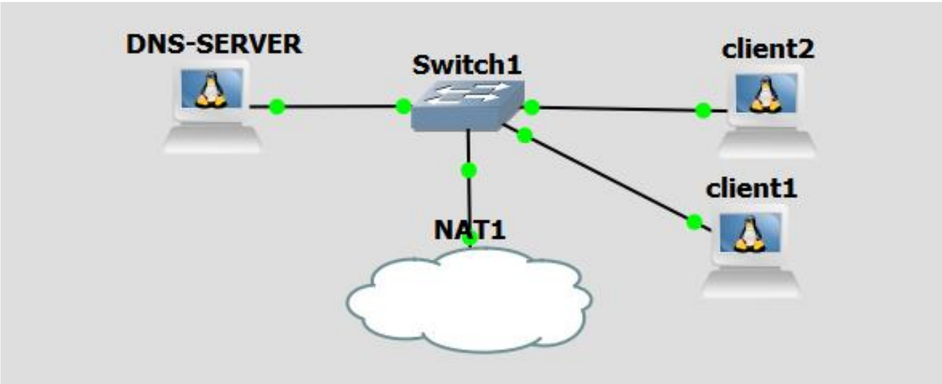


Fig 4.18 DNS network topology

To setup up the service in our server, we open a terminal and run the following command

-sudo apt-get install bind9 bind9utils

After installation, we proceed to configure our server as a primary master. Our domain is example.com.

We edit the /etc/named.conf.local file and add our forward zone (this translates domain names addresses to ip addresses) and our reverse zone (translates ip addresses to domain names).

We add the following lines

```
zone "example.com"{
        type master;
```

file “/etc/bind/forward.example.com”;

};

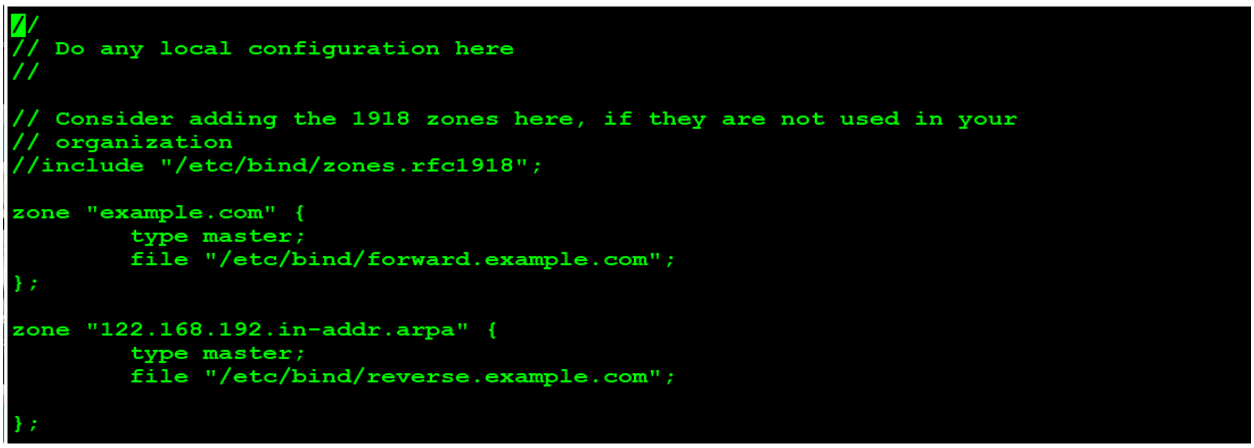
The above line is for the forward zone while below is for the reverse zone. In the reverse zone we write our network address in reverse and add “in-addr.arpa”.

zone “122.168.192.in-addr.arpa”{

type master;

file “/etc/bind/reverse.example.com”;

};



```
//
// Do any local configuration here
//

// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";

zone "example.com" {
        type master;
        file "/etc/bind/forward.example.com";
};

zone "122.168.192.in-addr.arpa" {
        type master;
        file "/etc/bind/reverse.example.com";

};
```

Fig 4.19 configuration file

We save the file and exit. Now we have to create the “forward.example.com” and “reverse.example.com” files in the /etc/bind directory. We do this by copying the template file.

-cd /etc/bind

-cp db.local forward.example.com

We then edit the file to look as below

```
;
; BIND data file for local loopback interface
;
$TTL    604800
@       IN      SOA     server.example.com. root.server.example.com. (
                              2         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      server.example.com.
@       IN      A       192.168.122.81
server  IN      A       192.168.122.81
client1 IN      A       192.168.122.254
client2 IN      A       192.168.122.85
```

Fig 4.20 forward zone

We make sure to '.' at the end of a domain name, this is very important or the dns server wont work.

Next we save and exit and create our "reverse.example.com" file same as we did for the forward zone file and edit it to look as below

```
;
; BIND data file for local loopback interface
;
$TTL    604800
@       IN      SOA     server.exmaple.com. root.server.example.com. (
                              2         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      server.example.com.
@       IN      PTR     example.com.
254     IN   PTR   client1.
85     IN   PTR   client2.
81     IN   PTR   server.example.com.
```

Fig 4.21 reverse zone

In reverse zones we set PTR or pointer records.

Next we check our that our configurations are correct using the following commands

-named-checkconf   -z /etc/bind/named.conf

```
root@DNS-SERVER:/etc/bind# named-checkconf -z /etc/bind/named.conf
zone example.com/IN: loaded serial 2
zone 122.168.192.in-addr.arpa/IN: loaded serial 2
zone localhost/IN: loaded serial 2
zone 127.in-addr.arpa/IN: loaded serial 1
zone 0.in-addr.arpa/IN: loaded serial 1
zone 255.in-addr.arpa/IN: loaded serial 1
```

Fig 4.22 Checking Configuration

Output should be as above.

To test we may ping any of the nodes or use nslookup or kdig tools to resolve hostnames.

We have use kdig

-kdig server.example.com

Our result from Wireshark shows that the traffic is unencrypted and we can see the server queries and responses.

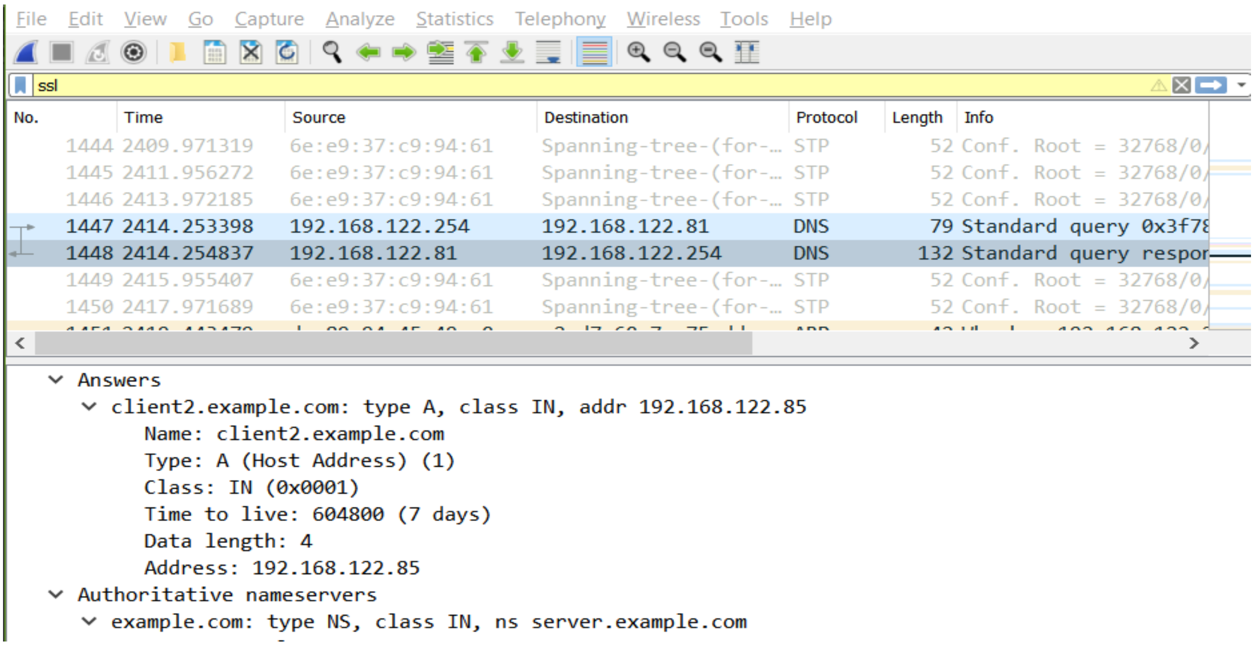


Fig 4.23 Plain DNS Traffic

This is observed for the reverse lookup as well.

-kdig –x IP ADDRESS

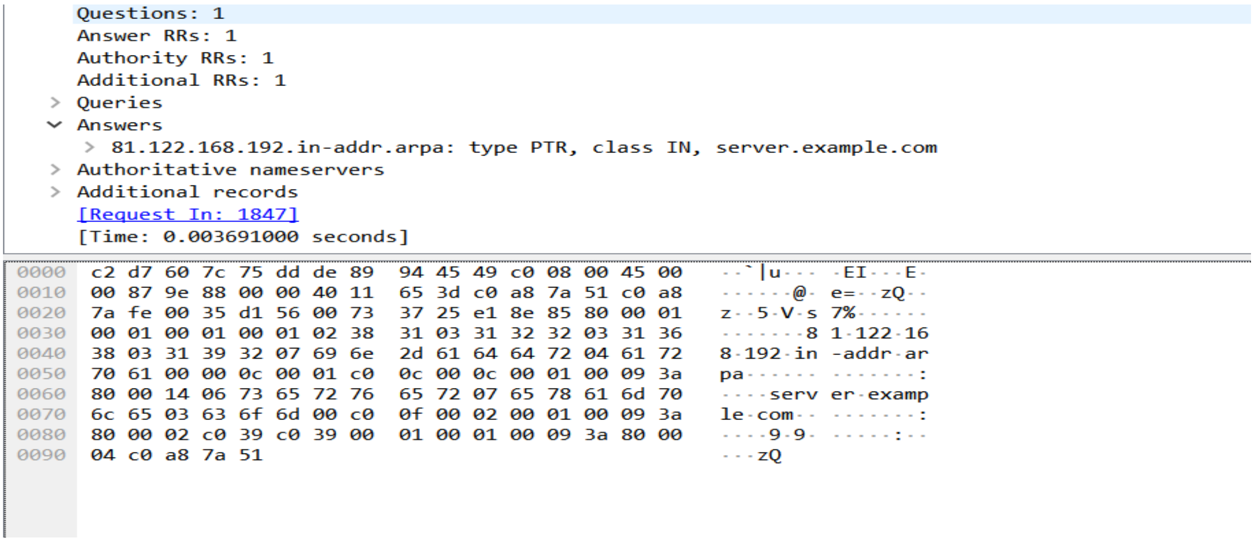


Fig 4.24 reverse lookup

To ensure traffic is encrypted we use a software called DNScrypt. We install it with the following command on the client node:

-sudo apt install dnscrypt-proxy

Next we configure our client to use one of many free public dnscrypt servers in the config file /etc/dnscrypt-proxy/dnscrypt-proxy.conf

-ResolverName random

We can choose a ResolverName from the list in the excel file located at /usr/share/dnscrypt-proxy/dnscrypt-resolvers.csv.

After this we save and restart the dnscrypt-proxy service.

From wireshark we see that the dns request are sent out as encrypted udp packets and ip addresses or domain names that are resolved cannot be seen.

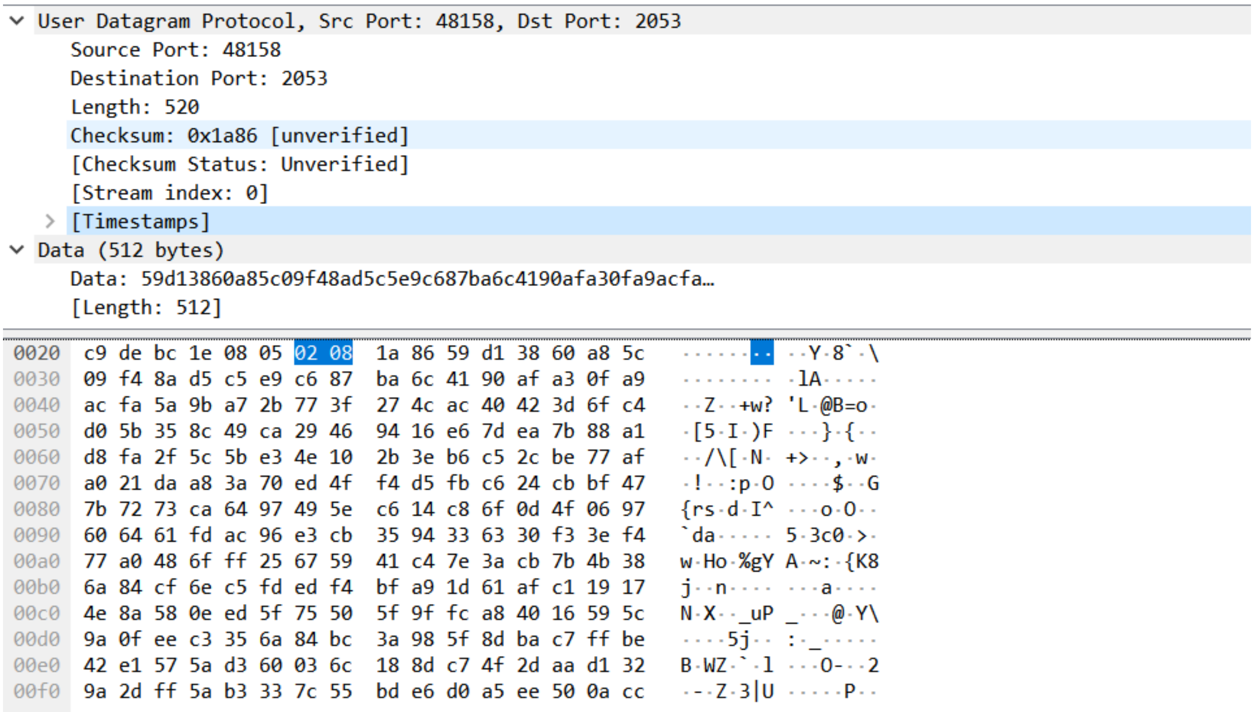


Fig 4.25 Encrypted DNS traffic

4.1.5BUILDING NEURAL NETWORK

As explaine in a previous chapeter in this paper. The same methos has been employed to build the CNN to classify the traffic we previously gathered although we have tweaked the code a bit and have ommitted and added in a few new libraries. We decide to use keras library.

Keras is an open source neural-network library written in Python. It can use any of the following libraries Tesnserflow (another neural-network library written in python), R, Microsoft Cognitive Toolkit, Theano in the backend. It is widey used for it simplicity and very user-friendly. Its primary authour is François Chollet, a Google engineer.

We will need a computer for to run this task on, a graphics card is most preferred of course as it can handle more task in lesser time.

Computational device used: Intel(R) Core(TM) i5-4200M CPU 2.50GHZ 12.0GB RAM

Libraries Used:

I. Os
II. Glob
III. Pandas
IV. Numpy
V. Functools
VI. Keafrs.models
VII. Keras.layers
VIII. Keras.utils
IX. Sklearn.model_selection

Editor: Visual studio code

We import our traffic data in raw format and preprocess it to machine understandable lanfuage so it can be fed into our neural network.

As seen in figure below:


```
5353482d322e302d4f70656e5353485f372e327032205562756e74752d347562756e7475322e38
5353482d322e302d4f70656e5353485f372e367031205562756e74752d347562756e7475302e33
000005340514ba45779800c38af6981f8b36ff242b35000000c4637572766532353531392d7368
2c656364682d736861322d6e697374703235362c656364682d736861322d6e697374703338342c
3532312c6469666669652d68656c6c6d616e2d67726f75702d65786368616e67652d7368613235
6e2d67726f75702d65786368616e67652d736861312c6469666669652d68656c6c6d616e2d6772
696e666f2d630000012265636473612d736861322d6e697374703235362d636572742d76303140
73612d736861322d6e697374703338342d636572742d763031406f70656e7373682e636f6d2c65
3532312d636572742d763031406f70656e7373682e636f6d2c65636473612d736861322d6e6973
322d6e697374703338342c65636473612d736861322d6e697374703532312c7373682d65643235
656e7373682e636f6d2c7373682d7273612d636572742d763031406f70656e7373682e636f6d2c
2d736861322d3531322c7273612d736861322d3235362c7373682d727361000000966368616368
```


Fig4.26 raw tcp payload.

We then create a function to create a label for our databased on the file name.

We split the date into two columns: data and label.

1000 bytes of the packet is collected, if the payload length is less than 1000 bytes we pad it with zeroes at the end. Then we convert it to integer’s and normalize.

Using the sklearn.model_selection library, we split the data into a training set and test set. We have decide to use 10% of the data for testing and training the remaining 90%.

Next the trainset and test set are reshaped to the correct tensor for the CNN.

Next we with the following code we build the neural network

```
model = Sequential()

model.add(Conv1D(512, strides=2, input_shape=X_train.shape[1:], activation=activation,
kernel_size=3, padding='same'))

model.add(MaxPooling1D())

model.add(Conv1D(256, strides=2, activation=activation, kernel_size=3, padding='same'))

model.add(MaxPooling1D())

model.add(Flatten())

model.add(Dense(128, activation=activation))

model.add(Dropout(0.5))

model.add(Dense(32, activation=activation))

model.add(Dropout(0.5))

model.add(Dense(num_classes, activation='softmax'))
print(model.summary())
```

print model summary gives us the summary of our model

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 500, 512)          2048
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 250, 512)          0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 125, 256)          393472
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 62, 256)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 15872)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               2031744
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                4128
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 198
=================================================================
```

Fig 4.27 Model summary

We can see all the layers involved in the CNN.

The trainset is trained for 50 epochs and our result at the end is quite satisfactory considering the fact that out data isn't a lot.

```
Epoch 46/50
165/165 [==============================] - 2s 14ms/step - loss: 0.4595 - accuracy: 0.8000 -
: 0.9927 - val_accuracy: 0.7381
Epoch 47/50
165/165 [==============================] - 3s 15ms/step - loss: 0.4182 - accuracy: 0.8000 -
: 0.8451 - val_accuracy: 0.7381
Epoch 48/50
165/165 [==============================] - 2s 15ms/step - loss: 0.3989 - accuracy: 0.8121 -
: 0.8181 - val_accuracy: 0.7381
Epoch 49/50
165/165 [==============================] - 2s 14ms/step - loss: 0.3856 - accuracy: 0.8182 -
: 0.8848 - val_accuracy: 0.7381
Epoch 50/50
165/165 [==============================] - 2s 14ms/step - loss: 0.4121 - accuracy: 0.8242 -
```

Fig 4.28 Result of Model

We had a loss of 0.4 and an accuracy of 0.82 that is to show that the model correctly classified the traffic in the test set as whether it is encrypted or not.

4.2 CONCLUSION

This thesis encompasses encrypted network traffic classification. Loads of techniques and methods have been proposed and used by different researches to classify network traffic. Different programming languages tools and libraries have been employed as well. While some have proven to work with great accuracy to with some drawbacks such as amount of data that can be processed, some have been unreliable.

Many statistical and machine-based learning methods have been applied to the task of traffic classification. Despite this, there are no conclusive results to show which method has the best properties. The main reason is that the results depend heavily on the data sets used and the configuration of the methods. Our results show that most of the authors use private data sets, sometimes in combination with public ones. Most of the methods use supervised or semi-supervised machine learning algorithms to classify flows and even determine the application protocol of a given flow.

This paper discusses a simple yet effective method using the convolutional neural network. It is highly accurate and reliable for both large dataset and few dataset. The greatest advantage will be that it requires not as much computational power. Of course the larger the data set the larger the computational power needed as is the case for other techniques. As compared to other techniques, our mode uses less overhead.

Also this paper has further introduced us into another area of information technology, that is, machine learning. Our knowledge has been broadened and it has sparked more interest in the aforementioned field. It has show that machine learning, deep learning to be more specific can be used to solve majority of tasks with or without human intervention and get a high accuracy on problems.

REFERENCES

[1]. K.Muthamil Sudar, P.Deepalakshmi http://www.ijitee.org/wpcontent/uploads/papers/v9i2s2/B11081292S219.pdf

[2]. Mohammad Lotfollahi1 ·Mahdi JafariSiavoshani1 ·Ramin ShiraliHosseinZade1· Mohammdsadegh Saberian1 M (2017) Deep packet: a novel approach for encrypted traffic classification using deep learning. CoRR abs/1709.02656. arXiv:1709.02656

[3]. SHERAZ NASEER1,2, YASIR SALEEM1, SHEHZAD KHALID3, MUHAMMAD KHAWAR BASHIR1,4, JIHUN HAN5, MUHAMMAD MUNWAR IQBAL 6, AND KIJUN HAN https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8438865

[4]. https://www.shaileshjha.com/step-by-step-install-vmware-workstation-player-12-in-windows-10/

[5]. Sumit Saha

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[6].https://www.geeksforgeeks.org/image-classifier-using-cnn/

[7]. T. Karagiannis, A. Broido, M. Faloutsos, and K. C. Claffy, "Transport layer identification of P2P traffic," in 4th ACM Special Interest Group on Data Communication Internet Measurement Conf. 2004, Taormina, Italy, pp. 121-134, October 2004.

[8]. K. Xu, Z.-L. Zhang, and S. Bhattacharyya, "Profiling Internet backbone traffic: Behavior models and applications," ACM SIGCOMM Comput. Commun. Rev., vol. 35, no. 4, pp. 169–180, 2005

[9]. M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese, “Network monitoring using traffic dispersion graphs (TDGs),” in Proc. Internet Measurement Conf. 2007, San Diego, CA, pp. 315-320, 2007.

[10]. Y. Jin, N. Duffield, P. Haffner, S. Sen, and Z.-L. Zhang, “Inferring applications at the network layer using collective traffic statistics,” SIGMETRICS Perform. Eval. Rev., vol. 38, p. 1-8, June 2010.

[11]. J. Levandoski, E. Sommer, and M. Strait, , “Application layer packet classifier for Linux”, 2008

[12]. Al Khater, N., & Overill, R. E. (2016). Network Traffic Classification Techniques and Challenges. In The 10thInternational Conference on Digital Information Management, ICDIM 2015 (pp. 43-48). [7381869] Institute of Electrical and Electronics Engineers Inc.. https://doi.org/10.1109/ICDIM.2015.738186

[13]Y. Kumano, S. Ata, N. Nakamura, Y. Nakahira, and I Oka.Towards real-time processing for application identification of encrypted traffic. In Computing, Networking and Communications (ICNC), 2014 International Conference on, pages 136–140, Feb 2014.

[14]Y. Okada, S. Ata, N. Nakamura, Y. Nakahira, and I Oka. Application Identification from Encrypted Traffic Based on Characteristic Changes by Encryption. In Communications Qual

[15]. R. Alshammari and A.N. Zincir-Heywood. A Flow Based Approach for SSH Traffic Detection. In Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on, pages 296–301, Oct 2007.

[16]. R. Alshammari and A.N. Zincir-Heywood. A Preliminary Performance Comparison of Two Feature Sets for Encrypted Traffic Classification. In Proceedings of the International Workshop on Computational Intelligence in Security for Information Systems CISIS'08, volume 53 of Advances in Soft Computing, pages 203–210. Springer Berlin Heidelberg, 2009.

[17]. R. Alshammari and A.N. Zincir-Heywood. Machine learning based encrypted traffic classification: Identifying SSH and Skype. In Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on, pages 1–8, July 2009.

[18]. R. Alshammari and A.N. Zincir-Heywood. An Investigation on the Identification of VoIP traffic: Case study on Gtalk and Skype. In Network and Service Management (CNSM), 2010 International Conference on, pages 310–313, Oct 2010.

[19]. R. Alshammari and A.N. Zincir-Heywood. Can encrypted traffic be identified without port numbers, IP addresses and payload inspection? Computer Networks, 55(6):1326 – 1350, 2011.

[20]. P.V. Amoli and T. Hamalainen. A Real Time Unsupervised NIDS for Detecting Unknown and Encrypted Network Attacks in High Speed Network. In Measurements and Networking Proceedings (M N), 2013 IEEE International Workshop on, pages 149–154, Oct 2013.

[21]. D.J. Arndt and A.N. Zincir-Heywood. A Comparison of Three Machine Learning Techniques for Encrypted Network Traffic Analysis. In Computational Intelligence for Security and Defense Applications (CISDA), 2011IEEE Symposium on, pages 107–114, April 2011.

[22]. Azureus Software Inc. Message Stream Encryption.Vuze Wiki. Web page, May 2014. Accessed: 2014-10-31.

[23]. C. Bacquet, A.N. Zincir-Heywood, and M.I. Heywood. An Investigation of Multi-objective Genetic Algorithmsfor Encrypted Traffic Identification. InComputational Intelligence in Security for Information Systems, volume 63ofAdvances in Intelligent and Soft Computing, pages 93–100. Springer Berlin Heidelberg, 2009.

[24]. C. Bacquet, A.N. Zincir-Heywood, and M.I. Heywood. Genetic Optimization and Hierarchical Clustering Appliedto Encrypted Traffic Identification. InComputational Intelligence in Cyber Security (CICS), 2011 IEEE Symposiumon, pages 194–201, April 2011.

[25]. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (ProposedStandard), August 2008. Updated by RFCs 5746, 5878, 617

[26]. A. Freier, P. Karlton, and P. Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101 (Historic),August 2011.

[27]. D. Harrison. Index of BitTorrent Enhancement Proposals. Web page, October 2014. Accessed: 2014-11-27.

[28]. Skype and Microsoft. Skype. Web page, 2014. Accessed: 2014-11-25

[29]. Petr Velan, Milanˇ Cerm´ak, Pavelˇ Celeda, Martin Draˇ sar A Survey of Methods for Encrypted TrafficClassification and Analysis INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT. 2014

APPENDIX

```
import os

import glob

import pandas as pd

import numpy as np

from functools import partial

from keras.models import Sequential

from keras.layers import Flatten, Conv1D, MaxPooling1D, Dropout,
Dense

from keras.utils import to_categorical

from sklearn.model_selection import train_test_split



LABELS = {}

counter = iter(range(20))


def pad_and_convert(s):

    """Collect 1000 bytes from packet payload. If payload length is less
than

    1000 bytes, pad zeroes at the end. Then convert to integers and
normalize."""

    if len(s) < 2000:

        s += '00' * (2000-len(s))
```

```
    else:
        s = s[:2000]
    return [float(int(s[i]+s[i+1], 16)/255) for i in range(0, 2000, 2)]


def read_file(f, label):
    df = pd.read_csv(f, index_col=None, header=0)
    df.columns = ['data']
    df['label'] = label
    return df


def preprocess(path):
    files = glob.glob(os.path.join(path, '*.txt'))
    list_ = []
    for f in files:
        label = f.split('/')[-1].split('.')[0]
        LABELS[label] = next(counter)
        labelled_df = partial(read_file, label=LABELS[label])
        list_.append(labelled_df(f))
    df = pd.concat(list_, ignore_index=True)
    return df


def main():
```

```
activation = 'relu'

df = preprocess('Dataset')

df['data'] = df['data'].apply(pad_and_convert)

num_classes = len(LABELS)

X_train, X_test, y_train, y_test = train_test_split(df['data'], df['label'],
                                                    test_size=0.2, random_state=4)

X_train = X_train.apply(pd.Series)

X_test = X_test.apply(pd.Series)

X_train = X_train.values.reshape(X_train.shape[0], X_train.shape[1],
1)

X_test = X_test.values.reshape(X_test.shape[0], X_test.shape[1], 1)

y_train = to_categorical(y_train, num_classes)

y_test = to_categorical(y_test, num_classes)


model = Sequential()

model.add(Conv1D(512, strides=2, input_shape=X_train.shape[1:],
activation=activation, kernel_size=3, padding='same'))

model.add(MaxPooling1D())

model.add(Conv1D(256, strides=2, activation=activation,
kernel_size=3, padding='same'))

model.add(MaxPooling1D())

model.add(Flatten())
```

```
    model.add(Dense(128, activation=activation))
    model.add(Dropout(0.5))
    model.add(Dense(32, activation=activation))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))
    print(model.summary())

    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    result = model.fit(X_train, y_train, verbose=1, epochs=50,
batch_size=16, validation_data=(X_test, y_test))

if __name__ == '__main__':
    main()
```

```
Model: "sequential_1"
_________________________________________________________________
_
Layer (type)                 Output Shape              Param #
=================================================================
========
conv1d_1 (Conv1D)            (None, 500, 512)          2048
```

____

_

max_pooling1d_1 (MaxPooling1 (None, 250, 512)   0

____

_

flatten_1 (Flatten)   (None, 15872)   0

____

_

dense_1 (Dense)   (None, 128)   2031744

____

_

dropout_1 (Dropout)   (None, 128)   0

____

_

dense_2 (Dense)   (None, 32)   4128

____

_

dropout_2 (Dropout)   (None, 32)   0

____

_

dense_3 (Dense)   (None, 6)   198

==
========

Total params: 2,431,590

Trainable params: 2,431,590

Non-trainable params: 0

_

None

2020-05-17 15:19:58.631635: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX AVX2

WARNING:tensorflow:From C:\Users\mrmal\anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead. nstructions that this TensorFlow binary

Train on 165 samples, validate on 42 samples py:422: The name tf.global_variables is

Epoch 1/50

165/165 [==============================] - 5s 33ms/step - loss: 1.6361 - accuracy: 0.4667 - val_loss: 1.2403 - val_accuracy: 0.5952

Epoch 2/50

165/165 [==============================] - 2s 15ms/step - loss: 1.4579 - accuracy: 0.5030 - val_loss: 1.2403 - val_accuracy: 0.5952: 1.2154 - val_accuracy: 0.5952

Epoch 3/50 : 1.2154 - val_accuracy: 0.5952

165/165 [==============================] - 2s 14ms/step - loss: 1.1707 - accuracy: 0.5576 - val_loss: 0.9637 - val_accuracy: 0.5714

: 0.9637 - val_accuracy: 0.5714

Epoch 4/50

165/165 [==============================] - 3s 17ms/step - loss: 0.9292 - accuracy: 0.6242 - val_loss: 0.8868 - val_accuracy: 0.6667

Epoch 5/50

165/165 [==============================] - 2s 14ms/step - loss: 0.9267 - accuracy: 0.6303 - val_loss: 0.9174 - val_accuracy: 0.6667

Epoch 6/50

165/165 [==============================] - 2s 14ms/step - loss: 0.9967 - accuracy: 0.6727 - val_loss: 0.8263 - val_accuracy: 0.7381

Epoch 7/50

165/165 [==============================] - 2s 15ms/step - loss: 0.9703 - accuracy: 0.6242 - val_loss: 0.8300 - val_accuracy: 0.7143

Epoch 8/50

165/165 [==============================] - 2s 15ms/step - loss: 0.8513 - accuracy: 0.7152 - val_loss: 0.7695 - val_accuracy: 0.7381

Epoch 9/50

165/165 [==============================] - 2s 14ms/step - loss: 0.9150 - accuracy: 0.7030 - val_loss: 0.7679 - val_accuracy: 0.7381

Epoch 10/50

165/165 [==============================] - 2s 14ms/step - loss: 0.8018 - accuracy: 0.7212 - val_loss: 0.8819 - val_accuracy: 0.7381

Epoch 11/50

165/165 [==============================] - 2s 14ms/step - loss: 0.7590 - accuracy: 0.6970 - val_loss: 0.7850 - val_accuracy: 0.7381

Epoch 12/50

165/165 [==============================] - 2s 14ms/step - loss: 0.7062 - accuracy: 0.7333 - val_loss: 0.8899 - val_accuracy: 0.7381

Epoch 13/50

165/165 [==============================] - 2s 15ms/step - loss: 0.7052 - accuracy: 0.7333 - val_loss: 0.8268 - val_accuracy: 0.7143

Epoch 14/50

165/165 [==============================] - 2s 15ms/step - loss: 0.6922 - accuracy: 0.7758 - val_loss: 0.8693 - val_accuracy: 0.7381

Epoch 15/50

165/165 [==============================] - 2s 14ms/step - loss: 0.6684 - accuracy: 0.7697 - val_loss: 0.8862 - val_accuracy: 0.7381

Epoch 16/50

165/165 [==============================] - 2s 14ms/step - loss: 0.5658 - accuracy: 0.8061 - val_loss: 0.8328 - val_accuracy: 0.7143

Epoch 17/50

165/165 [==============================] - 2s 14ms/step - loss: 0.7480 - accuracy: 0.7273 - val_loss: 0.8232 - val_accuracy: 0.7381

Epoch 18/50

165/165 [==============================] - 2s 14ms/step - loss: 0.6390 - accuracy: 0.7636 - val_loss: 0.8177 - val_accuracy: 0.7381

Epoch 19/50

165/165 [==============================] - 2s 14ms/step - loss: 0.5732 - accuracy: 0.7697 - val_loss: 0.8565 - val_accuracy: 0.7381

Epoch 20/50

165/165 [==============================] - 2s 15ms/step - loss: 0.6116 - accuracy: 0.7697 - val_loss: 0.7191 - val_accuracy: 0.7381

Epoch 21/50

165/165 [==============================] - 2s 15ms/step - loss: 0.5427 - accuracy: 0.7818 - val_loss: 0.8175 - val_accuracy: 0.7381

Epoch 22/50

165/165 [==============================] - 2s 14ms/step - loss: 0.5769 - accuracy: 0.8000 - val_loss: 0.7019 - val_accuracy: 0.7143

Epoch 23/50

165/165 [==============================] - 2s 14ms/step - loss: 0.5820 - accuracy: 0.7515 - val_loss: 0.8283 - val_accuracy: 0.7381

Epoch 24/50

165/165 [==============================] - 2s 14ms/step - loss: 0.5724 - accuracy: 0.7576 - val_loss: 0.8428 - val_accuracy: 0.7381

Epoch 25/50

165/165 [==============================] - 2s 14ms/step - loss: 0.5623 - accuracy: 0.7818 - val_loss: 0.7535 - val_accuracy: 0.7619

Epoch 26/50

165/165 [==============================] - 2s 14ms/step - loss: 0.4812 - accuracy: 0.7879 - val_loss: 0.9690 - val_accuracy: 0.7381

Epoch 27/50

165/165 [==============================] - 2s 15ms/step - loss: 0.4294 - accuracy: 0.8182 - val_loss: 0.9311 - val_accuracy: 0.7381

Epoch 28/50

165/165 [==============================] - 2s 15ms/step - loss: 0.5142 - accuracy: 0.7939 - val_loss: 0.9224 - val_accuracy: 0.7381

Epoch 29/50

165/165 [==============================] - 2s 14ms/step - loss: 0.5728 - accuracy: 0.7333 - val_loss: 0.6957 - val_accuracy: 0.7619

Epoch 30/50

165/165 [==============================] - 2s 14ms/step - loss: 0.5463 - accuracy: 0.7818 - val_loss: 0.8702 - val_accuracy: 0.7143

Epoch 31/50

165/165 [==============================] - 2s 14ms/step - loss: 0.4888 - accuracy: 0.8242 - val_loss: 0.8081 - val_accuracy: 0.7143

Epoch 32/50

165/165 [==============================] - 2s 14ms/step - loss: 0.4386 - accuracy: 0.8182 - val_loss: 0.9840 - val_accuracy: 0.7381

Epoch 33/50

165/165 [==============================] - 2s 14ms/step - loss: 0.4776 - accuracy: 0.8000 - val_loss: 0.7868 - val_accuracy: 0.7619

Epoch 34/50

165/165 [==============================] - 2s 15ms/step - loss: 0.5190 - accuracy: 0.8303 - val_loss: 0.7083 - val_accuracy: 0.7381

Epoch 35/50

165/165 [==============================] - 2s 15ms/step - loss: 0.4598 - accuracy: 0.8364 - val_loss: 0.9478 - val_accuracy: 0.7381

Epoch 36/50

165/165 [==============================] - 2s 14ms/step - loss: 0.4021 - accuracy: 0.8606 - val_loss: 0.8667 - val_accuracy: 0.6905

Epoch 37/50

165/165 [==============================] - 2s 14ms/step - loss: 0.4988 - accuracy: 0.7697 - val_loss: 1.0474 - val_accuracy: 0.7381

Epoch 38/50

165/165 [==============================] - 2s 15ms/step - loss: 0.4747 - accuracy: 0.7939 - val_loss: 0.8448 - val_accuracy: 0.7381

Epoch 39/50

165/165 [==============================] - 2s 15ms/step - loss: 0.4013 - accuracy: 0.8182 - val_loss: 0.8952 - val_accuracy: 0.7381

Epoch 40/50

165/165 [==============================] - 3s 16ms/step - loss: 0.4490 - accuracy: 0.7879 - val_loss: 0.9596 - val_accuracy: 0.7619

Epoch 41/50

165/165 [==============================] - 3s 16ms/step - loss: 0.4169 - accuracy: 0.8182 - val_loss: 0.9806 - val_accuracy: 0.7619

Epoch 42/50

165/165 [==============================] - 2s 15ms/step - loss: 0.3729 - accuracy: 0.8303 - val_loss: 1.0120 - val_accuracy: 0.7619

Epoch 43/50

165/165 [==============================] - 2s 14ms/step - loss: 0.3787 - accuracy: 0.8303 - val_loss: 1.1354 - val_accuracy: 0.7619

Epoch 44/50

165/165 [==============================] - 2s 14ms/step - loss: 0.3989 - accuracy: 0.8061 - val_loss: 1.0433 - val_accuracy: 0.7381

Epoch 45/50

165/165 [==============================] - 2s 14ms/step - loss: 0.4072 - accuracy: 0.8364 - val_loss: 1.1868 - val_accuracy: 0.7381

Epoch 46/50

165/165 [==============================] - 2s 14ms/step - loss: 0.4595 - accuracy: 0.8000 - val_loss: 0.9927 - val_accuracy: 0.7381

Epoch 47/50

165/165 [==============================] - 3s 15ms/step - loss: 0.4182 - accuracy: 0.8000 - val_loss: 0.8451 - val_accuracy: 0.7381

Epoch 48/50

165/165 [==============================] - 2s 15ms/step - loss: 0.3989 - accuracy: 0.8121 - val_loss: 0.8181 - val_accuracy: 0.7381

Epoch 49/50

165/165 [==============================] - 2s 14ms/step - loss: 0.3856 - accuracy: 0.8182 - val_loss: 0.8848 - val_accuracy: 0.7381

Epoch 50/50

165/165 [==============================] - 2s 14ms/step - loss: 0.4121 - accuracy: 0.8242 - val_loss: 1.1089 - val_accuracy: 0.7381